Innovations in Computer Science 2010

Game Theory with Costly Computation: Formulation and Application to Protocol Security

Joseph Y. Halpern[*] Rafael Pass[†]
Cornell University
halpern@cs.cornell.edu rafael@cs.cornell.edu

Abstract: We develop a general game-theoretic framework for reasoning about strategic agents performing possibly costly computation. In this framework, many traditional game-theoretic results (such as the existence of a Nash equilibrium) no longer hold. Nevertheless, we can use the framework to provide psychologically appealing explanations to observed behavior in well-studied games (such as finitely repeated prisoner's dilemma and rock-paper-scissors). Furthermore, we provide natural conditions on games sufficient to guarantee that equilibria exist. As an application of this framework, we develop a definition of protocol security relying on game-theoretic notions of implementation. We show that a natural special case of this this definition is equivalent to a variant of the traditional cryptographic definition of protocol security; this result shows that, when taking computation into account, the two approaches used for dealing with "deviating" players in two different communities—*Nash equilibrium* in game theory and *zero-knowledge "simulation"* in cryptography—are intimately related.

Keywords: game theory, costly computation, cryptography

1 Introduction

Consider the following game. You are given a random odd n-bit number x and you are supposed to decide whether x is prime or composite. If you guess correctly you receive \$2, if you guess incorrectly you instead have to pay a penalty of \$1000. Additionally you have the choice of "playing safe" by giving up, in which case you receive \$1. In traditional game theory, computation is considered "costless"; in other words, players are allowed to perform an unbounded amount of computation without it affecting their utility. Traditional game theory suggests that you should compute whether x is prime or composite and output the correct answer; this is the only Nash equilibrium of the one-person game, no matter what n (the size of the prime) is. Although for small n this seems reasonable, when n grows larger most people would probably decide to "play safe"—as eventually the cost of computing the answer (e.g., by buying powerful enough computers) outweighs the possible gain of \$1.

The importance of considering such computational issues in game theory has been recognized since at least the work of Simon [1]. There have been a number of attempts to capture various aspects of computation. Two major lines of research can be identified. The first line, initiated by Neyman [2], tries to model the fact that players can do only bounded computation, typically by modeling players as finite automata. Neyman focused on finitely repeated prisoner's dilemma, a topic which has contined to attract attention. (See [3] and the references therein for more recent work on prisoner's dilemma played by finite automata; Megiddo and Wigderson [4] considered prisoner's dilemma played by Turing machines.) In another instance of this line of research, Dodis, Halevi, and Rabin [5] and Urbano and Vila [6] consider the problem of implementing mediators when players are polynomial-time Turing machines. The second line, initiated by Rubinstein [7], tries to capture the fact that doing costly computation affects an agent's utility. Rubinstein assumed that players choose a finite automaton to play the game rather than choosing a strategy directly; a player's utility depends both on the move made by the automaton and the complexity of the automaton (identified with the number of states of the automaton). Intuitively, automata that use more states are seen as representing more complicated procedures. (See [8] for an overview of the work in this area in the 1980s, and [9] for more recent work.)

Our focus is on providing a general game-theoretic framework for reasoning about agents performing costly computation. As in Rubinstein's work, we view players as choosing a machine, but for us the machine

[*]Supported in part by NSF grants ITR-0325453, IIS-0534064, IIS-0812045, and IIS-0911036, and by AFOSR grants FA9550-08-1-0438 and FA9550-09-1-0266, and ARO grant W911NF-09-1-0281.

[†]Supported in part by a Microsoft New Faculty Fellowship, NSF CAREER Award CCF-0746990, AFOSR Award FA9550-08-1-0197, and BSF Grant 2006317.

is a Turing machine, rather than a finite automaton. We associate a complexity, not just with a machine, but with the machine and its input. The complexity could represent the running time of or space used by the machine on that input. The complexity can also be used to capture the complexity of the machine itself (e.g., the number of states, as in Rubinstein's case) or to model the cost of searching for a new strategy to replace one that the player already has. For example, if a mechanism designer recommends that player i use a particular strategy (machine) M, then there is a cost for searching for a better strategy; switching to another strategy may also entail a psychological cost. By allowing the complexity to depend on the machine *and* the input, we can deal with the fact that machines run much longer on some inputs than on others. A player's utility depends both on the actions chosen by all the players' machines and the complexity of these machines.

In this setting, we can define Nash equilibrium in the obvious way. However, as we show by a simple example (a rock-paper-scissors game where randomization is costly), a Nash equilibrium may not always exist. Other standard results in the game theory, such as the *revelation principle* (which, roughly speaking, says that there is always an equilibrium where players truthfully report their types, i.e., their private information [10, 11]) also do not hold. (See the full paper.) We view this as a feature. We believe that taking computation into account should force us to rethink a number of basic notions.

First, we show that the non-existence of Nash equilibrium is not such a significant problem. We identify natural conditions (such as the assumption that randomizing is free) that guarantee the existence of Nash equilibrium in computational games. Moreover, we demonstrate that our computational framework can explain experimentally-observed phenomena, such as cooperation in the finitely repeated prisoner's dilemma, that are inconsistent with standard Nash equilibrium in a psychologically appealing way.

We also show that our framework is normative in that it can be used to tell a mechanism designer how to design a mechanism that takes into account agents' computational limitations. We illustrate this point in the context of cryptographic protocols. We use our framework to provide a game-theoretic definition of what it means for a communication protocol to *securely compute* [12] a function of players' inputs. Rather than stipulating that "malicious" players cannot harm the "honest" ones (as traditionally done in cryptographic definitions [12, 13]), we say that a protocol Π is a secure implementation of function f if, given *any* computational game G for which it is an equilibrium for the players to provide their inputs to a mediator computing f and output what the mediator recommends, running Π on the same inputs is also an equilibrium. In other words, whenever a set of parties *want* to compute f on their inputs, they also want to honestly run the protocol Π using the same inputs. Perhaps surprisingly, we show that, under weak restrictions on the utility functions of the players (essentially, that players never prefer to compute more), our notion of implementation is equivalent to a variant of the cryptographic notion of secure computation. This result shows that the two approaches used for dealing with "deviating" players in two different communities—*Nash equilibrium* in game theory, and *zero-knowledge "simulation"* [13] in cryptography—are intimately connected once we take cost of computation into account.

Finally, we believe this work leaves open a huge number of exciting research questions. We outline several directions in Section 5. To give just one example, under natural restrictions on the utility function of players, we can circumvent classical impossibility results with respect to secure computation (e.g., [14]), suggesting an exploration of more game-theoretic (im)possibility results.

2 A Computational Framework

In a Bayesian game, it is implicitly assumed that computing a strategy—that is, computing what move to make given a type—is free. We want to take the cost of computation into account here. To this end, we consider what we call *Bayesian machine games*, where we replace strategies by *machines*. For definiteness, we take the machines to be Turing machines, although the exact choice of computing formalism is not significant for our purposes. Given a type, a strategy in a Bayesian game returns a distribution over actions. Similarly, given as input a type, the machine returns a distribution over actions. Just as we talk about the expected utility of a strategy profile in a Bayesian game, we can talk about the expected utility of a machine profile in a Bayesian machine game. However, we can no longer compute the expected utility by just taking the expectation over the action profiles that result from playing the game. A player's utility depends not only on the type profile and action profile played by the machine, but also on the "complexity" of the machine given an input. The complexity of a machine can represent, for example, the running time or space usage of the machine on that input, the size of the program description, or some combination of these factors. For simplicity, we describe the complexity by a single number, although, since a number of factors may be relevant, it may be more appropriate to represent it by a tuple of numbers in some cases. (We can, of course, always encode the tuple as a single number, but in that case, "higher" complexity is not necessarily worse.) Note that when determining player i's utility, we consider the complexity of all machines in the profile, not just that of i's machine. For example, i might be happy as long as his machine takes fewer steps than j's.

We assume that nature has a type in $\{0,1\}^*$. While there is no need to include a type for nature in standard Bayesian games (we can effectively incorporate nature's type into the type of the players), once we take computation into account, we obtain a more expressive class of games by allowing nature to have a type (since the complexity of computing the utility may depend on nature's type). We assume that machines take as input strings of 0s and 1s and output strings of 0s and 1s. Thus, we assume that both types and actions can be represented as elements of $\{0,1\}^*$. We allow machines to randomize, so given a type as input, we actually get a distribution over strings. To capture this, we assume that the input to a machine is not only a type, but also a string chosen with uniform probability from $\{0,1\}^\infty$ (which we can view as the outcome of an infinite sequence of coin tosses). The machine's output is then a deterministic function of its type and the infinite random string. We use the convention that the output of a machine that does not terminate is a fixed special symbol ω. We define a *view* to be a pair (t,r) of two bitstrings; we think of t as that part of the type that is read, and r is the string of random bits used. (Our definition is slightly different from the traditional way of defining a view, in that we include only the parts of the type and the random sequence *actually* read. If computation is not taken into account, there is no loss in generality in including the full type and the full random sequence, and this is what has traditionally been done in the literature. However, when computation is costly, this might no longer be the case.) We denote by $t;r$ a string in $\{0,1\}^*;\{0,1\}^*$ representing the view. (Note that here and elsewhere, we use ";" as a special symbol that acts as a separator between strings in $\{0,1\}^*$.) If $v=(t;r)$ and r is a finite string, we take $M(v)$ to be the output of M given input type t and random string $r\cdot 0^\infty$.

We use a *complexity function* $\mathscr{C}$: $\mathbf{M}\times\{0,1\}^*;\{0,1\}^*\cup\{0,1\}^\infty\rightarrow I\!N$, where $\mathbf{M}$ denotes the set of Turing machines to describe the complexity of a machine given a view. If $t\in\{0,1\}^*$ and $r\in\{0,1\}^\infty$, we identify $\mathscr{C}(M,t;r)$ with $\mathscr{C}(M,t;r')$, where r' is the finite prefix of r actually used by M when running on input t with random string r.

For now, we assume that machines run in isolation, so the output and complexity of a machine does not depend on the machine profile. We remove this restriction in the next section, where we allow machines to communicate with mediators (and thus, implicitly, with each other via the mediator).

Definition 2.1 (Bayesian machine game) *A Bayesian machine game G is described by a tuple $([m],\mathcal{M},T,\Pr,\mathscr{C}_1,\ldots,\mathscr{C}_m,u_1,\ldots,u_m)$, where*

- *$[m]=\{1,\ldots,m\}$ is the set of players;*
- *$\mathcal{M}$ is the set of possible machines;*
- *$T\subseteq(\{0,1\}^*)^{m+1}$ is the set of type profiles, where the $(m+1)$st element in the profile corresponds to nature's type;*
- *$\Pr$ is a distribution on T;*
- *$\mathscr{C}_i$ is a complexity function;*
- *$u_i: T\times(\{0,1\}^*)^m\times I\!N^m\rightarrow I\!R$ is player i's utility function. Intuitively, $u_i(\vec{t},\vec{a},\vec{c})$ is the utility of player i if $\vec{t}$ is the type profile, $\vec{a}$ is the action profile (where we identify i's action with M_i's output), and $\vec{c}$ is the profile of machine complexities.*

We can now define the expected utility of a machine profile. Given a Bayesian machine game $G=([m],\mathcal{M},\Pr,T,\vec{\mathscr{C}},\vec{u})$ and $\vec{M}\in\mathcal{M}^m$, define the random variable $u_i^{G,\vec{M}}$ on $T\times(\{0,1\}^\infty)^m$ (i.e., the space of type profiles and sequences of random strings) by taking

$$\begin{aligned}u_i^{G,\vec{M}}(\vec{t},\vec{r}) \quad = \quad & u_i(\vec{t},M_1(t_1;r_1),\ldots,M_m(t_m;r_m),\\ & \mathscr{C}_1(M_1,t_1;r_1),\ldots,\mathscr{C}_m(M_m,t_m)).\end{aligned}$$

Note that there are two sources of uncertainty in computing the expected utility: the type t and realization of the random coin tosses of the players, which is an element of $(\{0,1\}^\infty)^k$. Let $\Pr_U^k$ denote the uniform distribution on $(\{0,1\}^\infty)^k$. Given an arbitrary distribution $\Pr_X$ on a space X, we write $\Pr_X^{+k}$ to denote the distribution $\Pr_X\times\Pr_U^k$ on $X\times(\{0,1\}^\infty)^k$. If k is clear from context or not relevant, we often omit it, writing $\Pr_U$ and $\Pr_X^+$. Thus, given the probability $\Pr$ on T, the expected utility of player i in game G if $\vec{M}$ is used is the expectation of the random variable $u_i^{G,\vec{M}}$ with respect to the distribution $\Pr^+$ (technically, $\Pr^{+m}$): $U_i^G(\vec{M})=\mathbf{E}_{\Pr^+}[u_i^{G,\vec{M}}]$. Note that this notion of utility allows an agent to prefer a machine that runs faster to one that runs slower, even if they give the same output, or to prefer a machine that has faster running time to one that gives a better output. Because we allow the utility to depend on the whole profile of complexities, we can capture a situation where i can be "happy" as long as his machine runs faster than j's machine. Of course, an important special case is where i's utility depends only on his own complexity. All of our results continue to hold if we make this restriction.

2.1 Nash Equilibrium in Machine Games

Given the definition of utility above, we can now define (ϵ-) Nash equilibrium in the standard way.

Definition 2.2 (Nash equilibrium in machine games) *Given a Bayesian machine game $G = ([m], \mathcal{M}, T, \Pr, \vec{\mathscr{C}}, \vec{u})$, a machine profile $\vec{M} \in \mathcal{M}^m$, and $\epsilon \geq 0$, M_i is an ϵ-best response to $\vec{M}_{-i}$ if, for every $M_i' \in \mathcal{M}$,*

$$U_i^G[(M_i, \vec{M}_{-i})] \geq U_i^G[(M_i', \vec{M}_{-i})] - \epsilon.$$

*(As usual, $\vec{M}_{-i}$ denotes the tuple consisting of all machines in $\vec{M}$ other than M_i.) $\vec{M}$ is an ϵ-*Nash equilibrium *of G if, for all players i, M_i is an ϵ-best response to $\vec{M}_{-i}$. A* Nash equilibrium *is a 0-Nash equilibrium.*

There is an important conceptual point that must be stressed with regard to this definition. Because we are implicitly assuming, as is standard in the game theory literature, that the game is common knowledge, we are assume that the agents understand the costs associated with each Turing machine. That is, they do not have to do any "exploration" to compute the costs. In addition, we do not charge the players for computing which machine is the best one to use in a given setting; we assume that this too is known. This model is appropriate in settings where the players have enough experience to understand the behavior of all the machines on all relevant inputs, either through experimentation or theoretical analysis. We can easily extend the model to incorporate uncertainty, by allowing the complexity function to depend on the state (type) of nature as well as the machine and the input.

One immediate advantage of taking computation into account is that we can formalize the intuition that ϵ-Nash equilibria are reasonable, because players will not bother changing strategies for a gain of ϵ. Intuitively, the complexity function can "charge" ϵ for switching strategies. Specifically, an ϵ-Nash equilibrium $\vec{M}$ can be converted to a Nash equilibrium by modifying player i's complexity function to incorporate the overhead of switching from M_i to some other strategy, and having player i's utility function decrease by $\epsilon' > \epsilon$ if the switching cost is nonzero; we omit the formal details here. Thus, the framework lets us incorporate explicitly the reasons that players might be willing to play an ϵ-Nash equilibrium. This justification of ϵ-Nash equilibrium seems particularly appealing when designing mechanisms (e.g., cryptographic protocols) where the equilibrium strategy is made "freely" available to the players (e.g., it is accessible on a web-page), but any other strategy requires some implementation cost.

Although the notion of Nash equilibrium in Bayesian machine games is defined in the same way as Nash equilibrium in standard Bayesian games, the introduction of complexity leads to some significant differences in their properties. We highlight a few of them here. First, note that our definition of a Nash equilibrium considers *behavioral strategies*, which in particular might be randomized. It is somewhat more common in the game-theory literature to consider *mixed strategies*, which are probability distributions over deterministic strategies. As long as agents have perfect recall, mixed strategies and behavioral strategies are essentially equivalent [15]. However, in our setting, since we might want to charge for the randomness used in a computation, such an equivalence does not necessarily hold.

Mixed strategies are needed to show that Nash equilibrium always exists in standard Bayesian games. As the following example shows, since we can charge for randomization, a Nash equilibrium may not exist in a Bayesian machine game, even in games where the type space and the output space are finite.

Example 2.3 (Rock-paper-scissors) Consider the 2-player Bayesian game of roshambo (rock-paper-scissors). Here the type space has size 1 (the players have no private information). We model playing rock, paper, and scissors as playing 0, 1, and 2, respectively. The payoff to player 1 of the outcome (i, j) is 1 if $i = j \oplus 1$ (where $\oplus$ denotes addition mod 3), -1 if $j = i \oplus 1$, and 0 if $i = j$. Player 2's playoffs are the negative of those of player 1; the game is a zero-sum game. As is well known, the unique Nash equilibrium of this game has the players randomizing uniformly between 0, 1, and 2.

Now consider a machine game version of roshambo. Suppose that we take the complexity of a deterministic strategy to be 1, and the complexity of strategy that uses randomization to be 2, and take player i's utility to be his payoff in the underlying Bayesian game minus the complexity of his strategy. Intuitively, programs involving randomization are more complicated than those that do not randomize. With this utility function, it is easy to see that there is no Nash equilibrium. For suppose that (M_1, M_2) is an equilibrium. If M_1 uses randomization, then 1 can do better by playing the deterministic strategy $j \oplus 1$, where j is the action that gets the highest probability according to M_2 (or is the deterministic choice of player 2 if M_2 does not use randomization). Similarly, M_2 cannot use randomization. But it is well known (and easy to check) that there is no equilibrium for roshambo with deterministic strategies. (Of course, there is nothing special about the costs of 1 and 2 for deterministic vs. randomized strategies. This argument works as long as all three deterministic strategies have the same cost, and it is less than that of a randomized strategy.) It is well known that people have difficulty simulating randomization; we can think of the cost for randomizing as capturing this difficulty. Interestingly, there are roshambo tournaments (indeed, even a Rock Paper Scissors World Championship), and books written on roshambo strategies [16]. Championship players are clearly not randomizing uniformly (they could not hope to get a higher payoff than an opponent by randomizing). Our framework provides a psychologically plausible account of this lack of randomization. ∎

The non-existence of Nash euqilibrium does not depend on charging directly for randomization; it suffices that it be difficult to compute the best randomized response. Consider a variant of roshambo where we do not charge for the first, say, 10,000 steps of computation, and after that there is a positive cost of computation. It is not hard to show that, in finite time, using coin tossing with equal likelihood of heads and tails, we cannot exactly compute a uniform distribution over the three choices, although we can approximate it closely.[1] Since there is always a deterministic best reply that can be computed quickly ("play rock", "play scissors", or "play paper"), from this observation it easily follows, as above, that there is no Nash equilibrium in this game either.

In these examples, computing an appropriate randomized response was expensive, either because we charged directly for randomizing, or because getting the "right" response required a lot of computation. It turns out that the cost of randomization is the essential roadblock to getting a Nash equilibrium in Bayesian machine games. We make this precise in the full version of the paper; here we provide only an informal statement of the result.

Theorem 2.4 (Informally stated) *Every finite Baysian machine game, where (1) the utility functions and the type distribution are computable, and (2) "randomization is free", has a Nash equilibrium.*

As an intermediate result, we first establish the following "computational" analog of Nash's Theorem, which is of interest in its own right.

[1]Consider a probabilistic Turing machine M with running time bounded by T that outputs either 0, 1, or 2. Since M's running time is bounded by T, M can use at most T of its random bits. If M outputs 0 for m of the 2^T strings of length T, then its probability of outputting 0 is $m/2^T$, and cannot be 1/3.

Theorem 2.5 (Informally stated) *Every finite Bayesian game where the utility functions and the type distribution are computable has a Nash equilibrium that can be implemented by a probabilistic Turing machine.*

As noted above, no Turing machine with *bounded* running time can implement the Nash equilibrium strategy even for roshambo; the Turing machine in Theorem 2.5 has, in general, unbounded running time. We remark that the question of whether Nash equilibrium strategies can be "implemented" was considered already in Nash's original paper (where he showed the existence of games where the Nash equilibrium strategies require players to samples actions with irrational probabilities). Binmore [17] considered the question of whether Nash equilibrium strategies can be implemented by Turing machines, but mostly presented impossibility results.

Our proof relies on results from the first-order theory of reals guaranteeing the existence of solutions to certain equations in particular models of the theory [18]; similar techniques were earlier used by Lipton and Markakis [19] and Papadimitriou and Roughgarden [20] in a related context.

In the full version, we present several other differences between Nash equilibrium in traditional games and in Bayesian machine games. In particular, we show several examples where our framework can be used to give simple (and, arguably, natural) explanations to experimentally observed behavior in classical games where traditional game-theoretic solution concepts fail. For instance, we show that in a variant of the *finitely repeated prisonner's dilemma (FRPD)* where players are charged for the use of memory, the strategy *tit-for-tat*—which does exceedingly well in experiments [21]—is also a Nash equilibrium, whereas it is dominated by defecting in the classical sense. (This follows from the facts that (1) any profitable deviation from tit-for-tat requires the use of memory, and (2) if future utility is discounted, then for sufficiently long games, the potential gain of deviating becomes smaller than the cost of memory.)

3 A Game-Theoretic Definition of Security

The traditional cryptographic definition of secure computation considers two types of players: *honest* and *malicious*. Roughly speaking, a protocol is secure if (1) the malicious players cannot influence the output of the computation any more than they could have by selecting a different input; this is called *correctness*, and (2) the malicious players cannot "learn" more than what can be efficiently computed (i.e., in polynomial-time) from only the output of the function; this is called *privacy* and is formalized through the zero-knowledge simulation paradigm [13]. Note that this definition does not guarantee that honest players actually want to execute the protocol with their intended inputs.

By way of contrast, this property is considered in the game-theoretic notion of *implementation* (see [11, 22]), while privacy and correctness are not required. Roughly speaking, the game-theoretic notion of implementation says that a strategy profile $\vec{\sigma}$ implements a mediator if, as long as it is a Nash equilibrium for the players to tell the mediator their type and output what the mediator recommends, then $\vec{\sigma}$ is a Nash equilibrium in the "cheap talk" game (where the players just talk to each other, rather than talking to a mediator) that has the same distribution over outputs as when the players talk to the mediator. In other words, whenever a set of parties have incentives to tell the mediator their inputs, they also have incentives to honestly use $\vec{\sigma}$ using the same inputs, and get the same distribution over outputs in both cases.

There have been several recent works that attempt to bridge these two notions (e.g., [5, 23–28])). Most notably, Izmalkov, Lepinski and Micali [26] (see also [27]) present a "hybrid" definition, where correctness is defined through the game-theoretic notion,[2] and privacy through the zero-knowledge paradigm. We suggest a different approach, based on the game-theoretic approach.

Roughly speaking, we say that Π implements a mediator $\mathcal{F}$ if for all games G that use $\mathcal{F}$ for which (the utilities in G are such that) it is an equilibrium for the players to truthfully tell $\mathcal{F}$ their inputs, running Π on the same set of inputs (and with the same utility functions) is also an equilibrium and produces the same distribution over outputs as $\mathcal{F}$.[3] Note that by requiring the implementation to work for all games, not only do we ensure that players have proper incentives to execute protocols with their intended input, even if they consider computation a costly resource, but we get the privacy and correctness requirements "for free". For suppose that, when using Π, some information about i's input is revealed to j. We consider a game G where a player j gains some significant utility by having this information. In this game, i will not want to use Π. However, our notion of implementation requires that, even with the utilities in G, i should want to use Π if i is willing to use the mediator $\mathcal{F}$. (This argument depends on the fact that we consider games where computation is costly; the fact that j gains information about i's input may mean that j can do some computation faster with this information than without it.) As a consequence, our definition gives a relatively simple (and strong) way of formalizing the security of protocols, relying only on basic notions from game theory.

To make these notions precise, we need to introduce some extensions to our game-theoretic framework. Specifically, we will be interested only in equilibria that are robust in a certain sense, and we want equilibria that deal with deviations by coalitions, since the security literature allows malicious players that deviate in a coordinated way.

Computationally Robust Nash Equilibrium Computers get faster, cheaper, and more powerful every year. Since utility in a Bayesian machine game takes computational complexity into account, this suggests that an agent's utility function will change when he replaces one computer by a newer computer. We are thus interested in *robust* equilibria, intuitively, ones that continue to be equilibria (or, more precisely, ϵ-equilibria for some appropriate ϵ) even if agents' utilities change as a result of upgrading computers.

Definition 3.1 (Computationally robust NE) *Let $p : I\!N \to I\!N$. The complexity function $\mathscr{C}'$ is* at most a p-speedup *of the complexity function $\mathscr{C}$ if, for all machines M and views v,*

$$\mathscr{C}'(M,v) \le \mathscr{C}(M,v) \le p(\mathscr{C}'(M,v)).$$

Game $G' = ([m'], \mathcal{M}', \Pr', \vec{\mathscr{C}'}, \vec{u'})$ is at most a p-speedup *of game $G = ([m], \mathcal{M}, \Pr, \vec{\mathscr{C}}, \vec{u})$ if $m' = m$, $\Pr = \Pr'$ and $\vec{u} = \vec{u'}$ (i.e., G' and G' differ only in their complexity and machine profiles), and $\mathscr{C}'_i$ is at most a p-speedup of $\mathscr{C}_i$, for $i = 1, \dots, m$. M_i is a* p-robust ϵ-best response *to $\vec{M}_{-i}$ in G, if for every game $\tilde{G}$ that is at most a p-speedup of G, M_i is an ϵ-best response to $\vec{M}_{-i}$. $\vec{M}$ is a* p-robust ϵ-equilibrium *for G if, for all i, M_i is an p-robust ϵ-best response to $\vec{M}_{-i}$.*

Intuitively, if we think of complexity as denoting running time and $\mathscr{C}$ describes the running time of machines (i.e., programs) on an older computer, then $\mathscr{C}'$ describes the running time of machines on an upgraded computer. For instance, if the upgraded computer runs at most twice as fast as the older one (but never slower), then $\mathscr{C}'$ is $\bar{2}$-speedup of $\mathscr{C}$, where $\bar{k}$ denotes the constant function k. Clearly, if $\vec{M}$ is a Nash equilibrium of G, then it is a $\bar{1}$-robust equilibrium. We can think of p-robust equilibrium as a refinement

[2]In fact, Izmalkov, Lepinski, and Micali [26] consider an even stronger notion of implementation, which they call *perfect implementation*. See Section 3.1 for more details.

[3]While the definitions of implementation in the game-theory literature (e.g., [11, 22]) do not stress the uniformity of the implementation—that is, the fact that it works for all games—the implementations provided are in fact uniform in this sense.

of Nash equilibrium for machine games, just like *sequential equilibrium* [29] or *perfect equilibrium* [30]; it provides a principled way of ignoring "bad" Nash equilibria. Note that in games where computation is free, every Nash equilibrium is also computationally robust.

Coalition Machine Games We strengthen the notion of Nash equilibrium to allow for deviating coalitions. Towards this goal, we consider a generalization of Bayesian machine games called *coalition machine games*, where, in the spirit of *coalitional games* [31], each *subset* of players has a complexity function and utility function associated with it. In analogy with the traditional notion of Nash equilibrium, which considers only "single-player" deviations, we consider only "single-coalition" deviations. Roughly speaking, given a set $\mathcal{Z}$ of subsets of $[m]$, $\vec{M}$ is a *$\mathcal{Z}$-safe Nash equilibrium* for G if, for all $Z \in \mathcal{Z}$, a coalition controlling players in Z does not want to deviate. See Appendix A for the formal definitions.

Machine Games with Mediators We extend Bayesian machine games to allow for communication with a trusted mediator. A Bayesian machine game with a mediator is a pair $(G, \mathcal{F})$, where G is a Bayesian machine game (but where the machines involved now are *interactive* TMs) and $\mathcal{F}$ is a mediator. A particular mediator of interest is the *communication mediator*, denoted comm, which corresponds to what cryptographers call *authenticated channels* and economists call *cheap talk*. We defer the details to the full version.

3.1 A Computational Notion of Game-Theoretic Implementation

In this section we extend the traditional notion of game-theoretic implementation of mediators to consider computational games. Our aim is to obtain a notion of implementation that can be used to capture the cryptographic notion of secure computation. For simplicity, we focus on implementations of mediators that receive a single message from each player and return a single message to each player (i.e., the mediated games consist only of a single stage).

We provide a definition that captures the intuition that the machine profile $\vec{M}$ implements a mediator $\mathcal{F}$ if, whenever a set of players want to to truthfully provide their "input" to the mediator $\mathcal{F}$, they also want to run $\vec{M}$ using the same inputs. To formalize "whenever", we consider what we call *canonical coalition games*, where each player i has a type t_i of the form $x_i; z_i$, where x_i is player i's intended "input" and z_i consists of some additional information that player i has about the state of the world. We assume that the input x_i has some fixed length n. Such games are called *canonical games of input length n*.[4]

Let $\Lambda^{\mathcal{F}}$ denote the machine that, given type $t = x; z$ sends x to the mediator $\mathcal{F}$ and outputs as its action whatever string it receives back from $\mathcal{F}$, and then halts. (Technically, we model the fact that $\Lambda^{\mathcal{F}}$ is expecting to communicate with $\mathcal{F}$ by assuming that the mediator $\mathcal{F}$ appends a signature to its messages, and any messages not signed by $\mathcal{F}$ are ignored by $\Lambda^{\mathcal{F}}$.) Thus, the machine $\Lambda^{\mathcal{F}}$ ignores the extra information z. Let $\vec{\Lambda}^{\mathcal{F}}$ denote the machine profile where each player uses the machine $\Lambda^{\mathcal{F}}$. Roughly speaking, to capture the fact that whenever the players want to compute $\mathcal{F}$, they also want to run $\vec{M}$, we require that if $\vec{\Lambda}^{F}$ is an equilibrium in the game $(G, \mathcal{F})$ (i.e., if it is an equilibrium to simply provide the intended input to $\mathcal{F}$ and finally output whatever $\mathcal{F}$ replies), running $\vec{M}$ using the intended input is an equilibrium as well.

We actually consider a more general notion of implementation: we are interested in understanding how well equilibrium in a set of games with mediator $\mathcal{F}$ can be implemented using a machine profile $\vec{M}$ and a possibly different mediator $\mathcal{F}'$. Roughly speaking, we want that, for every game G in some set $\mathcal{G}$ of games, if $\vec{\Lambda}^{\mathcal{F}}$ is an equilibrium in $(G, \mathcal{F})$, then $\vec{M}$ is an equilibrium in $(G, \mathcal{F}')$. In particular, we want to understand what degree of robustness p in the game $(G, \mathcal{F})$ is required to achieve an ϵ-equilibrium in the game $(G, \mathcal{F}')$. We also require that the equilibrium with mediator $\mathcal{F}'$ be as "coalition-safe" as the equilibrium with mediator $\mathcal{F}$.

Definition 3.2 (Universal implementation) *Suppose that $\mathcal{G}$ is a set of m-player canonical games, $\mathcal{Z}$ is a set of subsets of $[m]$, $\mathcal{F}$ and $\mathcal{F}'$ are mediators, $M_1, \ldots, M_m$ are interactive machines, $p : I\!N \times I\!N \to I\!N$, and $\epsilon : I\!N \to I\!R$. $(\vec{M}, \mathcal{F}')$ is a $(\mathcal{G}, \mathcal{Z}, p)$-universal implementation of $\mathcal{F}$ with error ϵ if, for all $n \in I\!N$, all games $G \in \mathcal{G}$ with input length n, and all $\mathcal{Z}' \subseteq \mathcal{Z}$, if $\vec{\Lambda}^{\mathcal{F}}$ is a $p(n, \cdot)$-robust $\mathcal{Z}'$-safe Nash equilibrium in the mediated machine game $(G, \mathcal{F})$ then*

1. (Preserving Equilibrium) *$\vec{M}$ is a $\mathcal{Z}'$-safe $\epsilon(n)$-Nash equilibrium in the mediated machine game $(G, \mathcal{F}')$.*
2. (Preserving Action Distributions) *For each type profile $\vec{t}$, the action profile induced by $\vec{\Lambda}^{\mathcal{F}}$ in $(G, \mathcal{F})$ is identically distributed to the action profile induced by $\vec{M}$ in $(G, \mathcal{F}')$.*

[4]Note that by simple padding, canonical games represent a setting where all parties' input lengths are upper-bounded by some value n that is common knowledge. Thus, we can represent any game where there are only finitely many possible types as a canonical game for some input length n.

As we have observed, although our notion of universal implementation does not explicitly consider the privacy of players' inputs, it can nevertheless capture privacy requirements.

Note that, depending on the class $\mathcal{G}$, our notion of universal implementation imposes severe restrictions on the complexity of the machine profile $\vec{M}$. For instance, if $\mathcal{G}$ consists of all games, it requires that the complexity of $\vec{M}$ is the same as the complexity of $\vec{\Lambda}^{\mathcal{F}}$. (If the complexity of $\vec{M}$ is higher than that of $\vec{\Lambda}^{\mathcal{F}}$, then we can easily construct a game G by choosing the utilities appropriately such that it is an equilibrium to run $\vec{\Lambda}^{\mathcal{F}}$ in $(G, \mathcal{F})$, but running $\vec{M}$ is too costly.) Also note that if $\mathcal{G}$ consists of games where players strictly prefer smaller complexity, then universal implementation requires that $\vec{M}$ be the optimal algorithm (i.e., the algorithm with the lowest complexity) that implements the functionality of $\vec{M}$, since otherwise a player would prefer to switch to the optimal implementation. Since few algorithms algorithms have been shown to be provably optimal with respect to, for example, the number of computational steps of a Turing machines, this, at first sight, seems to severely limit the use of our definition. However, if we consider games with "coarse" complexity functions where, say, the first T steps are "free" (e.g., machines that execute less than T steps are assigned complexity 1), or n^2 computational steps count as one unit of complexity, the restrictions above are not so severe. Indeed, it seems quite natural to assume that a player is indifferent between "small" differences in computation in such a sense.

Our notion of universal implementation is related to a number of earlier notions of implementation. We now provide a brief comparison to the most relevant ones.

- Our definition of universal implementation captures intuitions similar in spirit to Forges' [22] notion of a *universal mechanism.* It differs in one obvious way: our definition considers *computational* games, where the utility functions depend on complexity considerations. Dodis, Halevi and Rabin [5] (and more recent work, such as [23, 23–25, 27, 32]) consider notions of implementation where the players are modeled as polynomially-bounded Turing machines, but do not consider computational games. As such, the notions considered in these works do not provide any *a priori* guarantees about the incentives of players with regard to computation.
- Our definition is more general than earlier notions of implementation in that we consider universality with respect to (sub-)classes $\mathcal{G}$ of games, and allow deviations by coalitions.
- Our notion of coalition-safety also differs somewhat from earlier related notions. Note that if $\mathcal{Z}$ contains all subsets of players with k or less players, then universal implementation implies that all k-resilient Nash equilibria and all strong k-resilient Nash equilibria are preserved. However, unlike the notion of k-resilience considered by Abraham et al. [23, 33], our notion provides a "best-possible" guarantee for games that do not have a k-resilient Nash equilibrium. We guarantee that if a certain subset Z of players have no incentive to deviate in the mediated game, then that subset will not have incentive to deviate in the cheap-talk game; this is similar in spirit to the definitions of [26, 32]. Note that, in contrast to [26, 34], rather than just allowing colluding players to communicate only through their moves in the game, we allow coalitions of players that are controlled by a single entity; this is equivalent to considering collusions where the colluding players are allowed to freely communicate with each other. In other words, whereas the definitions of [26, 34] require protocols to be "signalling-free", our definition does not impose such restrictions. We believe that this model is better suited to capturing the security of cryptographic protocols in most traditional settings (where signalling is not an issue).
- We require only that a Nash equilibrium is preserved when moving from the game with mediator $\mathcal{F}$ to the communication game. Stronger notions of implementation require that the equilibrium in the communication game be a *sequential equilibrium* [29]; see, for example, [35, 36]. Since every Nash equilibrium in the game with the mediator $\mathcal{F}$ is also a sequential equilibrium, these stronger notions of implementation actually show that sequential equilibrium is preserved when passing from the game with the mediator to the communication game.
  While these notions of implementation guarantee that an equilibrium with the mediator is preserved in the communication game, they do not guarantee that new equilibria are not introduced in the latter. An even stronger guarantee is provided by Izmalkov, Lepinski, and Micali's [26] notion of *perfect implementation*; this notion requires a one-to-one correspondence f between *strategies* in the corresponding games such that each player's utility with strategy profile $\vec{\sigma}$ in the game with the mediator is the same as his utility with strategy profile $(f(\sigma_1), \ldots, f(\sigma_n))$ in the communication game without the mediator. Such a correspondence, called *strategic equiva-*

lence by Izmalkov, Lepinski, and Micali [26], guarantees (among other things) that *all* types of equilibria are preserved when passing from one game to the other, and that no new equilibria are introduced in the communication game. However, strategic equivalence can be achieved only with the use of strong primitives, which cannot be implemented under standard computational and systems assumptions [34]. We focus on the simpler notion of implementation, which requires only that Nash equilibria are preserved, and leave open an exploration of more refined notions.

Strong Universal Implementation Intuitively, $(\vec{M}, \mathcal{F}')$ universally implements $\mathcal{F}$ if, whenever a set of parties want to use $\mathcal{F}$ (i.e., it is an equilibrium to use $\vec{\Lambda}^{\mathcal{F}}$ when playing with $\mathcal{F}$), then the parties also want to run $\vec{M}$ (using $\mathcal{F}'$) (i.e., using $\vec{M}$ with $\mathcal{F}'$ is also an equilibrium). We now strengthen this notion to also require that whenever a subset of the players do *not* want to use $\mathcal{F}$ (specifically, if they prefer to do "nothing"), then they also do not want to run $\vec{M}$, even if all other players do so. We use $\perp$ to denote the special machine that sends no messages and writes nothing on the output tape.

Definition 3.3 (Strong Universal Implementation) *Let* $(\vec{M}, \mathcal{F}')$ *be a* $(\mathcal{G}, \mathcal{Z}, p)$-universal implementation of $\mathcal{F}$ with error ϵ. $(\vec{M}, \mathcal{F}')$ *is a* strong $(\mathcal{G}, \mathcal{Z}, p)$-implementation of $\mathcal{F}$ *if, for all* $n \in I\!N$, *all games* $G \in \mathcal{G}$ *with input length* n, *and all* $Z \in \mathcal{Z}$, *if* $\vec{\perp}_Z$ *is a* $p(n, \cdot)$*-robust best response to* $\Lambda^{\mathcal{F}}_{-Z}$ *in then* $\vec{\perp}_Z$ *is an* ϵ*-best response to* $\vec{M}_{-Z}$ *in* $(G, \mathcal{F}')$.

4 Relating Cryptographic and Game-Theoretic Implementation

We briefly recall the notion of *precise secure computation* [37, 38], which is a strengthening of the traditional notion of secure computation [12]; more details are given in Appendix B. An *m-ary functionality* f is specified by a random process that maps vectors of inputs to vectors of outputs (one input and one output for each player). That is, $f : ((\{0,1\}^*)^m \times \{0,1\}^\infty) \to (\{0,1\}^*)^m$, where we view f_i as the ith component of the output vector; that is, $f = (f_1, \ldots, f_m)$. We often abuse notation and suppress the random bitstring r, writing $f(\vec{x})$ or $f_i(\vec{x})$. (We can think of $f(\vec{x})$ and $f_i(\vec{x})$ as random variables.) A mediator $\mathcal{F}$ (resp., a machine profile $\vec{M}$) *computes* f if, for all $n \in N$, all inputs $\vec{x} \in (\{0,1\}^n)^m$, if the players tell the mediator their inputs and output what the mediator $\mathcal{F}$ tells them (resp., the output vector of the players after an execution of $\vec{M}$ where M_i gets input x_i) is identically distributed to $f^n(\vec{x})$. Roughly speaking, a protocol $\vec{M}$ for computing a function f is *secure* if, for every adversary A participating in the *real execution* of $\vec{M}$, there exists a "simulator" $\tilde{A}$ participating in an *ideal execution* where all players directly talk to a trusted third party (i.e., a mediator) computing f; the job of $\tilde{A}$ is to provide appropriate inputs to the trusted party, and to reconstruct the view of A in the real execution such that no *distinguisher* D can distinguish the outputs of the parties and the view of the adversary A in the real and the ideal execution. (Note that in the real execution the view of the adversary is simply the actual view of A in the execution, whereas in the ideal execution it is the view output by the simulator $\tilde{A}$). The traditional notion of secure computation [12] requires only that the *worst-case* complexity (size and running-time) of $\tilde{A}$ is polynomially related to that of A. Precise secure computation [37, 38] additionally requires that the running time of the simulator $\tilde{A}$ "respects" the running time of the adversary A in an "execution-by-execution" fashion: a secure computation is said to have precision $p(n, t)$ if the running-time of the simulator $\tilde{A}$ (on input security parameter n) is bounded by $p(n, t)$ whenever $\tilde{A}$ outputs a view in which the running-time of A is t.

For our results, we slightly weaken the notion of precise secure (roughly speaking, by changing the order the quantifiers in analogy with the work of of Dwork, Naor, Reingold, and Stockmeyer [39]). We also generalize the notion by allowing arbitrary complexity measures $\vec{\mathscr{C}}$ (instead of just running-time) and *general adversary structures* [40] (where the specification of a secure computation includes a set $\mathcal{Z}$ of subsets of players such that the adversary is allowed to corrupt only the players in one of the subsets in $\mathcal{Z}$; in contrast, in [12, 37] only *threshold adversaries* are considered, where $\mathcal{Z}$ consists of all subsets up to a pre-specified size k). The formal definition of *weak* $\vec{\mathscr{C}}$*-precise secure computation* is given in Appendix B.1. Note that the we can always regain the "non-precise" notion of secure computation by instantiating $\mathscr{C}_Z(M, v)$ with the sum of the *worst-case* running-time of M (on inputs of the same length as the input length in v) and size of M. Thus, by the results of [12, 13, 41], it follows that there exists weak $\vec{\mathscr{C}}$-precise secure computation protocols with precision $p(n, t) = poly(n, t)$ when $\mathscr{C}_Z(M, v)$ is the sum of the worst-case running-time of M and size of M. The results of [37, 38] extend to show the existence of weak $\vec{\mathscr{C}}$-precise secure computation protocols with precision $p(n, t) = O(t)$ when $\mathscr{C}_Z(M, v)$ is the sum of the running time (as opposed to just worst-case running-time) of $M(v)$ and size of M. The results above continue to hold if we consider "coarse" measures of running-time and size; for instance, if, say, n^2

computational steps correspond to one unit of complexity (in canonical machine games with input length n). See Appendix 4.2 for more details.

4.1 Equivalences

As a warm-up, we show that "error-free" secure computation, also known as *perfectly-secure computation* [41], already implies the traditional game-theoretic notion of implementation [22] (which does not consider computation). To do this, we first formalize the traditional game-theoretic notion using our notation: Let $\vec{M}$ be an m-player profile of machines. We say that $(\vec{M}, \mathcal{F}')$ is a *traditional game-theoretic implementation* of $\mathcal{F}$ if $(\vec{M}, \mathcal{F}')$ is a $(\mathcal{G}^{\mathsf{nocomp}}, \{\{1\}, \dots \{m\}\}, 0)$-universal implementation of $\mathcal{F}$ with 0-error, where $\mathcal{G}^{\mathsf{nocomp}}$ denotes the class of all m-player canonical machine games where the utility functions do not depend on the complexity profile. (Recall that the traditional notion does not consider computational games or coalition games.)

Proposition 4.1 *If f is an m-ary functionality, $\mathcal{F}$ is a mediator that computes f, and $\vec{M}$ is a perfectly-secure computation of $\mathcal{F}$, then $(\vec{M}, \mathsf{comm})$ is a game-theoretic implementation of $\mathcal{F}$.*

Proof: We start by showing that running $\vec{M}$ is a Nash equilibrium if running $\vec{\Lambda}^{\mathcal{F}}$ with mediator $\mathcal{F}$ is one. Recall that the cryptographic notion of error-free secure computation requires that for every player i and every "adversarial" machine M_i' controlling player i, there exists a "simulator" machine $\tilde{M}_i$, such that the outputs of all players in the execution of $(M_i', \vec{M}_{-i})$ are *identically distributed* to the outputs of the players in the execution of $(\tilde{M}_i, \vec{\Lambda}^{\mathcal{F}}_{-i})$ with mediator $\mathcal{F}$.[5] In game-theoretic terms, this means that every "deviating" strategy M_i' in the communication game can be mapped into a deviating strategy $\tilde{M}_i$ in the mediated game with the same output distribution for each type, and, hence, the same utility, since the utility depends only on the type and the output distribution; this follows since we require universality only with respect to games in $\mathcal{G}^{\mathsf{nocomp}}$. Since no deviations in the mediated game can give higher utility than the Nash equilibrium strategy of using $\Lambda_i^{\mathcal{F}}$, running $\vec{M}$ must also be a Nash equilibrium.

It only remains to show that $\vec{M}$ and $\vec{\Lambda}^{\mathcal{F}}$ induce the same action distribution; this follows directly from the definition of secure computation by considering an adversary that does not corrupt any parties. ∎

We note that the converse implication does not hold. Since the traditional game-theoretic notion of implementation does not consider computational cost, it does not take into account computational advantages possibly gained by using $\vec{M}$, issues that are critical in the cryptographic notion of zero-knowledge simulation. We now show that weak precise secure computation is equivalent to strong $\mathcal{G}$-universal implementation for certain natural classes $\mathcal{G}$ of games. For this result, we assume that the only machines that can have a complexity of 0 are those that "do nothing": we require that, for all complexity functions $\mathscr{C}$, $\mathscr{C}(M, v) = 0$ for some view v iff $M = \perp$ iff $\mathscr{C}(M, v) = 0$ for all views v. (Recall that $\perp$ is a canonical representation of the TM that does nothing: it does not read its input, has no state changes, and writes nothing.) If $G = ([m], \mathcal{M}, \Pr, \vec{\mathscr{C}}, \vec{u})$ is a canonical game with input length n, then

1. G is *machine universal* if the machine set $\mathcal{M}$ is the set of terminating Turing machines;
2. G is *normalized* if the range of u_Z is $[0, 1]$ for all subsets Z of $[m]$;
3. G is *monotone* (i.e., "players never prefer to compute more") if, for all subset Z of $[m]$, all type profiles $\vec{t}$, action profiles $\vec{a}$, and all complexity profiles $(c_Z, \vec{c}_{-Z})$, $(c_Z', \vec{c}_{-Z})$, if $c_Z' > c_Z$, then $u_Z(\vec{t}, \vec{a}, (c_Z', \vec{c}_{-Z})) \leq u_i(\vec{t}, \vec{a}, (c_Z, \vec{c}_{-Z}))$;
4. G is a $\vec{\mathscr{C}}'$-game if $\mathscr{C}_Z = \mathscr{C}_Z'$ for all subsets Z of $[m]$.

Let $\mathcal{G}^{\vec{\mathscr{C}}}$ denote the class of machine-universal, normalized, monotone, canonical $\vec{\mathscr{C}}$-games. For our theorem we need some minimal constraints on the complexity function. For the forward direction of our equivalence results (showing that precise secure computation implies universal implementation), we require that honestly running the protocol should have constant complexity, and that it be the same with and without a mediator. More precisely, we assume that the complexity profile $\vec{\mathscr{C}}$ is $\vec{M}$-*acceptable*, that is, for every subset Z, the machines $(\Lambda^{\mathcal{F}})^b_Z$ and M^b_Z have the same complexity c_0 for all inputs; that is, $\mathscr{C}_Z((\Lambda^{\mathcal{F}})^b_Z, \cdot) = c_0$ and $\mathscr{C}_Z(M^b_Z, \cdot) = c_0$.[6] As mentioned in Section 3, an assumption of this nature is necessary to satisfy universality with respect to general classes of games; it is easily satisfied if we consider "coarse" complexity functions. For the backward direction of our equivalence (showing that universal implementation implies precise secure computation), we require that certain operations, like moving output from one tape to another, do not incur any additional complexity. Such complexity functions are called *output-invariant*; we provide a formal definition in Appendix C.

We can now state the connection between secure computation and game-theoretic implementation. In

[5]The follows from the fact that perfectly-secure computation is error-free.

[6]Our results continue to hold if c_0 is a function of the input length n, but otherwise does not depend on the view.

the forward direction, we restrict attention to protocols $\vec{M}$ computing some m-ary functionality f that satisfy the following natural property: if a subset of the players "aborts" (not sending any messages, and outputting nothing), their input is intepreted as λ.[7] More precisely, $\vec{M}$ is an *abort-preserving* computation of f if for all $n \in N$, every subset Z of $[m]$, all inputs $\vec{x} \in (\{0,1\}^n)^m$, the output vector of the players after an execution of $(\vec{\perp}_Z, \vec{M}_{-Z})$ on input $\vec{x}$ is identically distributed to $f(\lambda_Z, \vec{x}_{-Z})$.

Theorem 4.2 (Equivalence: Information-theoretic case) *Suppose that f is an m-ary functionality, $\mathcal{F}$ is a mediator that computes f, $\vec{M}$ is a machine profile that computes f, $\mathcal{Z}$ is a set of subsets of $[m]$, $\vec{\mathscr{C}}$ is a complexity function, and p a precision function.*

- *If $\vec{\mathscr{C}}$ is $\vec{M}$-acceptable and $\vec{M}$ is an abort-preserving weak $\mathcal{Z}$-secure computation of f with $\vec{\mathscr{C}}$-precision p and ϵ-statistical error, then $(\vec{M}, \mathsf{comm})$ is a strong $(\mathcal{G}^{\vec{\mathscr{C}}}, \mathcal{Z}, p)$-universal implementation of $\mathcal{F}$ with error ϵ.*
- *If $\vec{\mathscr{C}}$ is $\vec{M}$-acceptable and output-invariant, and $(\vec{M}, \mathsf{comm})$ is a strong $(\mathcal{G}^{\vec{\mathscr{C}}}, \mathcal{Z}, p)$-universal implementation of $\mathcal{F}$ with error ϵ', then for every $\epsilon < \epsilon'$, $\vec{M}$ is a weak $\mathcal{Z}$-secure computation of f with $\vec{\mathscr{C}}$-precision p and ϵ-statistical error.*

As a corollary of Theorem 4.2, we get that known (precise) secure computation protocols directly yield appropriate universal implementations, provided that we consider complexity functions that are $\vec{M}$-acceptable. For instance, by the results of [38, 41], every efficient m-ary functionality f has a weak $\mathcal{Z}$-secure computation protocol $\vec{M}$ with $\mathscr{C}$-precision $p(n,t) = t$ if $\mathscr{C}_Z(M, v)$ is the sum of the running time of $M(v)$ and size of M, and $\mathcal{Z}$ consists of all subsets of $[m]$ of size smaller than $|m|/3$. This result still holds if we consider "coarse" measures of running-time and size where, say, $O(n^c)$ computational steps (and size) correspond to one unit of complexity (in canonical machine games with input length n). Furthermore, protocol $\vec{M}$ is abort-preserving, has a constant description, and has running-time smaller than some fixed polynomial $O(n^c)$ (on inputs of length n). So, if we consider an appropriately coarse notion of run-time and description size, $\mathscr{C}_Z$ is $\vec{M}$-acceptable. By Theorem 4.2, it then immediately follows that that every efficient m-ary functionality f has a strong $(\mathcal{G}^{\vec{\mathscr{C}}}, \mathcal{Z}, O(1))$-universal implementation with error 0.

Theorem 4.2 also shows that a universal implementation of a mediator $\mathcal{F}$ computing a function f with respect to *general* classes of games is "essentially" as hard to achieve as a secure computations of f. In particular, as long as the complexity function is output-invariant, such a universal implementation is a weak precise secure computation. Although the output-invariant condition might seem somewhat artificial, Theorem 4.2 illustrates that overcoming the "secure-computation barrier" with respect to general classes of games requires making strong (and arguably unnatural[8]) assumptions about the complexity function. We have not pursued this path. In Section 5, we instead consider universal implementation with respect to restricted class of games. As we shall see, this provides an avenue for circumventing traditional impossibility results with respect to secure computation.

In Section 4.2 we also provide a "computational" analogue of Theorem 4.2, as well as a characterization of the "standard" (i.e., "non-precise") notion of secure computation. We provide a proof overview of Theorem 4.2 here; we defer the complete proof to the full version.

Proof overview Needless to say, this oversimplified sketch leaves out many crucial details that complicate the proof.

Weak precise secure computation implies strong universal implementation. At first glance, it might seem like the traditional notion of secure computation of [12] easily implies the notion of universal implementation: if there exists some (deviating) strategy A in the communication game implementing mediator $\mathcal{F}$ that results in a different distribution over actions than in equilibrium, then the simulator $\tilde{A}$ for A could be used to obtain the same distribution; moreover, the running time of the simulator is within a polynomial of that of A. Thus, it would seem like secure computation implies that any "poly"-robust equilibrium can be implemented. However, the utility function in the game considers the complexity of each execution of the computation. So, even if the worst-case running time of $\tilde{A}$ is polynomially related to that of A, the utility of corresponding executions might be quite different. This difference may have a significant effect on the equilibrium. To make the argument go through we need a simulation that preserves complexity in an execution-by-execution manner. This is exactly what precise zero knowledge [37] does. Thus, intuitively, the degradation in computational robustness by a universal implementation corresponds to the precision of a secure computation.

More precisely, to show that a machine profile $\vec{M}$ is a universal implementation, we need to show that

[7]All natural secure computation protocols that we are aware of (e.g., [12, 41]) satisfy this property.

[8]With a coarse complexity measure, it seems natural to assume that moving content from one output tape to another incurrs no change in complexity.

whenever Λ is a p-robust equilibrium in a game G with mediator $\mathcal{F}$, then $\vec{M}$ is an ϵ-equilibrium (with the communication mediator comm). Our proof proceeds by contradiction: we show that a deviating strategy M'_Z (for a coalition Z) for the ϵ-equilibrium $\vec{M}$ can be turned into a deviating strategy $\tilde{M}_Z$ for the p-robust equilibrium $\vec{\Lambda}$. We here use the fact that $\vec{M}$ is a weak precise secure computation to find the machine $\tilde{M}_Z$; intuitively $\tilde{M}_Z$ will be the simulator for M'_Z. The key step in the proof is a method for embedding any coalition machine game G into a distinguisher D that "emulates" the role of the utility function in G. If done appropriately, this ensures that the utility of the (simulator) strategy $\tilde{M}_Z$ is close to the utility of the strategy M'_Z, which contradicts the assumption that $\vec{\Lambda}$ is an ϵ-Nash equilibrium.

The main obstacle in embedding the utility function of G into a distinguisher D is that the utility of a machine $\tilde{M}_Z$ in G depends not only on the types and actions of the players, but also on the complexity of running $\tilde{M}_Z$. In contrast, the distinguisher D does not get the complexity of $\tilde{M}$ as input (although it gets its output v). On a high level (and oversimplifying), to get around this problem, we let D compute the utility *assuming* (incorrectly) that $\tilde{M}_Z$ has complexity $c = \mathscr{C}(M', v)$ (i.e., the complexity of M'_Z in the view v output by $\tilde{M}_Z$). Suppose, for simplicity, that $\tilde{M}_Z$ is *always* "precise" (i.e., it always respects the complexity bounds).[9] Then it follows that (since the complexity c is always close to the actual complexity of $\tilde{M}_Z$ in every execution) the utility computed by D corresponds to the utility of some game $\tilde{G}$ that is at most a p-speed up of G. (To ensure that $\tilde{G}$ is indeed a speedup and not a "slow-down", we need to take special care with simulators that potentially run faster than the adversary they are simulating. The monotonicity of G helps us to circumvent this problem.) Thus, although we are not able to embed G into the distinguisher D, we can embed a related game $\tilde{G}$ into D. This suffices to show that $\vec{\Lambda}$ is not a Nash equilibrium in $\tilde{G}$, contradicting the assumption that $\vec{\Lambda}$ is a p-robust Nash equilibrium. A similar argument can be used to show that $\perp$ is also an ϵ-best response to $\vec{M}_{-Z}$ if $\perp$ is a p-robust best response to $\vec{\Lambda}_{-Z}$, demonstrating that $\vec{M}$ in fact is a strong universal implementation. We here rely on the fact $\vec{M}$ is abort-preserving to ensure that aborting in $(G, \mathcal{F})$ has the same effect as in (G, comm).

Strong universal implementation implies weak precise secure computation. To show that strong universal implementation implies weak precise secure computation, we again proceed by contradiction. We show how the existence of a distinguisher D and an adversary M'_Z that cannot be simulated by *any* machine $\tilde{M}_Z$ can be used to construct a game G for which $\vec{M}$ is not a strong implementation. The idea is to have a utility function that assigns high utility to some "simple" strategy M^*_Z. In the mediated game with $\mathcal{F}$, no strategy can get better utility than M^*_Z. On the other hand, in the cheap-talk game, the strategy M'_Z does get higher utility than M^*_Z. As D indeed is a function that "distinguishes" a mediated execution from a cheap-talk game, our approach will be to try to embed the distinguisher D into the game G. The choice of G depends on whether $M'_Z = \perp$. We now briefly describe these games.

If $M'_Z = \perp$, then there is no simulator for the machine $\perp$ that simply halts. In this case, we construct a game G where using $\perp$ results in a utility that is determined by running the distinguisher. (Note that $\perp$ can be easily identified, since it is the only strategy that has complexity 0.) All other strategies instead get some canonical utility d, which is higher than the utility of $\perp$ in the mediated game. However, since $\perp$ cannot be "simulated", playing $\perp$ in the cheap-talk game leads to an even higher utility, contradicting the assumption that $\vec{M}$ is a universal implementation.

If $M'_Z \neq \perp$, we construct a game G' in which each strategy other than $\perp$ gets a utility that is determined by running the distinguisher. Intuitively, efficient strategies (i.e., strategies that have relatively low complexity compared to M'_Z) that output views on which the distinguisher outputs 1 with high probability get high utility. On the other hand, $\perp$ gets a utility d that is at least as good as what the other strategies can get in the mediated game with $\mathcal{F}$. This makes $\perp$ a best response in the mediated game; in fact, we can define the game G' so that it is actually a p-robust best response. However, it is not even an ϵ-best-response in the cheap-talk game: M'_Z gets higher utility, as it receives a view that cannot be simulated. (The output-invariant condition on the complexity function $\mathscr{C}$ is used to argue that M'_Z can output its view at no cost.) ∎

4.2 A Computational Equivalence Theorem

To prove a "computational" analogue of our equivalence theorem (relating computational precise secure computation and universal implementation), we need to introduce some further restrictions on the complexity functions, and the classes of games considered.

- A (vector of) complexity functions $\vec{\mathscr{C}}$ is *efficient* if each function is computable by a (randomized) polynomial-sized circuit.

[9]This is an unjustified assumption; in the actual proof we actually need to consider a more complicated construction.

- A secure computation game $G = ([m], \mathcal{M}^{T(n)}, \Pr, \vec{\mathscr{C}}, \vec{u})$ with input length n is said to be $T(\cdot)$-*machine universal* if
  - the machine set $\mathcal{M}^{T(n)}$ is the set of Turing machines implementable by $T(n)$-sized randomized circuits, and
  - $\vec{u}$ is computable by a $T(n)$-sized circuit.

Let $\mathcal{G}^{\vec{\mathscr{C}},T}$ denote the class of $T(\cdot)$-machine universal, normalized, monotone, canonical $\vec{\mathscr{C}}$-games. Let $\mathcal{G}^{\vec{\mathscr{C}},\text{poly}}$ denote the union of $\mathcal{G}^{\vec{\mathscr{C}},T}$ for all polynomial functions T.

Theorem 4.3 (Equivalence: Computational Case) *Suppose that f is an m-ary functionality, $\mathcal{F}$ is a mediator that computes f, $\vec{M}$ is a machine profile that computes f, $\mathcal{Z}$ is a set of subsets of $[m]$, $\vec{\mathscr{C}}$ is an efficient complexity function, and p a precision function.*

- *If $\vec{\mathscr{C}}$ is $\vec{M}$-acceptable and $\vec{M}$ is an abort-preserving weak $\mathcal{Z}$-secure computation of f with computational $\vec{\mathscr{C}}$-precision p, then for every polynomial T, there exists some negligible function ϵ such that $(\vec{M}, \mathsf{comm})$ is a strong $(\mathcal{G}^{\vec{\mathscr{C}},T}, \mathcal{Z}, p)$-universal implementation of $\mathcal{F}$ with error ϵ.*
- *If $\vec{\mathscr{C}}$ is $\vec{M}$-acceptable and output-invariant, and for every polynomial T, there exists some negligible function ϵ, such that $(\vec{M}, \mathsf{comm})$ is a strong $(\mathcal{G}^{\vec{\mathscr{C}}}, \mathcal{Z}, p)$-universal implementation of $\mathcal{F}$ with error ϵ, then $\vec{M}$ is a weak $\mathcal{Z}$-secure computation of f with computational $\vec{\mathscr{C}}$-precision p.*

A proof of Theorem 4.3 appears in the full version. Note that Theorem 4.3 also provides a game-theoretic characterization of the "standard" (i.e., "non-precise") notion of secure computation. We simply consider a "coarse" version of the complexity function $\mathsf{wc}(v)$ that is the sum of the size of M and the worst-case running-time of M on inputs of the same length as in the view v. (We need a coarse complexity function to ensure that $\mathscr{C}$ is $\vec{M}$-acceptable and output-invariant.) With this complexity function, the definition of weak precise secure computation reduces to the traditional notion of weak secure computation without precision (or, more precisely, with "worst-case" precision just as in the traditional definition). Given this complexity function, the precision of a secure computation protocol becomes the traditional "overhead" of the simulator (this is also called *knowledge tightness* [42]). Roughly speaking, "weak secure computation" with overhead p is thus equivalent to strong $(\mathcal{G}^{\vec{\mathsf{wc}},\text{poly}}, p)$-universal implementation with negligible error.

5 Directions for Future Research

We have defined a general approach to taking computation into account in game theory that subsumes previous approaches, and shown a close connection between computationally robust Nash equilibria and precise secure computation. This opens the door to a number of exciting research directions in both secure computation and game theory. We describe a few here:

- Our equivalence result for secure computation might seem like a negative result. It demonstrates that considering only rational players (as opposed to adversarial players) does not facilitate protocol design. Note, however, that for the equivalence to hold, we must consider implementations universal with respect to essentially *all* games. In many settings, it might be reasonable to consider implementations universal with respect to only certain subclasses of games; in such scenarios, universal implementations may be significantly simpler or more efficient, and may also circumvent traditional lower bounds. We list some natural restrictions on classes of games below, and discuss how such restrictions can be leveraged in protocol design. These examples illustrate some of the benefits of a fully game-theoretic notion of security that does not rely on the standard cryptographic simulation paradigm, and shows how our framework can capture in a natural way a number of natural notions of security.

  To relate our notions to the standard definition of secure computation, we here focus on classes of games $\mathcal{G}$ that are subsets of $\mathcal{G}^{\vec{\mathsf{wc}},\text{poly}}$ (as defined in Section 4.2). Furthermore, we consider only 2-player games and restrict attention to games G where the utility function is *separable* in the following sense: there is a standard game G' (where computational costs are not taken into account) and a function u_i^C on complexity profiles for each player i, such that, for each player i, $u_i(\vec{t}, \vec{a}, \vec{c}) = u_i^{G'}(\vec{t}, \vec{a}) + u_i^C(\vec{c})$. We refer to G' as the standard game *embedded* in G. Intuitively, this says that the utilities in G are just the sum of the utility in a game G' where computation is not taken into account and a term that depends only on computation costs.

  Games with punishment: Many natural situations can be described as games where players can choose actions that "punish" an individual player i. For instance, this punishment can represent the cost of being excluded from future interactions. Intuitively, games with punishment model situations where players do not want to be caught cheating. Punishment strategies (such as the *grim-trigger* strategy in repeated prisoner's dilemma, where a player defects forever once his opponent defects once [21]) are extensively used in the game-theory literature. We give two examples where cryptographic protocol design is facilitated when requiring only implementations that are universal with respect to games with punishment.

Covert adversaries: As observed by Malkhi et al. [43], and more recently formalized by Aumann and Lindell [44], in situations were players do not want to be caught cheating, it is easier to construct efficient protocols. Using our framework, we can formalize this intuition in a straightforward way. To explain the intuitions, we consider a particularly simple setting. Let $\mathcal{G}^{\mathsf{punish}}$ consist of normalized 2-player games G (with a standard game G' embedded in G), where (1) honestly reporting your input and outputting whatever the mediator replies (i.e., playing the strategy implemented by $\vec{\Lambda}$) is a Nash equilibrium in $(G', \mathcal{F})$ where both players are guaranteed to get utility 1/2 (not just in expectation, but even in the worst-case), and (2) there is a special string punish such that player $1-i$ receives payoff 0 in G if player i outputs the string punish. In this setting, we claim that any secure computation of a function f with respect to covert adversaries with deterrent 1/2 in the sense of [44, Definition 3.3] is a $(\mathcal{G}^{\mathsf{punish}}, poly, \mathcal{Z})$- universal implementation of $\mathcal{F}$ with negligible error (where $\mathcal{Z} = \{\{1\},\{2\}\}$, and $\mathcal{F}$ is a mediator computing f). Intuitively, this follows since a "cheating" adversary gets caught with probability 1/2 and thus punished; the expected utility of cheating is thus at most $1/2 \times 1 + 1/2 \times 0$, which is no greater than the expected utility of playing honestly.

Fairness: It is well-known that, for many functions, secure 2-player computation where both players receive output is impossible if we require *fairness* (i.e., that either both or neither of the players receives an output) [45]. Such impossibility results can be easily circumvented by considering universal implementation with respect to games with punishment. This follows from the fact that although it is impossible to get secure computation with fairness, the weaker notion of *secure computation with abort* [46] is achievable. Intuitively, this notion guarantees that the only attack possible is one where one of the players prevents the other player from getting its output; this is called an *abort*. This is formalized by adapting the trusted-party in the ideal model to allow the adversary to send a special abort message to the trusted party after seeing its own output, which blocks it from delivering an output to the honest party. To get a universal implementation with respect to games with punishment, it is sufficient to use any secure computation protocol with abort (see [38, 46]) modified so that players output punish if the other player aborts. It immediately follows that a player can never get a higher utility by aborting (as this will be detected by the other player, and consequently the aborting player will be punished). This result can be viewed as a generalization of the approach of [5].[10]

Strictly monotone games: In our equivalence results we considered monotone games, where players never prefer to compute more. It is sometimes reasonable to assume that players *strictly* prefer to compute less. We outline a few possible advantages of considering universal implementations with respect to strictly monotone games.

Gradual-release protocols: One vein of research on secure computation considers protocols for achieving fair exchanges using *gradual-release* protocols (see e.g., [47]). In a gradual-release protocol, the players are guaranteed that if at any point one player aborts, then the other player(s) can compute the output within a comparable amount of time (e.g., we can require that if a player aborts and can compute the answer in t time units, then all the other players should be able to compute it within $2t$ time units). We believe that by making appropriate assumptions about the utility of computation, we can ensure that players never have incentives to deviate. Consider, for instance, a two-player computation of a function f where in the last phase the players invoke a gradual exchange protocol such that if any player aborts during the gradual exchange protocol, the other players attempts to recover the secret using a brute-force search. Intuitively, if for each player the cost of computing t extra steps is positive, even if the other player computes, say, $2t$ extra steps, it will never be

[10]For this application, it is not necessary to use our game-theoretic definition of security. An alternative way to capture fairness in this setting would be to require security with respect to the standard (simulation-based) definition with abort, and additionally fairness (but not security) with respect to rational agents, according to the definition of [5, 25]; this approach is similar to the one used by Kol and Naor [27]. Our formalization is arguably more natural, and also considers rational agents that "care" about computation.

worth it for a player to abort: by the security of the gradual-release protocol, an aborting player can only get its output twice as fast as the other player. We note that merely making assumptions about the cost of computing will not be sufficient to make this approach work; we also need to ensure that players prefer to get the output than not getting it, even if they can trick other players into computing for a long time. Otherwise, a player might prefer to abort and not compute anything, while the other player attempts to compute the output. We leave a full exploration of this approach for future work.

Error-free implementations: Unlike perfectly-secure protocols, computationally-secure protocols protocols inherently have a nonzero error probability. For instance, secure 2-player computation can be achieved only with computational security (with nonzero error probability). By our equivalence result, it follows that strong universal implementations with respect to the most general classes of 2-player games also require nonzero error probability. Considering universality with respect to only strictly monotone games gives an approach for achieving error-free implementations. This seems particularly promising if we consider an idealized model where cryptographic functionalities (such as one-way functions) are modeled as black boxes (see, e.g., the random oracle model of Bellare and Rogaway [48]), and the complexity function considers the number of calls to the cryptographic function. Intuitively, if the computational cost of trying to break the cryptographic function is higher than the expected gain, it is not worth deviating from the protocol. We leave open an exploration of this topic. (A recent paper by Micali and Shelat [49] relies on this idea, in combination with *physical* devices, to acheive error-free implementations in the context of secret sharing.)

Using computation as payment. Shoham and Tennenholtz [28] have investigated what functions f of two players' inputs x_1, x_2 can be computed by the players if they have access to a trusted party. The players are assumed to want to get the output $y = f(x_1, x_2)$, but each player i does not want to reveal more about his input x_i than what can be deduced from y. Furthermore, each player i prefers that other players do not get the output (although this is not as important as i getting the output and not revealing its input x_i). Interestingly, as Shoham and Tennenholtz point out, the simple binary function AND cannot be truthfully computed by two players, even if they have access to a trusted party. A player that has input 0 always knows the output y and thus does not gain anything from providing its true input to the trusted party: in fact, it always prefers to provide the input 1 in order to trick the other player.

We believe that for strictly monotone games this problem can be overcome by the use of cryptographic protocols. The idea is to construct a cryptographic protocol for computing AND where the players are required to solve a computational puzzle if they want to use 1 as input; if they use input 0 they are not required to solve the puzzle. The puzzle should have the property that it requires a reasonable amount of computational effort to solve. If this computational effort is more costly than the potential gain of tricking the other player to get the wrong output, then it is not worth it for a player to provide input 1 unless its input actually is 1. To make this work, we need to make sure the puzzle is "easy" enough to solve, so that a player with input 1 will actually want to solving the puzzle in order to get the correct output. We leave a full exploration of this idea for future work.

More generally, we believe that combining computational assumptions with assumptions about utility will be a fruitful line of research for secure computation. For instance, it is conceivable that difficulties associated with concurrent executability of protocols could be alleviated by making assumptions regarding the cost of message scheduling; the direction of Cohen, Kilian, and Petrank [50] (where players who delay messages are themselves punished with delays) seems relevant in this regard.

- As we have seen, universal implementation is equivalent to a variant of precise secure computation with the order of quantification reversed. It would be interesting to find a notion of implementation that corresponds more closely to the standard definition, without a change in the order of quantifier; in particular, whereas the traditional definition of zero-knowledge guarantees *deniability* (i.e., the property that the interaction does not leave any "trace"), the new one does not. Finding a game-theoretic definition that also captures deniability seems like an interesting question.
- We showed that Nash equilibria do not always exist in games with computation. This leaves open the question of what the appropriate solution concept is.
- Our notion of universal implementation uses

Nash equilibrium as solution concept. It is well known that in (traditional) extensive form games (i.e., games defined by a game tree), a Nash equilibrium might prescribe non-optimal moves at game histories that do no occur on the equilibrium path. This can lead to "empty threats": "punishment" strategies that are non-optimal and thus not credible. Many recent works on implementation (see e.g., [26, 35]) therefore focus on stronger solution concepts such as *sequential equilibrium* [29]. We note that when taking computation into account, the distinction between credible and non-credible threats becomes more subtle: the threat of using a non-optimal strategy in a given history might be credible if, for instance, the overall complexity of the strategy is smaller than any strategy that is optimal at every history. Thus, a simple strategy that is non-optimal off the equilibrium path might be preferred to a more complicated (and thus more costly) strategy that performs better off the equilibrium path (indeed, people often use non-optimal but simple "rules-of-thumbs" when making decisions). Finding a good definition of empty threats in games with computation costs seems challenging.

One approach to dealing with empty threats to to consider the notion of sequential equilibrium in machine games. Roughly speaking, in a sequential equilibrium, every player must make a best response at every information set, where an information set is a set of nodes in the game tree that the agent cannot distinguish. The standard assumption in game theory is that the information set is given *exogenously*, as part of the description of the game tree. As Halpern [51] has argued, an exogenously-given information set does not always represent the information that an agent actually has. The issue becomes even more significant in our framework. While an agent may have information that allows him to distinguish two nodes, the computation required to realize that they are different may be prohibitive, and (with computational costs) an agent can rationally decide not to do the computation. This suggests that the information sets should be determined by the machine. More generally, in defining solution concepts, it has proved necessary to reason about players' beliefs about other players' beliefs; now these beliefs will have to include beliefs regarding computation. Finding a good model of such beliefs seems challenging. See the full paper for further discussion of sequential equilibrium.

- A natural next step would be to introduce notions of computation in the epistemic logic. There has already been some work in this direction (see, for example, [52–54]). We believe that combining the ideas of this paper with those of the earlier papers will allow us to get, for example, a cleaner knowledge-theoretic account of zero knowledge than that given by Halpern, Moses, and Tuttle [52]. A first step in this direction is taken in [55].
- Finally, it would be interesting to use behavioral experiments to, for example, determine the "cost of computation" in various games (such as the finitely repeated prisoner's dilemma).

Acknowledgments

The second author wishes to thank Silvio Micali and abhi shelat for many exciting discussions about cryptography and game theory, and Ehud and Adam Kalai for enlightening discussions.

References

[1] H. A. Simon. A behavioral model of rational choice. *Quarterly Journal of Economics*, 49:99–118, 1955.

[2] A. Neyman. Bounded complexity justifies cooperation in finitely repated prisoner's dilemma. *Economic Letters*, 19:227–229, 1985.

[3] C. H. Papadimitriou and M. Yannakakis. On complexity as bounded rationality. In *Proc. 26th ACM Symposium on Theory of Computing*, pages 726–733, 1994.

[4] N. Megiddo and A. Wigderson. On play by means of computing machines. In *Theoretical Aspects of Reasoning about Knowledge: Proc. 1986 Conference*, pages 259–274. 1986.

[5] Y. Dodis, S. Halevi, and T. Rabin. A cryptographic solution to a game theoretic problem. In *CRYPTO 2000: 20th International Cryptology Conference*, pages 112–130. Springer-Verlag, 2000.

[6] A. Urbano and J. E. Vila. Computationally restricted unmediated talk under incomplete information. *Economic Theory*, 23(2):283–320, 2004.

[7] A. Rubinstein. Finite automata play the repeated prisoner's dilemma. *Journal of Economic Theory*, 39: 83–96, 1986.

[8] E. Kalai. Bounded rationality and strategic complexity in repeated games. In *Game Theory and Applications*, pages 131–157. Academic Press, San Diego, 1990.

[9] E. Ben-Sasson, A. Kalai, and E. Kalai. An approach to bounded rationality. In *Advances in Neural Information Processing Systems 19 (Proc. of NIPS 2006)*, pages 145–152. 2007.

[10] R. Myerson. Incentive-compatibility and the bargaining problem. *Econometrica*, 47:61–73, 1979.

[11] F. Forges. An approach to communication equilibria. *Econometrica*, 54(6):1375–85, 1986.

[12] O. Goldreich, S. Micali, and A. Wigderson. How to

play any mental game. In *Proc. 19th ACM Symposium on Theory of Computing*, pages 218–229, 1987.

[13] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[14] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proc. 18th ACM Symposium on Theory of Computing*, pages 364–369, 1986.

[15] H. W. Kuhn. Extensive games and the problem of information. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games II*, pages 193–216. Princeton University Press, Princeton, N.J., 1953.

[16] G. Walker and D. Walker. *The Official Rock Paper Scissors Strategy Guide.* Simon & Schuster, Inc., New York, 2004.

[17] K. Binmore. Modeling rational players I. *Economics and Philosophy*, 3:179–214, 1987.

[18] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* Univ. of California Press, 2nd edition, 1951.

[19] R. J. Lipton and E. Markakis. Nash equilibria via polynomial equations. In *Proc. LATIN 2004: Theoretical Informatics*, pages 413–422, 2004.

[20] C. H. Papadimitriou and T. Roughgarden. Computing equilibria in multi-player games. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms*, pages 82–91, 2005.

[21] R. Axelrod. *The Evolution of Cooperation.* Basic Books, New York, 1984.

[22] F. Forges. Universal mechanisms. *Econometrica*, 58 (6):1341–64, 1990.

[23] I. Abraham, D. Dolev, R. Gonen, and J. Y. Halpern. Distributed computing meets game theory: Robust mechanisms for rational secret sharing and multiparty computation. In *Proc. 25th ACM Symposium on Principles of Distributed Computing*, pages 53–62, 2006.

[24] D. Gordon and J. Katz. Rational secret sharing, revisited. In *SCN (Security in Communication Networks) 2006*, pages 229–241, 2006.

[25] J. Y. Halpern and V. Teadgue. Rational secret sharing and multiparty computation: extended abstract. In *Proc. 36th ACM Symposium on Theory of Computing*, pages 623–632, 2004.

[26] S. Izmalkov, M. Lepinski, and S. Micali. Perfect implementation of normal-form mechanisms. Available at http://people.csail.mit.edu/silvio/. This is an revised version of "Rational secure computation and ideal mechanism design", *Proc. 46th IEEE Symposium on Foundations of Computer Science*, 2005, pp. 585–595., 2008.

[27] G. Kol and M. Naor. Cryptography and game theory: Designing protocols for exchanging information. In *Theory of Cryptography Conference*, pages 320–339, 2008.

[28] Y. Shoham and M. Tennenholtz. Non-cooperative computing: Boolean functions with correctness and exclusivity. *Theoretical Computer Science*, 343(1–2): 97–113, 2005.

[29] D. M. Kreps and R. B. Wilson. Sequential equilibria. *Econometrica*, 50:863–894, 1982.

[30] R. Selten. Reexamination of the perfectness concept for equilibrium points in extensive games. *International Journal of Game Theory*, 4:25–55, 1975.

[31] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, N.J., 2nd edition, 1947.

[32] M. Lepinski, S. Micali, C. Peikert, and A. Shelat. Completely fair SFE and coalition-safe cheap talk. In *Proc. 23rd ACM Symposium on Principles of Distributed Computing*, pages 1–10, 2004.

[33] I. Abraham, D. Dolev, and J. Y. Halpern. Lower bounds on implementing robust and resilient mediators. In *Fifth Theory of Cryptography Conference*, pages 302–319, 2008.

[34] M. Lepinski, S. Micali, and A. Shelat. Collusion-free protocols. In *Proc. 37th ACM Symposium on Theory of Computing*, pages 543–552, 2005.

[35] D. Gerardi. Unmediated communication in games with complete and incomplete information. *Journal of Economic Theory*, 114:104–131, 2004.

[36] E. Ben-Porath. Cheap talk in games with incomplete information. *Journal of Economic Theory*, 108(1): 45–71, 2003.

[37] S. Micali and R. Pass. Local zero knowledge. In *Proc. 38th ACM Symposium on Theory of Computing*, pages 306–315, 2006.

[38] S. Micali and R. Pass. Precise cryptography. Available at http://www.cs.cornell.edu/~rafael, 2007.

[39] C. Dwork, M. Naor, O. Reingold, and L. J. Stockmeyer. Magic functions. *Journal of the ACM*, 50(6): 852–921, 2003.

[40] M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000.

[41] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM Symposium on Theory of Computing*, pages 1–10, 1988.

[42] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity, or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.

[43] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay—secure two-party computation system. In *Proc. 13th USENIX Security Symposium*, pages 287–302, 2004.

[44] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Proc. of Theory of Cryptography Conference*, 2006.

[45] O. Goldreich. *Foundations of Cryptography, Vol. 2.* Cambridge University Press, 2004.

[46] S. Goldwasser and Y. Lindell. Secure computation without agreement. In *DISC '02: Proc. 16th Inter-*

national Conference on Distributed Computing, pages 17–32. Springer-Verlag, 2002.

[47] D. Boneh and M. Naor. Timed commitments. In *Proc. CRYPTO 2000*, Lecture Notes in Computer Science, Volume 1880, pages 236–254. Springer-Verlag, 2000.

[48] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[49] S. Micali and A. Shelat. Purely rational secret sharing. In *Proc. TCC 2009*, 2009.

[50] T. Cohen, J. Kilian, and E. Petrank. Responsive round complexity and concurrent zero-knowledge. In *Advances in Cryptology: ASIACRYPT 2001*, pages 422–441, 2001.

[51] J. Y. Halpern. On ambiguities in the interpretation of game trees. *Games and Economic Behavior*, 20: 66–96, 1997.

[52] J. Y. Halpern, Y. Moses, and M. R. Tuttle. A knowledge-based analysis of zero knowledge. In *Proc. 20th ACM Symposium on Theory of Computing*, pages 132–147, 1988.

[53] J. Y. Halpern, Y. Moses, and M. Y. Vardi. Algorithmic knowledge. In *Theoretical Aspects of Reasoning about Knowledge: Proc. Fifth Conference*, pages 255–266. 1994.

[54] Y. Moses. Resource-bounded knowledge. In *Proc. Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 261–276. 1988.

[55] J. Y. Halpern, R. Pass, and V. Raman. An epistemic characterization of zero knowledge. In *Theoretical Aspects of Rationality and Knowledge: Proc. Twelfth Conference (TARK 2009)*, pages 156–165, 2009.

[56] R. J. Aumann. Acceptable points in general cooperative n-person games. In A. W. Tucker and R. D. Luce, editors, *Contributions to the Theory of Games IV, Annals of Mathematical Studies 40*, pages 287–324. Princeton University Press, Princeton, N. J., 1959.

[57] R. Pass. *A Precise Computational Approach to Knowledge*. PhD thesis, MIT, 2006. Available at http://www.cs.cornell.edu/˜rafael.

Appendix

A Coalition Machine Games

We strengthen the notion of Nash equilibrium to allow for deviating coalitions. Towards this goal, we consider a generalization of Bayesian machine games called *coalition machine games*, where, in the spirit of *coalitional games* [31], each *subset* of players has a complexity function and utility function associated with it. In analogy with the traditional notion of Nash equilibrium, which considers only "single-player" deviations, we consider only "single-coalition" deviations.

More precisely, given a subset Z of $[m]$, we let $-Z$ denote the set $[m]/Z$. We say that a machine M'_Z *controls* the players in Z if M'_Z controls the input and output tapes of the players in set Z (and thus can coordinate their outputs). In addition, the adversary that controls Z has its own input and output tape. A *coalition machine game* G is described by a tuple $([m], \mathcal{M}, \Pr, \vec{\mathscr{C}}, \vec{u})$, where $\vec{\mathscr{C}}$ and $\vec{u}$ are sequences of complexity functions $\mathscr{C}_Z$ and utility functions u_Z, respectively, one for each subset Z of $[m]$; m, $\mathcal{M}$, and Pr are defined as in Definition 2.1. In contrast, the utility function u_Z for the set Z is a function $T \times (\{0,1\}^*)^m \times (I\!N \times I\!N^{m-|Z|+1}) \rightarrow I\!R$, where $u_Z(\vec{t}, \vec{a}, (c_Z, \vec{c}_{-Z}))$ is the utility of the coalition Z if $\vec{t}$ is the (length $m+1$) type profile, $\vec{a}$ is the (length m) action profile (where we identify i's action as player i output), c_Z is the complexity of the coalition Z, and $\vec{c}_{-Z}$ is the (length $m - |Z|$) profile of machine complexities for the players in $-Z$. The complexity c_Z is a measure of the complexity according to whoever controls coalition Z of running the coalition. Note that even if the coalition is controlled by a machine M'_Z that lets each of the players in Z perform independent computations, the complexity of M'_Z is not necessarily some function of the complexities c_i of the players $i \in Z$ (such as the sum or the max). Moreover, while cooperative game theory tends to focus on *superadditive* utility functions, where the utility of a coalition is at least the sum of the utilities of any partition of the coalition into sub-coalitions or individual players, we make no such restrictions; indeed when taking complexity into account, it might very well be the case that larger coalitions are more expensive than smaller ones. Also note that, in our calculations, we assume that, other than the coalition Z, all the other players play individually (so that we use c_i for $i \notin Z$); there is at most one coalition in the picture. Having defined u_Z, we can define the expected utility of the group Z in the obvious way.

The *benign machine for coalition* Z, denoted M^b_Z, is the one where that gives each player $i \in Z$ its true input, and each player $i \in Z$ outputs the output of M_i; M^b_Z write nothing on its output tape. Essentially, the benign machine does exactly what all the players in the coalition would have done anyway. We now extend the notion of Nash equilibrium to deal with coalitions; it requires that in an equilibrium $\vec{M}$, no coalition does (much) better than it would using the benign machine, according to the utility function for that coalition.

Definition A.1 (NE in coalition machine games) *Given an m-player coalition machine game G, a machine profile $\vec{M}$, a subset Z of $[m]$ and $\epsilon \geq 0$, M^b_Z is an ϵ-best response to $\vec{M}_{-Z}$ if, for every coalition* *machine $M'_Z \in \mathcal{M}$,*

$$U^G_Z[(M^b_Z, \vec{M}_{-Z})] \geq U^G_Z[(M'_Z, \vec{M}_{-Z})] - \epsilon.$$

*Given a set $\mathcal{Z}$ of subsets of $[m]$, $\vec{M}$ is a $\mathcal{Z}$-*safe ϵ-Nash equilibrium *for G if, for all $Z \in \mathcal{Z}$, M^b_Z is an ϵ-best response to $\vec{M}_{-Z}$.*

Our notion of coalition games is quite general. In particular, if we disregard the costs of computation, it allows us to capture some standard notions of coalition resistance in the literature, by choosing u_Z appropriately. For example, Aumann's [56] notion of *strong equilibrium* requires that, for all coalitions, it is not the case that there is a deviation that makes everyone in the coalition strictly better off. To capture this, fix a profile $\vec{M}$, and define $u^{\vec{M}}_Z(M'_Z, \vec{M'}_{-Z}) = \min_{i\in Z} u_i(M'_Z, \vec{M'}_{-Z}) - u_i(\vec{M})$.[11] We can capture the notion of *k-resilient equilibrium* [23, 33], where the only deviations allowed are by coalitions of size at most k, by restricting $\mathcal{Z}$ to consist of sets of cardinality at most k (so a 1-resilient equilibrium is just a Nash equilibrium). Abraham et al. [23, 33] also consider a notion of *strong* k-resilient equilibrium, where there is no deviation by the coalition that makes even one coalition member strictly better off. We can capture this by replacing the min in the definition of $u^{\vec{M}}_Z$ by max.

B Precise Secure Computation

In this section, we review the notion of precise secure computation [37, 38], which is a strengthening of the traditional notion of secure computation [12]. We consider a system where players are connected through secure (i.e., authenticated and private) point-to-point channels. We consider a malicious adversary that is allowed to corrupt a subset of the m players before the interaction begins; these players may then deviate arbitrarily from the protocol. Thus, the adversary is *static*; it cannot corrupt players based on history.

As usual, the security of protocol $\vec{M}$ for computing a function f is defined by comparing the *real execution* of $\vec{M}$ with an *ideal execution* where all players directly talk to a trusted third party (i.e., a mediator) computing f. In particular, we require that the outputs of the players in both of these executions cannot be distinguished, and additionally that the view of the adversary in the real execution can be reconstructed by the ideal-execution adversary (called the *simulator*). Additionally, *precision* requires that the running-time of the simulator in each run of the ideal execution is closely related to the running time of the real-execution adversary in the (real-execution) view output by the simulator.

The ideal execution Let f be an m-ary functionality. Let $\tilde{A}$ be a probabilistic polynomial-time machine (representing the ideal-model adversary) and suppose that $\tilde{A}$ controls the players in $Z \subseteq [m]$. We characterize the *ideal execution of f* given adversary $\tilde{A}$ using a function denoted $\textsc{ideal}_{f,\tilde{A}}$ that maps an input vector $\vec{x}$, an auxiliary input z, and a tuple $(r_{\tilde{A}}, r_f) \in (\{0,1\}^\infty)^2$ (a random string for the adversary $\tilde{A}$ and a random string for the trusted third party) to a triple $(\vec{x}, \vec{y}, v)$, where $\vec{y}$ is the output vector of the players $1, \ldots, m$, and v is the output of the adversary $\tilde{A}$ on its tape given input $(z, \vec{x}, r_{\tilde{A}})$, computed according to the following three-stage process.

In the first stage, each player i receives its input x_i. Each player $i \notin Z$ next sends x_i to the trusted party. (Recall that in the ideal execution, there is a trusted third party.) The adversary $\tilde{A}$ determines the value $x'_i \in \{0,1\}^*$ a player $i \in Z$ sends to the trusted party. We assume that the system is synchronous, so the trusted party can tell if some player does not send a message; if player i does not send a message, i is taken to have sent λ. Let $\vec{x}'$ be the vector of values received by the trusted party. In the second stage, the trusted party computes $y_i = f_i(\vec{x}', r_f)$ and sends y_i to P_i for every $i \in [m]$. Finally, in the third stage, each player $i \notin Z$ outputs the value y_i received from the trusted party. The adversary $\tilde{A}$ determines the output of the players $i \in Z$. $\tilde{A}$ finally also outputs an arbitrary value v (which is supposed to be the "reconstructed" view of the real-execution adversary A). Let $\mathsf{view}_{f,\tilde{A}}(\vec{x}, z, \vec{r})$ denote the the view of $\tilde{A}$ in this execution. We occasionally abuse notation and suppress the random strings, writing $\textsc{ideal}_{f,\tilde{A}}(\vec{x}, z)$ and $\mathsf{view}_{f,\tilde{A}}(\vec{x}, z)$; we can think of $\textsc{ideal}_{f,\tilde{A}}(\vec{x}, z)$ and $\mathsf{view}_{f,\tilde{A}}(\vec{x}, z)$ as random variables.

The real execution Let f be an m-ary functionality, let Π be a protocol for computing f, and let A be a machine that controls the same set Z of players as $\tilde{A}$. We characterize the *real execution of* Π given adversary A using a function denoted $\textsc{real}_{\Pi,A}$ that maps an input vector $\vec{x}$, an auxiliary input z, and a tuple $\vec{r} \in (\{0,1\}^\infty)^{m+1-|Z|}$ ($m - |Z|$ random strings for the players not in Z and a random string for the adversary A), to a triple $(\vec{x}, \vec{y}, v)$, where $\vec{y}$ is the output of players $1, \ldots, m$, and v is the view of A that results from executing protocol Π on inputs $\vec{x}$, when players $i \in Z$ are controlled by the adversary A, who is given auxiliary input z. As before, we often suppress the vector of random bitstrings $\vec{r}$ and write $\textsc{real}_{\Pi,A}(\vec{x}, z)$.

[11]Note that if we do not disregard the cost of computation, it is not clear how to define the individual complexity of a player that is controlled by $M'_{\mathcal{Z}}$.

We now formalize the notion of precise secure computation. For convenience, we slightly generalize the definition of [37] to consider *general adversary structures* [40]. More precisely, we assume that the specification of a secure computation protocol includes a set $\mathcal{Z}$ of subsets of players, where the adversary is allowed to corrupt only the players in one of the subsets in $\mathcal{Z}$; the definition of [12, 37] considers only *threshold adversaries* where $\mathcal{Z}$ consists of all subsets up to a pre-specified size k. We first provide a definition of precise computation in terms of running time, as in [37], although other complexity functions could be used; we later consider general complexity functions.

Let $\textsc{steps}$ be the complexity function that, on input a machine M and a view v, roughly speaking, gives the number of "computational steps" taken by M in the view v. In counting computational steps, we assume a representation of machines such that a machine M, given as input an encoding of another machine A and an input x, can emulate the computation of A on input x with only linear overhead. (Note that this is clearly the case for "natural" memory-based models of computation. An equivalent representation is a universal Turing machine that receives the code it is supposed to run on one input tape.)

In the following definition, we say that a function is *negligible* if it is asymptotically smaller than the inverse of any fixed polynomial. More precisely, a function $\nu : I\!N \to I\!R$ is negligible if, for all $c > 0$, there exists some n_c such that $\nu(n) < n^{-c}$ for all $n > n_c$.

Roughly speaking, a computation is secure if the ideal execution cannot be distinguished from the real execution. To make this precise, a *distinguisher* is used. Formally, a distinguisher gets as input a bitstring z, a triple $(\vec{x}, \vec{y}, v)$ (intuitively, the output of either $\textsc{ideal}_{f,\tilde{A}}$ or $\textsc{real}_{\Pi,A}$ on $(\vec{x}, z)$ and some appropriate-length tuple of random strings) and a random string r, and outputs either 0 or 1. As usual, we typically suppress the random bitstring and write, for example, $D(z, \textsc{ideal}_{f,\tilde{A}}(\vec{x}, z))$ or $D(z, \textsc{real}_{\Pi,A}(\vec{x}, z))$.

Definition B.1 (Precise Secure Computation) *Let f be an m-ary function, Π a protocol computing f, $\mathcal{Z}$ a set of subsets of $[m]$, $p : I\!N \times I\!N \to I\!N$, and $\epsilon : I\!N \to I\!R$. Protocol Π is a $\mathcal{Z}$-*secure computation of f with precision p and ϵ-statistical error *if, for all $Z \in \mathcal{Z}$ and every real-model adversary A that controls the players in Z, there exists an ideal-model adversary $\tilde{A}$, called the* simulator*, that controls the players in Z such that, for all $n \in N$, all $\vec{x} = (x_1, \ldots, x_m) \in (\{0,1\}^n)^m$, and all $z \in \{0,1\}^*$, the following conditions hold:*

1. *For every distinguisher D,*

$$\Big| \Pr_U[D(z, \textsc{real}_{\Pi,A}(\vec{x}, z)) = 1] - \Pr_U[D(z, \textsc{ideal}_{f,\tilde{A}}(\vec{x}, z))] = 1 \Big| \le \epsilon(n).$$

2. $\Pr_U[\textsc{steps}(\tilde{A}, v \le p(n, \textsc{steps}(A, \tilde{A}(v)))]) = 1$, *where $v = \mathsf{view}_{f,\tilde{A}}(\vec{x}, z))$.*[12]

Π *is a $\mathcal{Z}$-*secure computation of f with precision p and (T, ϵ)-computational error *if it satisfies the two conditions above with the adversary A and the distinguisher D restricted to being computable by a TM with running time bounded by $T(\cdot)$.*

Protocol Π *is a* $\mathcal{Z}$-secure computation of f with statistical precision p *if there exists some negligible function ϵ such that Π is a $\mathcal{Z}$-secure computation of f with precision p and ϵ-statistical error. Finally, protocol* Π *is a* $\mathcal{Z}$-secure computation of f with computational precision p *if for every polynomial T, there exists some negligible function ϵ such that Π is a $\mathcal{Z}$-secure computation of f with precision p and (T, ϵ)-computational error.*

The traditional notion of secure computation is obtained by replacing condition 2 with the requirement that the worst-case running-time of $\tilde{A}$ is polynomially related to the worst-case running time of A.

The following theorems were provided by Micali and Pass [37, 38], using the results of Ben-Or, Goldwasser and Wigderson [41] and Goldreich, Micali and Wigderson [12]. Let $\mathcal{Z}_t^m$ denote all the subsets of $[m]$ containing t or less elements. An m-ary functionality f is said to be *well-formed* if it essentially ignores arguments that are not in $(\{0,1\}^n)^m$ for some n. More precisely, if there exist j, j' such that $|x_j| \ne |x_{j'}|$, then $f_i(\vec{x}) = \lambda$ for all $i \in [m]$. (See [45, p. 617] for motivation and more details.)

Theorem B.2 *For every well-formed m-ary functionality f, there exists a precision function p such that $p(n, t) = O(t)$ and a protocol Π that $\mathcal{Z}^m_{\lceil m/3 \rceil - 1}$-securely computes f with precision p and 0-statistical error.*

This result can also be extended to more general adversary structures by relying on the results of [40]. We can also consider secure computation of specific 2-party functionalities.

[12]Note that the three occurrences of $\Pr_U$ in the first two clauses represent slightly different probability measures, although this is hidden by the fact that we have omitted the superscripts. The first occurrence of $\Pr_U$ should be $\Pr_U^{m-|Z|+3}$, since we are taking the probability over the $m + 2 - |Z|$ random inputs to $\textsc{real}_{f,A}$ and the additional random input to D; similarly, the second and third occurrences of $\Pr_U$ should be $\Pr_U^3$.

Theorem B.3 *Suppose that there exists an enhanced trapdoor permutation.*[13] *For every well-formed* 2*-ary functionality* f *where only one party gets an output (i.e.,* $f_1(\cdot) = 0$*), there exists a a precision function* p *such that* $p(n,t) = O(t)$ *and protocol* Π *that* $\mathcal{Z}_1^2$*-securely computes* f *with computational-precision* p*.*

Micali and Pass [37] also obtain unconditional results (using statistical security) for the special case of zero-knowledge proofs. We refer the reader to [37, 57] for more details.

B.1 Weak Precise Secure Computation

Universal implementation is not equivalent to precise secure computation, but to a (quite natural) weakening of it. *Weak precise secure computation*, which we are about to define, differs from precise secure computation in the following respects:

- Just as in the traditional definition of zero knowledge [13], precise zero knowledge requires that for every adversary, there exists a simulator that, on all inputs, produces an interaction that no distinguisher can distinguish from the real interaction. This simulator must work for all inputs and all distinguishers. In analogy with the notion of "weak zero knowledge" [39], we here switch the order of the quantifiers and require instead that for every input distribution Pr over $\vec{x} \in (\{0,1\}^n)^m$ and $z \in \{0,1\}^*$, and every distinguisher D, there exists a (precise) simulator that "tricks" D; in essence, we allow there to be a different simulator for each distinguisher. As argued by Dwork et al. [39], this order of quantification is arguably reasonable when dealing with concrete security. To show that a computation is secure in every concrete setting, it suffices to show that, in every concrete setting (where a "concrete setting" is characterized by an input distribution and the distinguisher used by the adversary), there is a simulator.
- We further weaken this condition by requiring only that the probability of the distinguisher outputting 1 on a real view be (essentially) no higher than the probability of outputting 1 on a simulated view. In contrast, the traditional definition requires these probabilities to be (essentially) equal. If we think of the distinguisher outputting 1 as meaning that the adversary has learned some important feature, then we are saying that the likelihood of the adversary learning an important feature in the real execution is essentially no higher than that of the adversary learning an important feature in the "ideal" computation. This condition on the distinguisher is in keeping with the standard intuition of the role of the distinguisher.
- We allow the adversary and the simulator to depend not only on the probability distribution, but also on the particular security parameter n (in contrast, the definition of [39] is *uniform*). That is why, when considering weak precise secure computation with (T,ϵ)-computational error, we require that the adversary A and the simulator D be computable by *circuits* of size at most $T(n)$ (with a possibly different circuit for each n), rather than a Turing machine with running time $T(n)$. Again, this is arguably reasonable in a concrete setting, where the security parameter is known.
- We also allow the computation not to meet the precision bounds with a small probability. The obvious way to do this is to change the requirement in the definition of precise secure computation by replacing 1 by $1-\epsilon$, to get

$$\Pr_U[\textsc{steps}(\tilde{A}, v) \le p(n, \textsc{steps}(A, \tilde{A}(v))] \ge 1-\epsilon(n),$$

where n is the input length and $v = \mathsf{view}_{f,\tilde{A}}(\vec{x}, z)$ We change this requirement in two ways. First, rather than just requiring that this precision inequality hold for all $\vec{x}$ and z, we require that the probability of the inequality holding be at least $1-\epsilon$ for all distributions Pr over $\vec{x} \in (\{0,1\}^n)^m$ and $z \in \{0,1\}^*$.
The second difference is to add an extra argument to the distinguisher, which tells the distinguisher whether the precision requirement is met. In the real computation, we assume that the precision requirement is always met, thus, whenever it is not met, the distinguisher can distinguish the real and ideal computations. We still want the probability that the distinguisher can distinguish the real and ideal computations to be at most $\epsilon(n)$. For example, our definition disallows a scenario where the complexity bound is not met with probability $\epsilon(n)/2$ and the distinguisher can distinguish the computations with (without taking the complexity bound into account) with probability $\epsilon(n)/2$.
- In keeping with the more abstract approach used in the definition of robust implementation, the definition of weak precise secure computation is parametrized by the abstract complexity measure $\mathscr{C}$, rather than using $\textsc{steps}$. This just gives us a more general definition; we can always instantiate $\mathscr{C}$ to measure running time.

[13]See [45] for a definition of enhanced trapdoor permutations; the existence of such permutations is implied by the "standard" hardness of factoring assumptions.

Definition B.4 (Weak Precise Secure Computation) *Let f, Π, $\mathcal{Z}$, p, and ϵ be as in the definition of precise secure computation, and let $\vec{\mathscr{C}}$ be a complexity function. Protocol Π is a* weak $\mathcal{Z}$-secure computation of f with $\vec{\mathscr{C}}$-precision p and ϵ-statistical error *if, for all $n \in N$, all $Z \in \mathcal{Z}$, all real-execution adversaries A that control the players in Z, all distinguishers D, and all probability distributions* $\Pr$ *over $(\{0,1\}^n)^m \times \{0,1\}^*$, there exists an ideal-execution adversary $\tilde{A}$ that controls the players in Z such that*

$$\Pr{}^+(\{(\vec{x},z) : D(z, \text{REAL}_{\Pi,A}(\vec{x},z), 1) = 1\}) - \Pr{}^+(\{(\vec{x},z) : D(z, \text{IDEAL}_{f,\tilde{A}}(\vec{x},z), p)) = 1\}) \leq \epsilon(n),$$

where $p = \mathsf{precise}_{Z,A,\tilde{A}}(n, \mathsf{view}_{f,\tilde{A}}(\vec{x},z)))$, and $\mathsf{precise}_{Z,A,\tilde{A}}(n,v) = 1$ if and only if $\mathscr{C}_Z(\tilde{A},v) \leq p(n, \mathscr{C}_Z(A, \tilde{A}(v)))$.[14] Π *is a* weak $\mathcal{Z}$-secure computation of f with $\vec{\mathscr{C}}$-precision p and (T,ϵ)-computational error *if it satisfies the condition above with the adversary A and the distinguisher D restricted to being computable by a randomized circuit of size $T(n)$. Protocol* Π *is a* $\mathcal{Z}$-weak secure computation of f with statistical $\vec{\mathscr{C}}$-precision p *if there exists some negligible function ϵ such that Π is a $\mathcal{Z}$-weak secure computation of f with precision p and statistical ϵ-error. Finally, Protocol* Π *is a* $\mathcal{Z}$-weak secure computation of f with computational $\vec{\mathscr{C}}$-precision p *if for every polynomial $T(\cdot)$, there exists some negligible function ϵ such that Π is a $\mathcal{Z}$-weak secure computation of f with precision p and (T,ϵ)-computational error.*

Our terminology suggests that weak precise secure computation is weaker than precise secure computation. This is almost immediate from the definitions if $\mathscr{C}_Z(M,v) = \text{STEPS}(M,v)$ for all $Z \in \mathcal{Z}$. A more interesting setting considers a complexity measure that can depend on $\text{STEPS}(M,v)$ and the size of the description of M. It directly follows by inspection that Theorems B.2 and B.3 also hold if, for example, $\mathscr{C}_Z(M,v) = \text{STEPS}(M,v) + O(|M|)$ for all $Z \in \mathcal{Z}$, since the simulators in those results incur only a constant additive overhead in size. (This is not a coincidence. As argued in [37, 57], the definition of precise simulation guarantees the existence of a "universal" simulator S, with "essentially" the same precision, that works for *every* adversary A, provided that S also gets the code of A; namely given a real-execution adversary A, the ideal-execution adversary $\tilde{A} = S(A)$.[15] Since $|S| = O(1)$, it follows that $|\tilde{A}| = |S| + |A| = O(|A|)$.) That is, we have the following variants of Theorems B.2 and B.3:

Theorem B.5 *For every well-formed m-ary functionality f, $\mathscr{C}_Z(M,v) = \text{STEPS}(M,v) + O(|M|)$ for all sets Z, there exists a precision function p such that $p(n,t) = O(t)$ and a protocol Π that weak $\mathcal{Z}^m_{\lceil m/3\rceil -1}$-securely computes f with $\vec{\mathscr{C}}$-precision p and 0-statistical error.*

Theorem B.6 *Suppose that there exists an enhanced trapdoor permutation, and $\mathscr{C}_Z(M,v) = \text{STEPS}(M,v) + O(|M|)$ for all sets Z. For every well-formed 2-ary functionality f where only one party gets an output (i.e., $f_1(\cdot) = \lambda$), there exists a precision function p such that $p(n,t) = O(t)$ and a protocol Π that weak $\mathcal{Z}^2_1$-securely computes f with computational $\vec{\mathscr{C}}$-precision p.*

It is easy to see that the theorems above continue to hold when considering "coarse" versions of the above complexity functions, where, say, n^2 computational steps (or size) correspond to one unit of complexity (in canonical machine game with input length n).

C Output-invariant complexity functions

Recall that for one direction of our main theorem we require that certain operations, like moving output from one tape to another, do not incur any additional complexity. We now make this precise.

Recall that in the definition of a secure computation, the ideal-execution adversary, M_Z, is an algorithm that controls the players in Z and finally provides an output for each of the players it controls and additionally produces an output of its own (which is supposed to be the reconstructed view of the real-execution adversary). Roughly speaking, a complexity function is output-invariant if M_Z can "shuffle" content between its tapes at no cost.

Definition C.1 *A complexity function $\mathscr{C}$ is* output-invariant *if, for every set Z of players, there exists some canonical player $i_Z \in Z$ such that the following three conditions hold:*

1. **(Outputting view)** *For every machine M_Z controlling the players in Z, there exists some machine M'_Z with the same complexity as M_Z such that the output of $M'_Z(v)$ is identical to $M_Z(v)$ except that player i_Z outputs $y;v$, where y is the output of i_Z in the execution of $M_Z(v)$ (i.e., M'_Z is identical to M_Z with the only exception being that player i_Z also outputs the view of M'_Z).*

[14] Recall that $\Pr^+$ denotes the product of $\Pr$ and $\Pr_U$ (here, the first $\Pr^+$ is actually $\Pr^{+(m+3-|Z|)}$, while the second is $\Pr^{+3}$).

[15] This follows by considering the simulator S for the universal TM (which receives the code to be executed as auxiliary input).

2. **(Moving content to a different tape)** *For every machine M'_Z controlling players Z, there exists some machine M_Z with the same complexity as M_Z such that the output of $M_Z(v)$ is identical to $M'_Z(v)$ except if player i_Z outputs $y; v'$ for some $v' \in \{0,1\}^*$ in the execution of $M'_Z(v)$. In that case, the only difference is that player i_Z outputs only y and $M_Z(v)$ outputs v'.*
3. **(Duplicating content to another output tape)** *For every machine M'_Z controlling players Z, there exists some machine M_Z with the same complexity as M_Z such that the output of $M_Z(v)$ is identical to $M'_Z(v)$ except if player i_Z outputs $y; v'$ for some $v' \in \{0,1\}^*$ in the execution of $M'_Z(v)$. In that case, the only difference is that $M_Z(v)$ outputs v'.*

Note that the only difference between condition 2 and 3 is that in condition 2, player i_z only outputs y, whereas in condition 3 it still outputs its original output $y; v'$.

We stress that we need to consider output-invariant complexity functions only to show that universal implementation implies precise secure computation.