Completely Fair SFE and Coalition-Safe Cheap Talk

Matt Lepinski* Silvio Micali Chris Peikert Abhi Shelat

MIT CSAIL
32 Vassar Street, G636
Cambridge, MA 02139

{lepinski,silvio,cpeikert,abhi}@csail.mit.edu

ABSTRACT

Secure function evaluation (SFE) enables a group of players, by themselves, to evaluate a function on private inputs as securely as if a trusted third party had done it for them. A *completely fair* SFE is a protocol in which, conceptually, the function values are learned *atomically*.

We provide a completely fair SFE protocol which is secure for *any number* of malicious players, using a novel combination of computational and physical channel assumptions.

We also show how completely fair SFE has striking applications to game theory. In particular, it enables "cheap-talk" protocols that

(a) achieve correlated-equilibrium payoffs in any game,

(b) are the first protocols which *provably* give no additional power to any coalition of players, and

(c) are exponentially more efficient than prior counterparts.

Categories and Subject Descriptors: F.0 [Theory of Computation]: General

General Terms: Theory, Security, Economics

Keywords: Game Theory, Secure Function Evaluation, Correlated Equilibrium, Mechanism Design

1. INTRODUCTION

Prior Secure Function Evaluation

Suppose a group of n players — each possessing his own private input x_i — wish to evaluate on their inputs a function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ so that each player i learns exactly y_i. This would not be a problem if our players trusted an external party, but they trust no one. What they need is, at a minimum, an SFE protocol, and better yet, a completely fair one.

*This material is based upon work supported under an NSF Graduate Research Fellowship.

PODC'04, July 25–28, 2004, St. Johns, Newfoundland, Canada.

Quite informally, an *SFE protocol* is an efficient communication protocol enabling our players to evaluate f both privately and correctly by themselves, even though some of them may arbitrarily deviate from their prescribed instructions. No matter what they might do, bad players cannot learn anything about an honest party's input or output (other that what is implied by their own inputs and outputs), nor can they cause any honest party to compute an incorrect or inconsistent value as his output.

An SFE protocol is called *completely fair* if it additionally guarantees that all outputs are learned by the respective players in a conceptually "atomic" step. That is, no matter what dishonest players might do, if *any* player learns any partial knowledge about his own output, then *all* players learn their outputs entirely. Complete fairness is indeed a useful property, and from now on an SFE protocol should be assumed not to be completely fair unless explicitly so described.

SFE protocols were introduced in the two-party setting by Yao [30] in 1986. General SFE and completely fair SFE were introduced and first exemplified by Goldreich, Micali and Wigderson [16] in 1987. More precisely, they proved two results under standard complexity assumptions and using broadcast channels for communication. First, they showed that any n-input/n-output function f has an SFE protocol withstanding *any number* of bad players. (We shall refer to this protocol as GMWa.) Second, they showed that any n-input/n-output function f has a completely fair SFE protocol withstanding *any minority* of bad players.

Unfortunately, completely fair SFE protocols require the presence of an honest majority whenever security is solely based on computational assumptions. (This follows from a result of Cleve [10], which implies that unless physical assumptions are made, even simple functions do not have completely fair SFE protocols without an honest majority among the players.[1]) Completely fair SFE protocols solely relying on physical assumptions (namely, the perfect security of the communication channels between every pair of players) have also been discovered [4, 9, 25], but they too require an honest majority of players —indeed, even for being correct, let alone fair!

Without an honest majority, only weaker notions of fairness were known to be achievable. The first such notion was originally proposed by Blum [**?**] and Even, Goldreich and Lempel [**?**] in special contexts such as contract signing.

[1]In particular, n players, each having a secret random bit, cannot fairly and securely compute the exclusive-or of their bits unless a majority of the players are honest.

In essence, a protocol is considered fair if, at any point of its execution, the computational effort required to compute the outputs is roughly equal for all parties. Formally proving such fairness, however, is extremely hard and requires unusually strong complexity assumptions. Recently, Garay, MacKenzie and Yang [14] presented a general SFE protocol that is provably fair in this sense provided that, in addition to very strong complexity assumptions, (1) an upperbound to the running time of malicious players is known in advance and (2) the good players must sometimes compute more than this upperbound. A second, weaker notion of fairness was put forward (and first exemplified under traditional complexity assumptions) by Luby, Micali and Rackoff [20]. According to this notion a fair protocol is one in which (1) each player's probability of correctly guessing his own output slowly approaches 1, and (2) all players' probabilities of correctly guessing their own outputs are guaranteed to remain close to one another. This notion was extended and shown to hold for general SFE by Goldwasser and Levin [17] (who actually were the ones to coin the term "completely fair SFE" to differentiate it from this second, weaker notion). Neither of these two weaker definitions of fairness, however, suffices for our game theoretic applications: what we need is a reasonable model to achieve completely fair SFE without an honest majority.

Our Completely Fair SFE

We put forward an SFE protocol that is completely fair for any number of players and for any number of bad players (i.e., from 1 to $n-1$, and trivially even n itself).

Notably, our protocol relies on *both* computational assumptions and physical assumptions about our communication channel. Our reliance on physical assumptions is necessary in view of the mentioned result of Cleve [10]. Specifically, we assume that the players have access to communication channels akin to ideal *envelopes* (see Section 2.2). We note that similar physical assumptions are actually standard in the game theory community.[2]

Our Applications to Game Theory

It is well known in game theory that *correlated equilibria*, introduced by Aumann [1], are a desirable generalization of traditional Nash equilibria. Not only do correlated equilibria yield higher and more flexible payoffs, but they are always computable in polynomial time from the binary representation of a game — a property which is not known to be the case for Nash equilibria. Unfortunately, correlated equilibria cannot be achieved by the players themselves, but require the use of an external trusted party or device. Starting with Bárány [2] in 1988, many ways have been proposed [5, 15, 28, 11, 27], in various restricted settings, to replace any such external device by *cheap talk* (i.e., by a protocol in which the players alone engage in non-binding communication before choosing their actions in the game) while guaranteeing the same correlated-equilibrium payoffs.

[2]For example, Bárány's protocol — though often informally described to be implementable via ordinary phone lines — requires a communication channel between each pair of players which is *publicly auditable*: at the request of *any* player, all past message traffic is publicly and correctly revealed. Indeed, hand delivery of messages in envelopes (which will be opened upon request) almost implements these requirements.

Prior protocols, however, require an amount of computation that is exponential in the binary representation of the correlated equilibrium.[3] But, *much more crucially,* they are extremely vulnerable to coalitions (i.e., to the coordinated action of bad players) when the number of players is greater than 2. (Coalitions are of course meaningless in 2-player games.) Indeed, all such cheap-talk protocols always empower a coalition of just two players to choose the payoffs of all players in an undetectable way! Under these circumstances, perhaps, players should avoid cheap talk altogether: in an attempt to get correlated equilibrium payoffs, players de facto put themselves at the mercy of just two rational, collusive players.

In sharp contrast, we prove that completely fair SFE yields efficient cheap-talk protocols that provably do not introduce any coalition power that is not intrinsic to the sought-after correlated equilibrium. We argue that complete fairness is the key ingredient to achieve this property of *coalition safety.*

Secure protocols, game theory, and distributed computation all deal with complex interactions among parties; we hope this paper contributes to a deeper understanding of their interconnections.

2. DEFINING COMPLETELY FAIR SFE

To define a completely fair SFE for a function f, we follow the general GMW paradigm [16]. First, we define an ideal evaluation of f, that is, one in the presence of an adversary but with the help of an external trusted party. (Conceptually, this provides our golden standard: no protocol could possibly be more secure!) Second, we define a real evaluation of f (i.e., one obtained by running a real protocol in the presence of an adversary and without the aid of any trusted party). Third, we recall a traditional measure of "closeness" in cryptography (i.e., *computational indistinguishability*). Finally, we define a completely fair SFE to be a real evaluation of f that closely mimics an ideal one in this sense.

This original paradigm has been refined over the years so as to capture important additional desiderata (most notably, *dynamic adversaries* [8] and *universal composability* [7]). In order to focus on a *strong* definition of a completely fair SFE without honest majority, however, we choose to work in a simpler setting: namely, we envisage the set of bad players to be fixed at the start, and that the protocol is executed in a stand-alone manner. Still, this setting is complex enough and we shall rely on the ideas of [21] to formalize it properly.

2.1 Ideal Evaluation

Let f be an efficiently computable function (without loss of generality a deterministic one) mapping n inputs to n outputs. Informally, in an ideal evaluation of f, the n players hand their private inputs to a trusted party, T, who then evaluates f and returns the outputs to the corresponding players. It is assumed that there exist a set of good players, G, a corresponding set of bad players, B, and an efficient ITM, the *simulator*, S, who acts on behalf of all bad players. The dynamic of this interaction is detailed in the 2-round protocol below, where z is a binary string and x_i the binary, secret input of good player i.

A random *ideal evaluation of* f with security parameter k, auxiliary input z, set of good players G (and cor-

[3]This is not self-evident, but we provide constructive proofs of it in Section 5.

responding set of bad players B), good inputs $\vec{x}_G = \{x_i \in \{0,1\}^* : i \in G\}$, and adversary S is generated as follows: first, the random tape of S is initialized with a random string R_S. Then, on inputs z and 1^k, S computes for each bad player j an *effective input* $x_j \in \{0,1\}^* \cup \perp$. In the first round, all $x_1, \ldots, x_n$ are privately sent to T. In the second round, if any $x_j = \perp$, then T returns the output $\perp$ to every good player. Else, T computes the outputs $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and, for every good player i, privately returns y_i to i, and, for every bad player j, privately returns y_j to S. Upon receiving the bad players' outputs, S computes a *final string*, ω, and halts.

We denote by $\text{IDEAL}_S^f(1^k, z, G, \vec{x}_G)$ the random variable consisting of the following three components:

1. the sequence of inputs and effective inputs $(x_1, \ldots, x_n)$;
2. the sequence of outputs: either $(\perp, \ldots, \perp)$ or $(y_1, \ldots, y_n)$;
3. the final string ω output by S.

2.2 Real-Protocol Execution

We assume that the reader is familiar with efficient (i.e., probabilistic polynomial time) interactive Turing machines (ITMs), which are the standard formalization of parties in a protocol. Protocol executions proceed in rounds, with the players sending their messages over communication channels — in our case, authenticated broadcast and ideal envelopes. (Formal details on how a protocol is executed, in a setting similar to ours, can be found, for example, in [13].)

ADVERSARIES AND PROTOCOLS. A *(static) adversary* is an efficient ITM. An n-party *protocol* Π is a sequence of n ITMs: $\Pi = (P_1, \ldots, P_n)$. We say that Π is efficient if each of its ITMs is efficient.

COMMUNICATION CHANNELS. Our "envelope channel" satisfies the following properties: (1) every envelope has a visible, distinct identifier; (2) every envelope hides all information about its content for as long as it stays sealed, (3) an envelope reveals its full original content once opened; (4) an envelope can be opened either privately by a single player — in which case its content will be learned only by that player — or publicly — in which case its content will be learned by all players; and (5) it is possible to deliver different envelopes to different parties so that no envelope is opened before all envelopes are properly delivered. Then, every envelope is privately opened by its recipient.

EXECUTING A PROTOCOL WITH AN ADVERSARY. Let $\Pi = (P_1, \ldots, P_n)$ be an n-player protocol and A an adversary. In an execution e of Π with A, security parameter k, set of good players $G \subset \{1, \ldots, n\}$, and good inputs $\vec{x}_G = \{x_i \in \{0,1\}^* : i \in G\}$,

- A starts with common input 1^k, private inputs n and the set of bad players $B = \{1, \ldots, n\} - G$, auxiliary input $z \in \{0,1\}^*$ and a random tape $R_A \in \{0,1\}^\infty$;
- for all $i \in G$, player i runs P_i with common input 1^k, private input x_i and random tape $R_i \in \{0,1\}^\infty$.

(Notice that honest players are not explicitly told the composition or the existence of B, but may deduce something about it from the way the execution unfolds.) We envisage the strongest, natural form of bad players' behavior. Namely, e proceeds by having all players compute and post their messages over the right channels, except that we allow A to invisibly replace and thus "perfectly coordinate" all bad players. Each message that should be sent by a bad player j over a given channel is actually computed and sent by A over the same channel (and is received by all the good players as though coming from j). Every message sent by a good player i over a given channel is actually received by A (knowing that it came from i) whenever it should be received a bad player j. If $i \in G$ and P_i halts, then prior to halting P_i produces a private output string.

A random execution of Π with A is one in which each bit of every random tape is randomly and independently selected.

An execution of a protocol Π with adversary A *terminates* at round r if r is the first round in which all good players have halted. A protocol is a r-round protocol if (1) in any execution without an adversary it terminates at round r and (2) in any execution with any adversary it terminates at a round $\leq r$.

AUGMENTED VIEWS. Consider an execution e of a protocol Π with an adversary A (with auxiliary input z and random tape R_A), and denote by M_A the sequence of messages received by A (in the order in which they are received) and, for any integer r, by M_A^r be the sequence of messages received by A up to and including round r. Then, we define (1) the *augmented adversary view* of e, denoted $\text{VIEW}_A(e)$, to be the quadruple $(z, R_A, M_A, \hat{A})$ — where $\hat{A}$ denotes the description of ITM A in some standard encoding — and (2) the *augmented adversary view up to round r of e*, denoted $\text{VIEW}_A^r(e)$ to be the quadruple $(z, R_A, M_A^r, \hat{A})$.

2.3 Completely Fair SFE Protocols

We say that a quadruple $((n, r, \rho), \Pi, \mathcal{AI}, \mathcal{AO})$ is a *(n-player, r-round) potential SFE* if there exists positive integers r and n such that

- n, r, and ρ are integers, where n and r are greater than 2 and ρ, referred to as the *effective-input round*, is less than r,
- Π is an efficient, n-player, r-round protocol,
- $\mathcal{AI}$, the *effective-input function*, is an efficient function which, for any execution e of Π with an adversary A and set of bad players B, maps the augmented view $\text{VIEW}_A^\rho(e)$ to a vector $\vec{x}_B = \{x_j^e : j \in B\}$, where either x_j^e is a binary string for every $j \in B$, or $x_j^e = \perp$ for every $j \in B$.
- $\mathcal{AO}$, the *effective-output function* is an efficient function such that, for any execution e of Π with an adversary A and set of bad players B, maps the augmented view $\text{VIEW}_A(e)$ to a vector $\vec{y}_B = \{y_j^e : j \in B\}$, where either (a) y_j^e is a binary string for every $j \in B$, or (b) $y_j^e = \perp$ for every $j \in B$.

Let $((n, r, \rho), \Pi, \mathcal{AI}, \mathcal{AO})$ be a potential SFE. Then, letting e be a random execution of Π with A, security parameter k, set of good players G, set of good inputs $\vec{x}_G = \{x_j : j \in G\}$ and auxiliary input z, and letting y_i be the output of P_i for each good player $i \in G$ in e and B the set of bad players in e, we denote by $\text{REAL}_A^\Pi(1^k, z, \vec{x}_G)$ the random variable consisting of the following three components:

1. the sequence $(x_1, \ldots, x_n)$, where $\forall j \in B$, $x_j = x_j^e$;
2. the sequence $(y_1, \ldots, y_n)$, where $\forall j \in B$, $y_j = y_j^e$; and
3. the final view of the adversary, $\text{VIEW}_A^\rho(e)$.

COMPUTATIONAL INDISTINGUISHABILITY. Our intuitive notion of a secure protocol is one in which the real execution is "close" to ideal evaluation. In cryptography, a standard notion of closeness is *computational indistinguishability.* Two sequences of probability distributions $\{X_k\}$ and $\{Y_k\}$ over finite binary strings are computationally indistinguishable if, as k grows large, X_k and Y_k become essentially interchangeable. That is, if it becomes infeasible to determine whether a given sample has been drawn from X_k or Y_k, because all efficient experiments (i.e., those whose results we hope to "see in our lifetime") yield essentially identical results.

It is often useful to write computational indistinguishability in terms of general ensembles — i.e., probability distributions indexed by a countable subset I of $\{0,1\}^*$ — using polynomial-size (distinguishing) *circuits* rather than polynomial-time (distinguishing) algorithms.[4]

By $\nu(k)$ we denote some *negligible* function, i.e., one such that, for all $c > 0$ and all sufficiently large k, $\nu(k) < 1/k^c$.

DEFINITION 1. *Two ensembles $\{X_w\}_{w\in I}$ and $\{Y_w\}_{w\in I}$ with identical index set I are said to be* computationally indistinguishable (over I) *if for every polynomial-size circuit family $\{D_k\}_{k\in\mathbb{N}}$, every sufficiently large k, and every $w \in I \cap \{0,1\}^k$, we have*

$$|\Pr[D_k(X_w) = 1] - \Pr[D_k(Y_w) = 1]| < \nu(k).$$

We are now ready to define a completely fair SFE for a function f. For generality, we allow all inputs of f to grow polynomially with the security parameter k.

Let n be an integer greater than 1, and G a subset of $\{1,\ldots,n\}$. We say that an infinite set $I \subset \{0,1\}^*$ is a *n-G-allowable input set* if

- every member of I is (the standard encoding of) a quadruple $(1^k, z, G, \vec{x}_G)$ such that $k \in N$, $z \in \{0,1\}^*$, $G \subset \{1,\ldots,n\}$, and $\vec{x}_G = \{x_i \in \{0,1\}^* : i \in G\}$; and
- there exists a constant $c > 0$ such that if $(1^k, z, G, \vec{x}_G) \in I$, then the length of z and that of each component of $\vec{x}_G$ is upper-bounded by k^c.

DEFINITION 2 (COMPLETELY FAIR SFE). *Let f be an efficiently computable function from n inputs to n outputs. We say that a 4-tuple $(\Pi, \rho, \mathcal{AI}, \mathcal{AO})$ as above is a* completely fair SFE for f *if, for every efficient adversary A, there exists a simulator S such that for all $n > 2$, $G \subset \{1,\ldots,n\}$, and n-G-allowable input sets I, the ensembles*

$$\{\text{IDEAL}_S^f(1^k, z, G, \vec{x}_G)\}_{(1^k,z,G,\vec{x}_G)\in I}$$
and
$$\{\text{REAL}_A^\Pi(1^k, z, G, \vec{x}_G)\}_{(1^k,z,G,\vec{x}_G)\in I}$$

are computationally indistinguishable over I.

A Simple Enhancement

The ideal evaluation we have described — where the trusted party outputs $\perp$ to all players whenever the simulator wishes to abort — is the standard one, and thus, in some sense, so is our definition of completely fair SFE. For our game-theoretic applications, however, we need to consider the following slight variant.

[4]Recall a polynomial-size circuit family is a sequence $\{D_k\}$ of combinatorial circuits with AND and NOT gates, such that there exists a constant d for which each D_k has at most k^d gates.

First, we enhance an ideal evaluation so that a single bad player is identified in case of abort. That is, whenever the simulator chooses $\perp$ to be the effective input of a bad player j, we require that (1) the adversary does not choose $\perp$ as the effective input of any other player, (2) the trusted party sends the pair $(j, \perp)$ to all players, and (3) the second component of IDEAL_S^f consists of $((j,\perp),\ldots,(j,\perp))$.

Then, we require that $(\Pi, \rho, \mathcal{AI}, \mathcal{AO})$ be such that IDEAL_S^f and REAL_A^Π are computationally indistinguishable. We call such a protocol *aborter-identifying*, and note that the protocol described in Section 3.2 is aborter-identifying.

3. ACHIEVING COMPLETELY FAIR SFE

We construct our completely fair SFE protocol by suitably modifying the unfair version (GMWa) of the SFE protocol in [16] (hereafter, we refer to it as GMW). Let us first recall the tools we shall use.

3.1 Tools of Our Construction

Secret-Data Testing

Secret-data testing (SDT) is a crucial element of our completely fair SFE. While this sub-protocol was originally presented in [3], we feel the need to define this primitive in full because the original paper only dealt with it in an intuitive manner.

SDT enables a prover to prove any given predicate Q about any given secret string σ to a verifier so that the only knowledge that the verifier may gain is that the prover's string (whatever it may be) satisfies Q. (For instance, σ is the binary representation of a positive integer, and Q is the predicate "this number is prime.") To make such a proof meaningful, Q is publicly known while σ is perfectly "hidden and committed:" conceptually it is the content of ideal envelope. To make such a proof possible, σ is actually encoded in a special, redundant fashion as a sequence of other strings, $\sigma_1, \sigma_2, \ldots$, each one of which is individually put into its own envelope. The proof consists of the verifier asking for a certain subsequence of envelopes to be opened, and then performing a check on the revealed strings.

Definition: An SDT protocol is a 4-tuple of polynomial time algorithms (E, D, P, V), along with positive bivariate polynomials p and q, called the *expansion rate* and the *proof length* respectively, such that:

- E and D are respectively the probabilistic *encoding algorithm* and the deterministic *decoding algorithm.* When E is given as input an ℓ-bit string σ and a unary security parameter k, it generates a tuple of $p(k,\ell)$ strings $(\sigma_1,\ldots,\sigma_{p(k,\ell)})$. On input a unary integer and a tuple of binary strings, D produces either a binary string or the special symbol $\perp$. Additionally, E and D satisfy the following property: For all $\sigma \in \{0,1\}^*$ and for all integers k,
$$Pr[D(1^k, E(1^k, \sigma)) = \sigma] = 1$$
- P and V are interactive Turing machines (ITMs), the latter consisting of two separate algorithms, a challenge algorithm C and an acceptance predicate A. In an execution of (P, V), both machines receive two common inputs: a security parameter 1^k, and a g-gate

predicate Q. P also receives as private input, the tuple of strings $(\sigma_1, \ldots, \sigma_{p(k,\ell)})$ — a random output of $E(1^k, \sigma)$, for some string σ. V receives no additional private input. The joint computation starts with P internally computing strings $\tau_1, \ldots, \tau_q(k, g)$, and involves the following two rounds:

1. V selects a random string, β, uniformly at random and sends it to P.
2. P runs $C(\beta)$ to compute two sets of challenge indices $I_\beta \subset \{1, \ldots, p(k, \ell)\}$ and $J_\beta \subset \{1, \ldots, q(k, g)\}$. Then, P sends V strings σ_i and τ_j for each $i \in I$ and $j \in J$.

Afterwards, V runs $A(1^k, Q, \beta, \{\sigma_i : i \in I_\beta\}, \{\tau_j : j \in J_\beta\})$ to determine whether to accept or reject.

DEFINITION 3 (SDT SYSTEM). *We say that a SDT system (E, D, P, V) with positive polynomials p and q is* secure *if it satisfies the following three properties:*

Completeness: *for all g-gate predicates Q, σ such that $Q(\sigma) = 1$, and integers k, the probability that V accepts in any execution with public inputs Q and 1^k and private prover input $E(1^k, \sigma)$ is 1.*

Soundness: *for all g-gate predicates Q, ℓ, sufficiently large k, and strings $\sigma'_1, \ldots, \sigma'_{p(k,\ell)}$, $\tau_1, \ldots, \tau_{q(k,g)}$ such that either (1) $D(\sigma'_1, \ldots, \sigma'_{p(k,\ell)}) = \perp$ or (2) $Q(D(\sigma'_1, \ldots, \sigma'_{p(k,\ell)})) = 0$,*

$$Pr_\beta[A(1^k, Q, \beta, \{\sigma_i : i \in I_\beta\}, \{\tau_j : j \in J_\beta\}) = accept] \leq 2^{-k}$$

Zero-Knowledge: *There exists a probabilistic, polynomial-time algorithm S such that for all g-gate predicates Q, $\sigma \in \{0,1\}^\ell$ such that $Q(\sigma) = 1$, strings $\beta \in \{0,1\}^*$, and integers k, the output distribution of $S(1^k, Q, \beta)$ is equal to the distribution obtained by:*

1. *generating $(\sigma_1, \ldots, \sigma_{p(k,\ell)})$ via a random execution of $E(1^k, \sigma)$,*
2. *generating $(\tau_1, \ldots, \tau_{q(k,g)})$ via a random execution of $P(1^k, Q, \sigma_1, \ldots, \sigma_{p(k,\ell)})$,*
3. *outputting $(\beta, \{\sigma_i : i \in I_\beta\}, \{\tau_j : j \in J_\beta\})$.*

LEMMA 1 ([3]). *Secure SDT protocols exist.*

Non-interactive zero knowledge

Non-interactive zero-knowledge (NIZK) proofs [6] have become essential components of modern cryptographic protocols. Much like their interactive counterparts, NIZK proofs allow a prover to prove any NP-statement in a manner that (a) allows all true theorems to be proven (completeness), (b) prevents false theorems from being proven except with negligible probability (soundness), and (c) reveals nothing other than the verity of the theorem (zero-knowledge). Unlike traditional ZK proofs, however, NIZK proofs do not require any interaction, but instead require there to be a uniform random string available to both the prover and verifier. An NIZK system consists of two algorithms: (1) an efficient prover algorithm P_N, which accepts a theorem, a witness, and a common random string, and produces a proof string, and (2) a deterministic verifier algorithm V_N which accepts a theorem, a proof string, and a common random string, and outputs whether or not the proof string is acceptable. Giving control over the choice of the CRS to either the prover or verifier destroys the soundness or the zero-knowledgeness (respectively) of the proof system. Indeed, the zero-knowledge property of an NIZK system is usually proven by constructing a simulator which chooses the CRS so that, for any (possibly false) statement, it can construct a proof that the verifier accepts.

The GMW protocol in a nutshell

The GMW protocol consists of two parts. The first part is to define and construct an SFE protocol for "honest-but-curious" players. (Such players — whether good or bad — send every message according to their prescribed instructions. However, those who are bad may share information with one another and try to deduce as much as they can about good players' inputs and outputs.) The second part is a general compilation technique which transforms the honest-but-curious SFE protocol into an SFE that also works in the presence of malicious players.

The honest-but-curious protocol. The honest-but-curious protocol consists of a protocol in which parties communicate via private channels.

The key property of this protocol is that it ensures privacy: that is, even if $n - 1$ curious players share all the information that they learn during the execution, they cannot deduce anything at all about the input or output of the remaining party, other than what is implicit from their own inputs and outputs. Correctness and complete fairness are easily shown, given the honesty of all parties.

The compiler. In the compiled protocol, players instead communicate via broadcast, encrypting all their private messages to the proper recipients. Additionally, all players prove publicly and in zero-knowledge that the ciphertexts they send are encryptions of messages that are properly computed according to the honest-but-curious protocol. This forces even malicious players to behave honestly, lest they be detected.

More specifically, the compiled protocol works in the following way:

1. In a pre-processing stage, the players interact so that each player i produces a secret value and three public values. The secret value is a decryption key sk_i of a public key cryptosystem. The first public value is the corresponding encryption key pk_i, the second is an encryption (relative to pk_i) of a random initial state for P_i (the ITM of player i in the honest-but-curious protocol), and the third is an encryption of i's input (again relative to pk_i). The difficulty in this step is to guarantee the mutual independence of the secret random tapes (which are part of the players' initial states) and of the players' inputs, given that bad players may try to correlate their inputs to those of good players.

2. The players emulate the honest-but-curious protocol: every time a player i is to send a message m to player j, player i instead broadcasts an encryption of m under pk_j, then provides a zero-knowledge proof that the ciphertext is constructed properly. The proof is relative to the state of the honest-but-curious program P_i at that point, which in turn is based on the initial, encrypted state and input, and all the encrypted messages intended for P_i thus far. (Note that the statement of correctness is efficiently verifiable given the

witness sk_i, so the statement can be proved in zero-knowledge.)

These proofs are interactive, and are actually performed $n-1$ times during which the role of the verifier cycles through the $n-1$ other players.

3. In the final stage, each player is left with a single final message for each player (including himself). These messages are sent exactly as in Step 2, and each player computes his final output based entirely upon these messages. (We note that until the final messages are received, no player has any knowledge about his final output.)

3.2 Our Construction

The GMW protocol presented in the previous section ignores fairness. In particular, the last player who is scheduled to broadcast her shares during the final stage might simply refuse to do so, and therefore deny output to all of the other players (after having seen her own outputs). Even using a simultaneous broadcast channel does not solve the problem—one malicious party who encrypts and then broadcasts junk creates a situation in which only she receives her output and other players do not. In order to guarantee fairness, a tighter coupling of the encrypt-broadcast-prove paradigm must be employed. All parties must be convinced that they are going to receive legitimate shares before any party is allowed to exchange a single share.

Our approach embraces this requirement through a set of complicated envelope manipulations during the final stage of GMW. A high-level description of the protocol follows:

- **Partially Run GMW.** Run the original GMW protocol as before until the final stage of the compiled protocol. At this point, each player i has privately computed the share y_{ij} for every player j.

  (*Comment: Notice that up to now, the protocol is completely fair in the sense that no coalition of bad players has any knowledge about the input or output of any honest player.*)

- **Create Envelopes.** The players generate a common random string, CRS, of suitable length.

  In order to generate such a string, the players engage in the following $2n$ round process: Each of the players from 1 to n selects a random string r_i, and announces a (probabilistic) encryption, $E_{pk_i}(r_i)$. Then, in reverse order, starting with player n, the players reveal r_i and the coin-tosses c_i used to encrypt it.

  Every player verifies the encryption $E_{PK_i}(r_i)$ was indeed the encryption of r_i using the random coins c_i. If all agree, then all players set $CRS = \bigoplus_1^n r_i$.

  Using an NIZK proof system (P_N, V_N) and the common random string CRS selected in the previous step, each player i produces a sequence of NIZK proofs, $\{\pi_{ij}\}$, one for each share y_{ij}, for the following statement:

  > The share y_{ij} is precisely what the honest-but-curious machine P_i would have computed on the initial state and input whose encryptions were sent earlier, and all of the message traffic that has been subsequently received by P_i. (The NP-witness for this statement is sk_i, player i's secret decryption key.)

  Let (E, D, P, V) be a SDT-system as described in Section 3.1 and let σ^{ij} be the concatenation of the share y_{ij} and its proof π_{ij} of correctness. Let $m = p(k, |\sigma^{ij}|)$ where $p()$ is the expansion rate of the encoding algorithm E.

  Each player i runs the encoding algorithm E on each string σ^{ij} in order to generate a sequence of sets, $\{S^{ij}\}$, one for each player, where each $S^{ij} = \{\sigma_1^{ij}, \ldots, \sigma_m^{ij}\}$.

  Let $Q_{ij}(\pi)$ be the predicate which is true if the NIZK verifier V_N accepts proof string π relative to CRS, and let g_{ij} be the number of gates in Q_{ij}.

  Player i now runs the SDT prover P on 1^k, S^{ij} and Q_{ij} in order to generate an internal set of strings, $T^{ij} = \{\tau_1^{ij}, \ldots, \tau_{q(k,g_{ij})}^{ij}\}$.

  Each player generates n piles of closed but publicly viewable envelopes. The jth pile of envelopes generated by player i, denoted e_{ij}, consists of one envelope for each string in S^{ij}.

  (*Comment: Again, the protocol is completely fair at this point. The channel restricts any player from running off with envelopes that have been filled.*)

- **Verify Envelopes.** Players engage in a coin-flipping protocol (as described above) in order to generate several random strings, which we denote β_{ij}, where $1 \leq i, j \leq n$ index all pairs of players.

  Each player i runs the challenge algorithm, $C(\beta_{ij})$, for all $1 \leq j \leq n$ in order to generate two sets of challenge indices, $I_{\beta_{ij}}$ and $J_{\beta_{ij}}$.

  At this point, each player i opens the envelopes indexed by $I_{\beta_{ij}}$ and broadcasts those strings from the set T^{ij} which are indexed by $J_{\beta_{ij}}$ for $1 \leq j \leq n$.

  Finally, each player j, for each i, runs the SDT acceptance predicate $A(1^k, Q_{ij}, \beta_{ij}, \{\sigma_k^{ij} : k \in I_{\beta_{ij}}\}, \{\tau_k^{ij} : k \in J_{\beta_{ij}}\})$.

  If any acceptance predicate returns false, then the protocol is aborted, the player who created the bad envelopes is identified as the aborter, and all of the envelopes are destroyed.

  (*Comment: Since the SDT protocol is perfect zero-knowledge, no information about the strings in the envelopes has been released. Thus, the protocol at this point remains completely fair.*)

- **Exchange Envelopes.** Otherwise, each player j, for each i, collects the pile of envelopes e_{ij} addressed to him. The channel forbids any interference or interruption of the envelope exchange. With these envelopes, player j can reconstruct y_j as in the GMW protocol.

  (*Comment: The envelopes are exchanged only after all parties have verified the envelopes that are "on the table." This feature ensures our complete fairness.*)

THEOREM 1. *If trapdoor permutations and ideal envelopes exist, the protocol described above is an aborter-identifying completely fair SFE for general functions.*

We defer the proof of Theorem 1 to the final version.

4. APPLICATIONS TO GAME THEORY

4.1 Basic Game Theory Concepts

In this work, we focus on non-cooperative games written in normal form[5]. In a game $G = (I, (A_i)_{i\in I}, (u_i)_{i\in I})$, the set of players is I, and each player $i \in I$ has a finite, public set of *actions*, A_i, and a public *utility function*, u_i. Each u_i maps an n-tuple of actions $(a_1, \ldots, a_n) \in A_1 \times \cdots \times A_n$ to a real number, referred to as i's *payoff* or *utility*. The game is played *in a single stage:* every player independently chooses an action from his action set, and all players receive their corresponding payoffs.

For a player i, an *individual strategy* is a probability distribution over A_i, that is, a way for i to choose which action to play. For a vector of strategies σ and a set of players C, we denote by σ_C the vector of strategies for players in C, and by σ_{-C} the vector of strategies for players outside of C. For ease of notation, we will often apply utility functions to *strategy* vectors (rather than action vectors). When doing so, we refer to the *expected* utility, taken over the distribution of actions within each strategy.

A *Nash equilibrium* [24] is a vector σ of individual strategies, one for each player, such that no single player can improve his expected payoff by switching to a different strategy. Formally, σ is a Nash equilibrium if for all i and all σ'_i, $u_i(\sigma'_i, \sigma_{-\{i\}}) \leq u_i(\sigma)$. It is well known that every game has a Nash equilibrium [24]. More generally, a *correlated equilibrium* [1] is a probability distribution, E, over $A_1 \times \cdots \times A_n$ satisfying the following property: if an n-tuple is chosen according to E and its ith component is privately given to player i as his "recommended" action, then no single player can improve his expected payoff by disobeying his recommendation.

Correlated equilibria are more advantageous than their Nash counterparts on two accounts. First, being more general, they may (as for all games that are not *strategically zero sum* [23]) yield strictly better payoffs to each player. Second, they can always be found efficiently (i.e., in time polynomial in the binary representation of the game) via linear programming — whereas the complexity of computing Nash equilibria for general games is still unknown. However, while a Nash equilibrium can be implemented by individual players, achieving a general correlated equilibrium requires the help of an external trusted party, T.[6] A game $G = (I, (A_i), (u_i))$ *extended* by an external trusted party T (implementing a correlated equilibrium E) is denoted by $G^T = (I, (A_i^T), (u_i^T))$ and proceeds as follows. First, T randomly selects an n-tuple of actions, $a_1, \ldots, a_n$, according to E and privately sends the *recommendation* a_i to every player i. Subsequently, the players choose which actions to play in the original game G —a good player g choosing his recommended a_g— and receive the payoffs of G.

In 1988, Bárány [2] envisioned replacing external trusted parties by *cheap talk*: that is, by a *communication protocol* in which the players exchange messages with each other over a suitable communication channel. Each cheap-talk protocol essentially transforms an ordinary game G into a corresponding *extended game* G' in which the players (1) execute the prescribed communication protocol, and then (2) play actions in the original game G. Bárány showed that there exists a chep-talk protocol such that, for any game G and any correlated equilibrium E in G, the corresponding extended game G' has strategies that form a Nash equilibrium and have the same payoffs as E.

Unfortunately, however, Bárány's and all subsequent cheap-talk protocols are extremely vulnerable to even 2-player *coalitions* of deviating players. That is, two colluding players (e.g., in Bárány's case players 1 and 3, and in Ben-Porath's case players 1 and 2) may *always and undetectably* force the payoffs received by all players to be any ones they want. This is a very serious drawback, and we shall eliminate it via our completely fair SFE without honest majority.

4.2 Coalitions and Coalition Safety

Modeling coalitions involves many subtleties, and the game theory literature has explored several notions —e.g., [26, 22]. In this work we start with a simple, but most adversarial model: the monolithic coalition.

A *monolithic coalition* is a subset of players who totally trust one another, and thus perfectly coordinate their actions. Though such a total, mutual trust may be unrealistic, monolithic coalitions have the best chance of controlling the payoffs of the game, and thus is crucial for good players to achieve protection against them. (In the final version of this paper, we shall refine our results relative to more sophisticated types of coalitions —e.g., the ones of Moreno and Wooders [22].)

Monolithic Coalitions and Trusted Parties

If T is an external trusted party implementing a correlated equilibrium E, a monolithic coalition in $G^T = (I, (A_i^T), (u_i^T))$ is a single algorithm, $\mathscr{C}$, controlling a subset of (bad) players, B. The interaction among $\mathscr{C}$, T, and the good players, with security parameter 1^k, is as follows. First, T chooses the recommended actions $a_1, \ldots, a_n$. Second, $\mathscr{C}$ chooses a subset $R_B \subseteq B$ representing the bad players whose recommended actions it wishes to receive.[7] Third, $\mathscr{C}$ receives a_j for each bad player $j \in R_B$. Fourth, $\mathscr{C}$ select an action a'_b for each bad player b in B. Finally, all players choose their actions in G —the strategy of every good player g, σ_g, is choosing a_g, while every bad player $b \in B$ chooses a'_b— and all receive G's payoffs. In line with game-theoretic tradition, such a coalition $\mathscr{C}$ corresponds to a vector of individual strategies, $\{\sigma_b^{\mathscr{C}} : b \in B\}$, compactly denoted by $\sigma_B^{\mathscr{C}}$. (To ensure the monolithic nature of the coalition, each strategy $\sigma_b^{\mathscr{C}}$ includes private, "extra" communication with other bad players.[8]) Accordingly, the expected payoffs of G^T with the above $\mathscr{C}$ will be denoted by $u_k^T(\sigma_B^{\mathscr{C}}, \sigma_{-B})$.

[5]Note that extensive-form games, which are played out in several stages (with potentially imperfect information), can be written in normal form.

[6]Often such a party is referred to as a "correlating device." But this psychologically benign term seems to hide the extraordinary assumption that ALL players must trust the SAME device to work properly. In general, a correlating device needs to be a complex special-purpose computer, and can be manufactured so as to hide a malicious program "favoring" some of the players!

[7]In cryptography and non-cooperative game theory there are no mechanisms to *force* a player to do anything. As a consequence, some colluders may refuse to look at their recommendations, if they so want!

[8]In particular, the bad players may share their recommendations, or select a common random string if they wish to coordinate their random choices.

(The security parameter 1^k will be necessary, in the cheap-talk setting, for upperbounding the computational resources of an efficient coalition, and thus becomes necessary in the trusted-party setting, too, in order to express that "whatever a coalition can do in one setting, it can also do —with the same computational resources!— in the other. Notice, however, that the security parameter does not constrains the good players nor T, even if we require that they too be efficient. In fact, all is required from a good player is to read his recommended action, and in a finite game all actions are described by strings of constant size. As for T, it is assumed that he has the correlated equilibrium E as an additional input, and thus can perform its required selection in time polynomial in the length of its total input, $|E| + k$.)

Monolithic Coalitions and Cheap Talk

When trusted parties are replaced by n-party protocols, a monolithic coalition is the already discussed *adversary*, that is, a single ITM that totally controls a set B of bad players.

Let $G = (I, (A_i), (u_i))$ be a game, E a correlated equilibrium for G, and $CT = (s_1, \ldots, s_n)$ a protocol. We say that CT is a *cheap-talk* protocol implementing E if, a random execution with an adversary and common input 1^k, produces outputs $a_1, \ldots, a_n$ distributed according to E.

(Notice that our definition of cheap talk is much stronger than envisaged so far. Indeed, up to now, all that was required was "to withstand a single-player adversary.")

We denote by $G^{CT} = (I, (A_i^{CT}), (u_i^{CT}))$ the extended game in which the players first execute CT (with some common security parameter 1^k) and then choose an action to play in G and receive G's payoffs.

Playing G^{CT} with C controlling the set of bad players B consists of executing CT with adversary C, and then having C use its final view to choose the actions of the bad players. To preserve again game-theoretic tradition, our C corresponds to a set of individual bad strategies, $\{s_b^C : b \in B\}$, compactly denoted by s_B^C. (Again, these strategy include private, "extra-protocol" communication among the bad players to achieve the desired coordination.) Thus, the expected payoffs of G^{CT} are denoted by $u_k^{CT}(s_B^C, s_{-B})$.

We are now ready to define *coalition-safe* cheap talk. Informally, this refers to a cheap-talk protocol for which "any advantage that an efficient coalition may obtain can also be obtained by the same coalition when an external trusted party is used instead of the cheap talk."

Definition 4 (Coalition-Safety). *Let E be a correlated equilibrium for a game $G = (I, (A_i), (u_i))$, and let T be an external trusted party and CT a cheap-talk protocol both implementing E. We say that CT is coalition-safe if, for any set of players B, and any monolithic, efficient, cheap-talk strategies s_B^C there exist monolithic, trusted-party, strategies $\sigma_B^{\mathscr{C}}$ such that the payoffs*

$$\{u_k^{CT}(s_B^C, s_{-B})\}_{k \in N} \quad \textit{and} \quad \{u_k^T(\sigma_B^{\mathscr{C}}, \sigma_{-B})\}_{k \in N}$$

are computationally indistinguishable.

Note that the above indistinguishability is between *vectors*, and thus much stronger than simple indistinguishability between individual components. In particular, therefore, coalition safety not only prevents all bad players from improving all their payoffs, but also prevents a single bad player from improving his own payoff at the expense of another bad (or good!) player.

4.3 Achieving Coalition-Safety

Theorem 2. *Given completely fair, aborter-identifying, SFE protocols for any efficient function f, then, for any game G and any correlated equilibrium E for G, there exists an efficient, coalition-safe, cheap-talk protocol $CT = (s_1, \ldots, s_n)$ such that the strategies in G^{CT} consisting of "following CT and then playing in G the finally computed recommendations" form a computational Nash equilibrium[9] with payoffs indistinguishable from those of E.*

Proof Sketch: (Very informal)

It is immediately seen that there exists an efficient, function f which, on input a random k^d-bit string ρ (where $d > 0$ is a constant), approximates E to within an exponentially vanishing (in k) probability of error.[10]

Since f is probabilistic and we defined SFE only for deterministic functions, we now modify f (in a standard way). First, define $f_{1,j}$ to be the deterministic j-input function which computes the exclusive-or of all its inputs, and then evaluates f on the resulting string. Consider now the following cheap-talk protocol $CT[1, n]$ in which (1) every player i privately chooses a k^d-bit long random string ρ_i, and then (2) all players run a completely fair, aborter-identifying, SFE protocol for $f_{1,n}$ on their inputs ρ_i.

Suppose that, in the cheap talk game $G^{CT[1,n]}$, a coalition C controls a *proper* subset B of the players (otherwise, the Theorem 2 follows trivially).

First let us analyze the distribution of the payoffs of the extended game $G^{CT[1,n]}$ when C never induces an abort. Because B is a proper subset of the players, there is at least one honest player, and thus the exclusive-or of the players' inputs is a truly random k^d-bit string. Thus, because $CT[1, n]$ is a completely fair SFE for $f_{1,n}$, no matter what C does, all parties' recommended outputs, $a_1, \ldots, a_n$, are distributed according to E. At this point, without loss of generality, C can perform some additional computation, C', to map the bad-players' recommendations $\{a_b : b \in B\}$ to new actions $\{a'_b : b \in B\}$. Then all players play their actions in G and receive G's payoffs. We now argue that such payoffs can essentially also be obtained by a monolithic coalition $\mathscr{C}$ in G^T.

In essence, $\mathscr{C}$ specifies the recommendation-receiving set B' to be equal to B. Thus, after the trusted party T selects a random n-tuple of actions $a_1, \ldots, a_n$ according to E, $\mathscr{C}$ receives all recommendations $\{a_b : b \in B\}$. At this point, $\mathscr{C}$ runs the same algorithm C' of C to transform the received recommendations into the actions that the players in B actually play in G. Every good player g plays instead his originally received a_g, and all players receive G's payoffs.

Because the actions ultimately played in G, in the above cheap-talk and the trusted-party scenarios, are computationally indistinguishable[11], so are their relative payoffs, be-

[9] In a computational Nash equilibrium, defined by Dodis, Halevi and Rabin [11], no single player has an *efficient* deviating strategy that yields a *non-negligibly* greater payoff that the equilibrium strategy.

[10] I.e., $\sum_{A \in A_1 \times \cdots \times A_n} |p_E(A) - p_f(A)| < 2^{-k}$, where $p_E(A)$ is the probability of A according to E, and $p_f(A)$, the probability that, on inputs a random k^d-bit string f outputs A.

[11] Actually, because we are dealing with finite functions here, the two distributions of actions are even "statistically indistinguishable."

cause the latter —being finite functions— are certainly efficiently computable.

Assume now that C induces an abort during $CT[1,n]$. Because $CT[1,n]$ consists of a completely fair SFE for $f_{1,n}$, if an abort occurs, then no player, bad or good, learns any recommendation. Nonetheless, the strategies of G^{CT} must specify which actions to play in G in every situation, and thus in this one too. Fortunately, because our SFE also is aborter-identifying, the players at least learn who caused the abort. Without loss of generality, let it be (bad) player n. In prior cheap-talks such as Bárány's, the good players would now "punish" player n by choosing actions that minimize n's payoff (which immediately guarantees the Nash equilibrium of the prescribed strategies in the extended game G'). Such an approach works relative to a single deviating strategy. In our context, however, there are coalitions to consider: the good players, while aware that n is bad, do not know who else might be bad, and thus cannot adopt a good "punishment strategy." Assume, for example, that they choose to play actions $\bar{a}_1, \ldots, \bar{a}_{n-1}$, as in Bárány. Then, if, without loss of generality, player $n-1$ was also in C, he might play a different action a'_{n-1}, while player n plays an action a'_n, so that both colluding players may receive payoffs higher than in E. Notice that the so generated payoffs do not have any "counterpart" in the trusted-party setting, because there good players always play the recommendation received from T, without knowing who might be bad. Thus, this and all other punishing procedures may both backfire and destroy any hope of "payoff indistinguishability from G^T."

Our solution, instead, is to run a new cheap talk protocol for $n-1$ players, $CT[1,(n-1)]$, in which every player $i \in [1,n-1]$ selects a random string ρ_i, and then all players run a completely fair, aborter-identifying, SFE which computes the function $f_{1,n-1}$ to obtain an n-tuple of actions, $a_1, \ldots, a_n$, distributed according to E, and then returns a_i to player i for $i = 1, \ldots, n-1$. (In a sense, $CT[1,n-1]$ implicitly computes a_n and then discards it.)

Once again, there are two cases to consider. If there are no aborts during this second execution, then every good player g is instructed to play a_g in G. (The bad players may instead run some other algorithm, C'', to turn their computed recommendations a_b, for each bad player in $B - \{n\}$, into actions $\{a'_b : b \in B\}$ to play in G.) It is easy to see that "the payoffs so generated by C in $G^{CT[1,n-1]}$ can be generated in an indistinguishable way by a corresponding coalition $\mathscr{C}$ in G^T." If there is an abort, then the remaining players run another cheap talk protocol, $CT[1,n-2]$, and so on. Eventually, all members of C who might trigger an abort will be removed, and the remaining honest players (who are never instructed to trigger an abort) will complete a cheap-talk protocol and play their output recommendations in G. We refer to this informally described cheap-talk protocol (i.e., $CT[1,n]$, possibly followed by $CT[1,n-1]$, possibly followed by $CT[1,n-2]$, etc.) as CT. In the final paper we shall prove that CT indeed is a coalition-safe cheap-talk protocol for E and G, and that the strategies consisting of following CT and then playing in G the finally computed recommendations is a computational Nash equilibrium in G^{CT} with payoffs indistinguishable from E. ■

Let us now state an obvious consequence of Theorems 1 and 2.

COROLLARY 1. *Given trapdoor permutations and ideal envelopes, then, for any game G and any correlated equilibrium E for G, there exists an efficient, coalition-safe, cheap-talk protocol $CT = (s_1, \ldots, s_n)$ such that the strategies in G^{CT} consisting of following s_i and then playing CT's output recommendation a_i form a computational Nash equilibrium with payoffs indistinguishable from those of E.*

5. INEFFICIENCY OF PRIOR PROTOCOLS

Let us now prove that ours is the first, general cheap-talk protocol to be efficient in the description of the correlated equilibrium.

PROPOSITION 1. *The cheap-talk protocols of Bárány [2], Ben-Porath [5], Gerardi [15], Dodis, Halevi, Rabin [11] and Teague [27] may require communication that is exponential in the size of the binary description of the desired correlated equilibrium.*

PROOF. First, consider the following family of 2-player games, parameterized by any perfect cube $\alpha > 8$, which we call "high-stakes chicken." (Typically, α is large.) Each player has two actions: C, for "chicken out," and D, for "dare." The payoffs are as follows, where $\delta = \alpha - 2\alpha^{2/3}$ and $\gamma = \alpha + \alpha^{2/3}$:

	C	D
C	α, α	δ, γ
D	γ, δ	$0, 0$

A straightforward analysis shows that this game admits a correlated equilibrium with probability mass $q = \alpha^{-1/3}$ on each of (C, D) and (D, C), and the rest on (C, C). The expected payoff of this equilibrium is $\alpha - \alpha^{1/3}$ for both players, which exceeds those of the best mixed-strategy Nash equilibrium by at least $\alpha^{1/3}$ —so the correlated equilibrium provides significant benefit. It is trivial to extend this game to more players (who unconditionally receive 0 payoffs) without changing the equilibria.

All of the cited works represent the correlated equilibrium distribution by rational probabilities $p_1, \ldots, p_j$ written as fractions. The central idea for all of these protocols (except Teague's [27]) is to generate a list of size L, the least common denominator of the fractions. The problem with this list-based approach is that L may be very large with respect to the size of the representation of the p_is.

In our example, $L = \alpha^{1/3}$. On the other hand, the game and its correlated equilibrium can be written in $O(\log \alpha)$ bits because all of the payoffs are positive integral values that are bounded by a constant multiple of α. Therefore, in order to run any of the cited protocols, the communication complexity is $\Omega(L) = \Omega(\alpha^{1/3})$, which is *exponential* in the size of the game and correlated equilibrium.

Although Teague's protocol does not explicitly create a list of size L, the communication complexity is related to the inverse of the smallest non-zero probability in the correlated equilibrium. In our example, this value is still $\Omega(\alpha^{1/3})$. ■

Acknowledgments

We would like to thank Alon Rosen for suggesting the use of non-interactive zero knowledge for conceptually simplifying our original construction.

6. REFERENCES

[1] R. Aumann. Subjectivity and correlation in randomized strategies. *J. Math. Econ.*, 1:67–96, 1974.

[2] I. Bárány. Fair distribution protocols or how the players replace fortune. *Mathematics of Operation Research*, 17:327–341, May 1992.

[3] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Hastad, J. K. S. Micali, and P. Rogaway. Everything provable is provable in zero-knowledge. In *Proc. CRYPTO 88*, pages 37–56. Springer Verlag, 1988.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for fault-tolerant distributed computing. In *Proc.* 20*th STOC*, pages 1–10. ACM, 1988.

[5] E. Ben-Porath. Correlation without mediation: Expanding the set of equilibria outcomes by "cheap" pre-play procedures. *Journal of Economic Theory*, 80:108–122, 1998.

[6] M. Blum, A. D. Santis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *SIAM J. Computing*, 20(6):1084–1118, 1991.

[7] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd FOCS*, pages 136–147. IEEE Computer Society, 2001.

[8] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *Proc. 28th annual ACM Symposium on Theory of Computing*, pages 639–648. ACM Press, 1996.

[9] D. Chaum, C. Crepeau, and I. Damgård. Multi-party unconditionally secure protocols. In *Proc.* 20*th STOC*, Chicago, 1988. ACM.

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

[11] Y. Dodis, S. Halevi, and T. Rabin. A cryptographic solution to a game theoretic problem. In *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *LNCS*, pages 112–130. Springer-Verlag, 2000.

[12] Y. Dodis and S. Micali. Parallel reducibility for information-theoretically secure computation. In *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *LNCS*, pages 74–92. Springer-Verlag, 2000.

[13] P. Feldman and S. Micali. Byzantine agreement in constant expected time (and trusting no one). In *Proc. 26th FOCS*, pages 267–276. IEEE Computer Society, 1985.

[14] J. A. Garay, P. MacKenzie, and K. Yang. Efficient and secure multi-party computation with faulty majority and complete fairness. Technical Report 9, Eprint, January 2004. http://eprint.iacr.org/2004/009.

[15] D. Gerardi. Unmediated communication in games with complete and incomplete information. *Journal of Economic Theory*, to appear.

[16] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc.* 19*th STOC*, pages 218–229. ACM, 1987.

[17] S. Goldwasser and L. Levin. Fair computation of general functions in the presence of an immoral majority. In *Proc. CRYPTO 90*, pages 77–93. Springer Verlag, 1990.

[18] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Science*, 28(2), 1984.

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

[20] M. Luby, S. Micali, and C. Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin. In *FOCS*, pages 11–21, 1983.

[21] S. Micali and P. Rogaway. Secure computation. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 392–404. Springer, 1992. Lecture Notes in Computer Science No. 576.

[22] D. Moreno and J. Wooders. Coalition-proof equilibrium. *Games and Economic Behavior*, 17:80–112, 1996.

[23] H. Moulin and J. P. Vial. Strategically zero sum games. *Internat. J. Game Theory*, 7:201–221, 1978.

[24] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.

[25] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc.* 21*st STOC*, pages 73–85. ACM, 1989.

[26] I. Ray. Coalition-proof correlated equilibrium: A definition. *Games and Economic Behavior*, 17:56–79, 1996.

[27] V. Teague. Selecting correlated random actions. In *Proc. Financial Cryptography*, 2004.

[28] A. Urbano and J. E. Vila. Computational complexity and communication: Coordination in two-player games. *Econometrica*, 70(5):1893–1927, 2002.

[29] A. Yao. Theory and application of trapdoor functions. In *Proc. 23rd FOCS*, 1982.

[30] A. C.-C. Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pages 162–167. IEEE Computer Society, 1986.