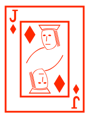
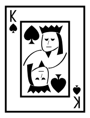
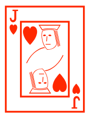
Collusion-Free Protocols

Matt Lepinksi[*]
CSAIL, MIT
lepinski@csail.mit.edu

Silvio Micali
CSAIL, MIT
silvio@csail.mit.edu

abhi shelat
CSAIL, MIT
abhi@csail.mit.edu

ABSTRACT

Secure protocols attempt to minimize the injuries to privacy and correctness inflicted by malicious participants who *collude during run-time.* They do not, however, prevent malicious parties from colluding and coordinating their actions in the first place!

Eliminating such collusion of malicious parties during the execution of a protocol is an important and exciting direction for research in Cryptography. We contribute the first general result in this direction:

(1) We provide a rigorous definition of what a *collusion-free* protocol is; and

(2) We prove that, under standard physical and computational assumptions —i.e., plain envelopes and trapdoor permutations— collusion-free protocols exist for all finite protocol tasks with publicly observable actions.

(Note that such tasks are allowed to have secret global state, and thus include Poker, Bridge, and other such games.)

Our solution is *tight* in the sense that, for a collusion-free protocol to exist, each of (a) the finiteness of the game of interest, (b) the public observability of its actions, and (c) the use of some type of physically private channel is provably essential.

Categories and Subject Descriptors: F.1.2 [Modes of Computation]: Interactive and Reactive Computation

General Terms: Security, Theory

Keywords: Secure Function Evaluation, Steganography

1. INTRODUCTION

Consider an *ideal implementation* of a multi-player game as one in which an external trusted party handles all of the crucial details of the game's execution. The goal of *secure multi-party computation* [11] (SMC, for short) is to provide a *real implementation* of the game via a communication protocol which does not rely on a trusted party. Specifically, SMC guarantees that even if a proper subset of the players collude and deviate from their prescribed protocol instructions in an arbitrary and coordinated manner during an execution, the *real implementation* unfolds with the same privacy and correctness properties as the *ideal implementation.*

SMC has been extensively investigated and improved; in particular, by (a) solely relying on private channels [2, 7] in place of computational assumptions, (b) tolerating the dynamic corruption of players [6], and (c) guaranteeing its safe use as a subroutine within arbitrary and larger SMC protocols [9, 5]. However, no prior SMC protocol could *prevent* malicious players from successfully coordinating their actions during execution, because all prior work assumes that players have access to side channels during an execution. Under this assumption, bad players can coordinate their actions in the *ideal* setting, and thus bad players retain this collusion power in the *real* setting.

In many games, however, a colluding set of just two players using side channels would be disastrous. Take poker, for example. Two players who secretly share information about their hands during the game have a significant —and illegal!— advantage. In 1985, Crépeau [8] introduced techniques for limiting the power of a coalition of bad players in Mental Poker [17], and raised the important question of whether this power could be eliminated altogether. His question has remained unanswered until now.

We provide a positive solution to his open problem by (1) formalizing what it means to make a coalition as powerless as possible in a SMC protocol, and (2) achieving such security, under standard computational and physical assumptions, not only for poker, but for any finite game with *publicly observable actions* —i.e., games in which each player's actions are publicly announced to all other players.

The Heart of the Problem: Steganography

A *coalition* of bad players arises only if its members coordinate their actions by communicating with each other (else, we would already be facing a *set* of independent bad players). Therefore, to prevent a coalition from arising, honest players in a collusion-free protocol should abort an execution in which they detect any *extra* (i.e., non protocol-specified) communication, since its very presence indicates that bad players are at work. Right away, this implies that private channels cannot be the sole means of communication during a collusion-free protocol: they would automatically make any communication among bad players undetectable. Indeed, when constructing our collusion-free protocols, we de-

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

mand that, at least during critical portions, all communication is via broadcast only. But the use of broadcasting is not sufficient. Bad players may *subliminally* use the broadcast messages within the protocol to secretly coordinate themselves. We view it as a high responsibility of a protocol designer to guarantee that this does not happen, i.e., that *the protocol itself cannot be used to provide any additional power to a coalition.*

The heart of the problem in this context becomes *steganography*, which is the ability to convey a hidden message via a public and apparently innocuous one. (For instance, a photographer in a censored country may use the photograph of a middle-aged man to secretly transmit the bit 0, if the 30th hair from the left is white, and 1 otherwise.) Steganography is a formidable problem in our context because

1. *Steganographic communication is provably impossible to detect whenever there is a minimum amount of entropy* (as shown in [4, 16, 14, 1]); and
2. *Any secure protocol must have a lot of entropy* (as shown by [12] even in the case of simple encryption).

Fortunately, we shall prove that reconciling these two truths is paradoxical, but not impossible. That is, for a broad class of games, while the presence of steganographic communication is *impossible to detect*, its presence is *possible to prevent.*

Remark. Our work shuts down a main avenue of subliminal communication: that which is introduced by the protocol itself. Other *physical* avenues for such communication, however, might still be open to adversaries who wish to coordinate themselves. (E.g., they may signal each other by precisely timing the sending of their protocol messages, by winking, by coughing, etc.) Shutting down these physical avenues as well will not be easy. Nonetheless, our results represent a significant paradigm shift concerning the responsibility for preventing cheating. In the past, colluding players could have simply used the *protocol itself.* Now they must use *external* means.

Collusion-Free Protocols

In this extended abstract, we examine collusion-free protocols in the static, stand-alone model. Informally, this means that we focus on a single execution of a game where the set of bad players is fixed in advance, only one player is active at any given time, and any player may abort the game. We do not place any restriction on the number of bad players, and thus do not consider issues of "fairness."

To define collusion-free protocols, we embrace the traditional Ideal-Real paradigm introduced in [11]. We refer to the abstract specification of a protocol as a *game G*, and then consider the ideal and real settings as two different types of implementations of G. For instance, for the first stage of poker (i.e., the deal stage), the game G specifies that each player privately learns a random and independent subset of 5 cards, subject only to the constraint that the subsets be disjoint; the *ideal* implementation consists of a trusted dealer choosing a permutation of the 52 cards, and privately handing the first five to player 1, the second five to player 2, and so on; and the *real* implementation is a protocol for dealing cards.

Ideal Implementation. Essentially, this is a trusted-party implementation of a game G, as in traditional SMC definitions, with the caveat that *all extra communication channels are removed.*

For clarity, we refer to a player in the ideal implementation as a *human* player. Such a player communicates with the trusted party in a private and specific manner. In simple terms (see Section 2 for more details), when it is his turn to play, a human player can only send an *abort* signal, or his chosen *action*; and, at the end of a player's turn, he can only receive either the signal *"game aborted"* or his own *partial information* about the game's *global state.* The trusted party privately receives a player's action, properly updates G's global state, and returns to every player his proper partial information about the new global state. This traditional mechanics capture the best possible correctness and privacy for G. Our only addition to the above mechanics is that each human player is confined to a "Faraday cage" from which he can only communicate with the trusted party. The absence of any extra, player-to-player channels captures our new desideratum that *each player must act as independently as possible.* Our modification to the ideal setting is thus extremely simple —the real difficulty will lie in *implementing* it! To this end, however, we must first understand the implications of our modification and explain what "as independently as possible" means.

Though the ideal implementation forces all communication to occur —as sketched above— via the trusted party, two bad players, i and j, still possess some *game-intrinsic* ability to communicate (and indeed secretly communicate!) with one another. In fact, by choosing an action when it is his turn, i affects the global state of G, and the proper partial information about the new state that will be privately delivered to j. Thus, though the new state is not totally known or controlled, it may still be possible for i to achieve some level of communication with j during an execution. The ideal implementation does *not* — and should not — prevent this game intrinsic communication. It *does*, however, prevent any additional communication during an execution!

Incidentally, bad players may freely talk *before* and *after* executing with the trusted party.[1] Thus, even in an ideal execution, an honest player should accept that bad players may

1. *coordinate their strategies beforehand,* in an attempt to "tilt" the game to their favor, and
2. *compare notes afterwards,* in an attempt to gain some knowledge about a honest player's adopted strategy.

Real Implementation. Essentially, this is an implementation of a game G via a protocol P, as in traditional SMC definitions, again with the caveat that *all extra communication channels are removed.*

In addition, the real implementation preserves the "reactive nature" of game playing.[2] That is, a real player i has two distinct components: a human player, $\mathscr{H}_i$, and an interactive Turing machine, T_i, specified by P. Human player $\mathscr{H}_i$

[1]No protocol can prevent what may happen outside the protocol, and the ideal implementation should not promise what cannot be delivered! Accordingly, players cannot be confined to Faraday cages for ever! However, it is the protocol designer's responsibility to ensure that bad players cannot use post-game communication to disrupt the protocol's privacy guarantee in any way.

[2]Collusion-free protocols would be trivial in a model in which players commit to their strategy for G and then run a secure function evaluation on these commitments. Besides excising the joy from game-play, this approach is infeasible because writing down complete strategies for games such as Poker requires more symbols than elementary particles in the universe. Moreover, in many natural models "players may be able to play the game but not be able to compute their own description."

acts as in the ideal implementation: he chooses actions in G and receives partial information about G's current global state. Machine T_i acts as $\mathscr{H}_i$'s *interpreter:* essentially, it transforms an action of $\mathscr{H}_i$ into a protocol message, processes "the message traffic," and presents $\mathscr{H}_i$ with proper partial information. In between $\mathscr{H}_i$'s actions, therefore, T_i is busy using P's specified channels (e.g., by sending encrypted messages, performing zero-knowledge proofs, etc.) so as to simulate, together with the other machines T_j, the correct evolution of G's global state.

As mentioned above, our only modification to this traditional mechanics is that *at any time, the only channel available to a player (whether good or bad) is the one specified by P at that time.* The absence of any extra communication channels physically matches the situation of the ideal implementation.

Collusion-free Implementation. Essentially, this is a real implementation of G that "tightly simulates" the ideal one.

Easy to describe informally, a proper formal definition of collusion-free protocols is quite elusive, *even assuming familiarity with the usual subtleties of secure computation.* Personally, we regard the very definition of collusion-freeness as a primary contribution of this paper, in part because our new conceptual tools also promise to be useful in other security settings. For instance, game theory models players as selfish but independent agents, and traditionally analyzes only one player "going bad" at time. Perhaps the notion of collusion-free security can help bridge the gap between game-theoretic models and more realistic settings with multiple-player deviations.

At the risk of stating the obvious, let us emphasize that collusion-free protocols do not (and indeed cannot) prevent players from acting maliciously. Rather, they guarantee that each malicious player, if any, *acts as independently as possible.* (In a sense, they "divide and conquer" the bad players, but do not eliminate them!)

Our Results

Our main, positive result, shows that collusion-free protocols are possible under standard computational and physical assumptions. Namely,

> **Theorem 1:** If trapdoor permutations exist, any finite, partial-information game with publicly observable actions[3] has a collusion-free protocol whose communication channels consist of broadcast and plain envelopes.

Furthermore, we elucidate why our prerequisites for collusion-free protocols are necessary by proving that *each* of (a) the use of envelopes, (b) the finiteness of the game, and (c) the observability of its actions, is an *essential* element for Theorem 1 to hold.

> **Theorem 2:** There exists a finite game with publicly observable actions that has no collusion-free protocol whose *only* communication channel is broadcast.
>
> **Theorem 3:** There exists an *non-finite* game with publicly observable actions that has no collusion-free protocol whose communication channels consist of broadcast and plain envelopes.
>
> **Theorem 4:** There exists a finite game *with private actions* that has no collusion-free protocol whose communication channels consist of broadcast and plain envelopes.

Remarks

- ▷ Using envelopes in a collusion-free protocol may appear contradictory rather than essential, because any form of physically private channels could make colluding communication absolutely undetected. However, we shall use envelopes only during the initial phase of our protocols, that is, before the game proper begins. Malicious players, therefore, cannot use these envelopes to collude in any meaningful way. (In the subsequent phase when the game properly begins, all communication is broadcast only, and our computational assumptions become crucial.)
- ▷ Theorem 1 solves Crépeau's open problem [8], because Poker is a finite game whose actions are publicly observable: the only actions are publicly announcing a subset of $\{1, \ldots, 5\}$ (representing the cards in his own hand which the player wants to replace[4]) or announcing bets, calls, and folds.
- ▷ Collusion-free security is very subtle and depends, as to be expected, on the precise details of the game's specification. To illustrate, let Poker$'$ be the following variant of Poker: when a player chooses which of the $32 (=2^5)$ subsets of his cards he wants to replace, all other players learn only the cardinality of the chosen subset (i.e., an integer between 0 and 5). While both games have implementations secure against monolithic adversaries, Poker has collusion-free implementations, but Poker$'$ has none.[5]

High-Level View of Our Solution

Our collusion-free protocol for a game G consists of two, distinct subprotocols, P_1 and P_2, which are executed sequentially. Every player runs P_1 with an empty private input, and then P_2 with a private input consisting of his own history from P_1's execution.

The two subprotocols play dramatically different roles in our solution. To begin, P_1 is heavily probabilistic, while P_2 is purely deterministic. Additionally, P_1 uses broadcast and envelopes as communication channels, while P_2 solely uses broadcast. Finally, each subprotocol satisfies its own crucial property. The first one satisfies *game independence:* throughout P_1's execution, no portion of G is actually played. The second one satisfies *verifiable uniqueness:* all honest players can verify that the only source of non-determinism for a player in P_2 is the choice of actions in G made by his corresponding "human player." A bit more precisely, fix any player i and any sequence of actions for $\mathscr{H}_i$. Then, whenever T_i is about to broadcast a message in P_2, though what he is going to say is unpredictable, there is a

[3]Recall (see Section 2 anyway) that a game with partial information is *finite* if it has a finite number of players and finitely many stages, each specified by a finite stage function. Such a game has publicly observable actions if any action a player takes becomes immediately known to all players (unlike the global state and the players' private information).

[4]This is a bit open to interpretation, because Poker and its variants are traditionally described in a hybrid "physical-deck implementation" instead of an ideal or real one. However, in all such hybrid implementations, a player must keep his cards visible at all time. Therefore, everyone can see that a player is choosing —say— the action of replacing his first and third card with the next two cards in the deck.

[5]Since Poker$'$ is not a game with publicly observable actions, Theorem 1 does not apply. To be sure, the fact that Poker$'$ has no collusion-free implementations can be proven using the same techniques as in our proof of Theorem 4. Indeed, at the price of a few additional complications, we could have chosen Poker$'$ as the example game necessary to establish Theorem 4.

single message he can broadcast without causing all honest players to abort.

As already mentioned, collusion-free protocols for nontrivial games must overcome the paradox that every secure protocol must be probabilistic, yet probabilism introduces steganography. However, our two-subprotocol structure carefully avoids any contradiction. Since P_1 is probabilistic, our resulting collusion-free protocol is also probabilistic. However, thanks to game independence, the potential for steganography in P_1 is useless because no player has yet received any partial information about the initial global state of G (nor selected any action in G). In other words, whatever information bad players may steganographically exchange in P_1, they also may exchange —and are entitled to exchange— in the ideal implementation of G before the execution with the trusted party begins! But if steganography is useless in P_1, it is impossible in P_2! Thanks to verifiable uniqueness, P_2 eliminates any possible choice due to the protocol proper, and therefore removes all steganography except for that already intrinsic to G.

Having clarified the logical structure of our solution, let us now give some idea of how P_1 and P_2 actually work. In essence, P_2 replaces the probabilistic player i of a traditionally secure protocol for G by a deterministic one that utilizes randomness that is properly selected and fixed in P_1, before the game begins. The honest players of P_2, however, should be able to verify that this is indeed the case. We thus demand that player i provide a zero-knowledge proof that he is properly executing P_2. Unfortunately, this would result in a vicious circle, since a ZK proof —being a secure protocol— would itself require randomness. To break this circularity, we use so called *unique zero knowledge* proofs (uniZK for short) [6] rather than traditional, ZK ones. A uniZK system for a NP-language L satisfies the completeness, soundness and zero-knowledgeness properties of a traditional ZK system, but differs in the following two ways. First, uniZK works in a public-key setting. (Namely, the prover has a public key, UPK, and a matching secret key, USK, and uses the latter, along with a proper witness, to create a proof of membership in L. Conversely, the verifier uses PK in order to check such a proof.) Second, whenever there is a unique witness for $x \in L$, there is exactly one proof acceptable by the uniZK verifier! (Notice that our construction will only need to prove theorems for "unique-witness languages.") To enable our use of uniZK proofs in P_2, we also use P_1 for generating public and secret uniZK keys for all players, because such key generation requires randomness.

One last complication arises. If a bad player i shares knowledge about his secret uniZK key with a bad player j, then any uniZK proof that player i generates is no longer zero-knowledge, and can actually be used to subliminally convey information. For instance, by giving a uniZK proof in P_2 that "there exists a string w satisfying a polynomial-time predicate Q", player i precisely reveals w to j, and does so without being detected by any honest player! It is thus a requirement in P_2 that player i be the *only* player to know his own secret uniZK key. This requirement prohibits us from instructing player i to generate his matching uniZK keys, USK_i and UPK_i, on his own. Else, a malicious i could inform his accomplice j, before P_1 begins, that $(\text{USK}_i, \text{UPK}_i)$ will be the uniZK keys that he will generate during P_1. Nor can the keys be jointly generated via a broadcast-only protocol (e.g., via a proper secure function evaluation). This would be another vicious circle, because secure, broadcast-only protocols rely on public-key encryption. Once again, i can share with j the secret decryption key he plans to use in P_1, and then j could simply compute the same USK_i that i computes in P_1 by reading all the broadcasted, encrypted message "addressed to" i.

It is precisely in order to break this second circle that our protocol makes use of physical envelopes. Namely, we ensure that USK_i —as well as any other secret of player i— depends on unpredictable messages that the other players deliver to i in sealed envelopes as the last step of P_1. Thus, at that moment, because there is at least one good player that delivers an unpredictable string σ to i in a sealed envelope, no other bad player j may know what σ can be. Moreover, because no extra channels channels are available, i cannot inform j about σ, and thus only player i will be able to compute USK_i. (Notice that i cannot use such envelopes to slip σ to j. This is so because each player a is instructed to put his string σ_{ab} for player b into a sealed envelope E_{ab} and "put it on the table" before any envelope is actually delivered.) To maintain this unique-knowledge situation, it must also be impossible for i to steganographically communicate USK_i to j during P_2. But in P_2, each message is broadcast and verifiably unique. Thus, the only hope for i to subliminally inform j about USK_i rests in the game-intrinsic entropy of G. Here we use the fact that G is a finite game, and thus the total entropy available to a player is constant and known. Consequently, only a constant number of bits about USK_i can be communicated by i to j using the game. With a sufficiently large security parameter, and properly choosing our crypto primitives, such a small amount of information about USK_i is provably useless for any adversary to violate our collusion-free definition.

Organization The focus of our paper is the definition of collusion-free protocols in Section 2. In Section 3 we present a construction for them, and in Section 4 we end with our impossibility results.

2. DEFINITION

A protocol is secure if "all malicious parties in the real execution of the protocol can be simulated in an ideal execution." Our primary modifications to this mantra are that (1) our ideal adversaries do not communicate during an ideal execution —to capture their independence— and (2) we require an efficient function f which "reconciles" separate ideal adversary views into a single output that is indistinguishable from the *joint* views of the real adversaries. Our definition must ensure that whatever malicious players can learn by comparing notes *after* a real execution, they can also learn after an ideal execution. Of course, it would be preferable if the malicious players, separated during the game, remain separated forever after, but players do not live in Faraday cages forever! (See Remark after the definition for a further discussion of how our definition achieves this.)

As already mentioned, in our basic scenario only one player is active at any time, and between the actions of two players each player receives his own partial information (possibly empty). [7] We tightly couple a real execution to its corresponding ideal one by requiring that all information about

[6]The original definition and construction of uniZK proofs is presented in [15]. However, for our application, a weaker version (requiring weaker assumptions) suffices.

[7]Our definition generalizes to cases where multiple ideal players take actions at the same time, but this requires corresponding our protocol to make use of simultaneous broadcast channels. For clarity, we defer this generalization and focus on the single action case.

the global state of the ideal execution can be extracted (perhaps in exponential time) from the message traffic of the real execution. In order to make sure that every human player "knows" his own partial information, we require that a real player's view of the message traffic can be (efficiently) mapped into an ideal player's state.

Games

A finite game G with n players and s stages consists of

- ▷ Σ, a finite set of game states,
- ▷ X_i^j, a finite set of possible actions for player i at stage j,
- ▷ Y_i^j, a finite set of possible outputs for player i at stage j representing partial information about the global state,
- ▷ TURN : $[1...s] \rightarrow [1 \dots n]$, a function which maps each stage to the player who must take an action during that stage, and
- ▷ $g^1, \dots, g^s$, a sequence of probabilistic stage functions where $g^j : (X^j_{\text{TURN}(j)} \times \Sigma) \rightarrow (Y_1^j \times \cdots \times Y_n^j \times \Sigma)$.

In an ideal execution, an honest human player, $\mathscr{H}_i$, is a sequence of probabilistic (strategy) functions, $\{\mathscr{H}_i^1, \dots, \mathscr{H}_i^s\}$.[8] A malicious player, $\mathscr{A}_i$, is an efficient interactive Turing machine which takes a unary input (and as usual retains state between stages). Such $\mathscr{A}_i$ chooses actions for player i during an execution of the game and additionally, at the end of the game, produces an auxiliary output.

The global state of a game at stage j, denoted σ^j, is a string encoding all actions taken by all players prior to stage j, all outputs sent to the players prior to round j and any private randomness used by the stage functions. [9] The local state of a player i, denoted σ_i^j, consists of the actions taken by i and all of the outputs received from the trusted party by i prior to stage j. Letting Σ_i^j be the set of all possible local states for player i at the beginning of stage j, then each $\mathscr{H}_i^j$ maps Σ_i^j into the action set $\rightarrow X_i^j$.

An action, $x_i^j \in X_i^j$ is a *public action* if, whenever the active player i of stage j chooses action x_i^j, then x_i^j is part of the partial information each player receives at the end of stage j. (Notice that this is a constraint on the stage function g^j.)

Ideal Executions Let C be a proper subset of $[1, n]$. In an ideal execution of G with security parameter 1^k and trusted party $\mathscr{T}_G$, the players are partitioned into a set of ideal adversaries, $\{\mathscr{A}_c : c \in C\}$, and a set of human players, $\{\mathscr{H}_i : i \notin C\}$.

Prior to the first stage of an ideal execution of G, the trusted party generates σ^1, which, without loss of generality, contains all the random coins needed to evaluate the stage functions of G. For stage j, suppose the trusted party has global state σ^j and that $i = \text{TURN}(j)$. If $i \notin C$, then the honest player function $\mathscr{H}_i^j(\sigma_i^j)$ is invoked on player i's local state, σ_i^j, to compute an action x_i^j which is sent to $\mathscr{T}_G$. If $i \in C$, the ideal adversary $\mathscr{A}_c$ is given unary input 1^k and (running from his previous state) computes an action x_i^j which is delivered to $\mathscr{T}_G$. The trusted party then computes the partial information vector $(\sigma^{j+1}, y_1^j, \dots, y_n^j) = g^j(\sigma^j, x_i^j)$. The players receive their partial information, one-by-one in lexigraphic order, via the following process : for the ith player, $\mathscr{T}_G$ first asks all the players, one-by-one, if player i should receive his partial information. If no player objects, $\mathscr{T}_G$ delivers y_i^j to player i. Else, if a player objects, then $\mathscr{T}_G$ informs all players that the game has been aborted (and appends "abort" to the global state). The execution continues in a similar fashion for all s stages.

We use the notation $e_I \leftarrow \langle\ (\mathscr{H}_i : i \notin C) \ \overset{1^k}{\longleftrightarrow}\ (\mathscr{A}_c : c \in C)\ \rangle$ to denote that e_I is a random ideal execution of G with security parameter 1^k, ideal adversaries $\{\mathscr{A}_c : c \in C\}$, and human players $\{\mathscr{H}_i : i \notin C\}$. If e_I is an ideal execution, the final output of adversary $\mathscr{A}_c$ is denoted $\text{OUT}_c(e_I)$. When indexed by a set C, $\text{OUT}_C(e_I)$ consists of the concatenation of $\text{OUT}_c(e_I)$ for all adversaries $c \in C$.

Protocols An r-round, n-player protocol Π for a game G is defined by (1) a set of n efficient oracle ITMs, $\{T_i, \dots, T_n\}$, each of which makes at most s oracle calls, (2) r communication channels, $\{B_1, \dots, B_r\}$, (3) a (not necessarily efficient) extraction function EXT described below, and (4) an efficient player-extraction function EXT_i for each player i. We refer to machine T_i as the *interpreter* for player i.

Real World Executions Let C be a subset of $[1, n]$. An execution of protocol P with human players $\mathscr{H}_1, \dots, \mathscr{H}_n$ and a set of malicious players $\{A_c : c \in C\}$, is defined as follows. The ith player in this execution is A_i if $i \in C$, and $T_i^{\mathscr{H}_i}$ otherwise. Since the ITM T_i is an oracle machine making at most s oracle calls, by the notation $T_i^{\mathscr{H}_i}$, we denote the interpreter T_i running with oracle access to human player $\mathscr{H}_i$ such that the jth oracle call of T_i is answered by the probabilistic function $\mathscr{H}_i^j$.

Prior to round 1, all parties are given a security parameter 1^k as input. For each round t of P, all communication occurs via channel B_t, for honest and malicious players alike.

As before, we use the notation

$$e \leftarrow \langle\ (T_i^{\mathscr{H}_i} : i \notin C)\ \overset{1^k}{\longleftrightarrow}\ (A_c : c \in C)\ \rangle$$

to denote that e is a random execution of P with human players $\{\mathscr{H}_i : i \notin C\}$ and adversarial algorithms $\{A_c : c \in C\}$. The message traffic in a real execution e is denoted $\text{TRAFFIC}(e)$ and consists of all messages sent during an of execution of P. Similarly, $\text{VIEW}_i(e)$ denotes the view of player i and consists all messages received by player i as well as the random coins used by either A_i or T_i (depending on whether $i \in C$ or not respectively). As before, $\text{VIEW}_C(e)$ is the concatenation of $\text{VIEW}_c(e)$ for all players $c \in C$. Finally, the (possibly exponential time) function EXT : TRAFFIC $\rightarrow \Sigma$ maps the traffic of an execution to a global game state in G and the efficient function $\text{EXT}_i : \text{VIEW}_i(e) \rightarrow \Sigma_i$ maps a player's view to a human player state.

Definition 1 (Collusion-free Protocol). *A protocol P is a* Collusion-free *realization of game G if for every proper subset C of the players and any set of efficient adversarial algorithms $\{A_c : c \in C\}$, there exist ideal adversaries $\{\mathscr{A}_c : c \in C\}$ such that for any set of honest human players $\{\mathscr{H}_i : i \notin C\}$, for all $c \in C$, there exists an efficient function f so that the following pair of ensembles are computationally indistinguishable*

$$\left\{ \begin{array}{l} e \leftarrow \langle\ (T_i^{\mathscr{H}_i} : i \notin C)\ \overset{1^k}{\longleftrightarrow}\ (A_c : c \in C)\ \rangle : \\ (\text{VIEW}_C(e), \text{EXT}_c(\text{VIEW}_c(e)), \text{EXT}(\text{TRAFFIC}(e))) \end{array} \right\}_k \quad (1)$$

$$\left\{ \begin{array}{l} e_I \leftarrow \langle\ (\mathscr{H}_i : i \notin C)\ \overset{1^k}{\longleftrightarrow}\ (\mathscr{A}_c : c \in C)\ \rangle : \\ (f(\text{OUT}_C(e_I)), \sigma_c^s(e_I), \sigma^s(e_I)) \end{array} \right\}_k \quad (2)$$

Our notation above follows closely the convention set forth in [13] and [3]. Briefly, the set notation in (1) denotes the

[8]We may think of these functions as being non-deterministically chosen to capture the whimsical manner in which a player acts in a game.

[9]In this section, a superscript indexes time, a subscript indexes the player.

ensemble of probability spaces over the tuple of strings, $(\textsc{view}_C(e_I), \textsc{ext}_c(\textsc{view}_c(e_I)), \textsc{ext}(\textsc{traffic}(e_I)))$, which is indexed by a security parameter k, and generated by randomly sampling a Real protocol execution e and applying the appropriate VIEW and EXT functions. Similarly, the set notation in (2) denotes the ensemble of probability spaces over $(f(\textsc{out}_C(e)), \sigma^s_c(e), \sigma^s(e))$, also indexed by a security parameter k, but generated by randomly sampling an Ideal execution and applying the OUT and σ functions as specified.

Remark. The reconciliation function f models the privacy guarantee that malicious players cannot combine their information *after* an execution to learn *extra* information about an honest player's strategy that they could not learn in an ideal execution.

There are several ways that one could model this requirement. An optimist might consider removing f from the definition and just requiring that $(\textsc{out}_C(e), \ldots)$ be indistinguishable from $(\textsc{view}_C(e_I), \ldots)$. This would imply that for $c \in C$, $\mathscr{A}_c$ must generate the public broadcast traffic of the protocol, and must do so without knowledge of any other player's private outputs.

In a real execution, however, the broadcast traffic in each view perfectly coincides, bit for bit! The optimistic definition would thus obligate each $\mathscr{A}_1$ to generate the same broadcast traffic independently during the game. Informally, for every round j, $\mathscr{A}_1$ must produce *exactly* the message broadcast by A_2 on input σ^j_2 *without* knowing σ^j_2 — a task which seems impossible.

By using the reconciling function f, and, *crucially*, the fact that the game is finite, we are able to circumvent this difficulty and design a protocol which meets our definition. At the same time, since f is efficiently computable, our definition still captures the stated privacy guarantee.

3. CONSTRUCTION

THEOREM 1. *If trapdoor permutations exist, any finite, partial-information game with publicly observable actions has a collusion-free protocol whose communication channels consist of broadcast and plain envelopes.*

In this section, we present our Collusion-free protocol which is based on the original GMW [11] protocol for secure multi-party computation. We feel it is most natural to present our protocol by making references to the steps of the GMW protocol, and therefore, we provide an overview of the important pieces of the GMW protocol in Appendix A. (A more thorough explanation of the GMW protocol can be found in [10].)

As noted in the Introduction, our protocol for playing any finite game G consists of two subprotocols, a *pre-processing subprotocol*, P_1, immediately followed by a *computation subprotocol*, P_2. (We denote the entire protocol as $P_1 + P_2$.) The pre-processing phase computes a function PRE in such a way that, even if players maliciously collude, no player has knowledge of another player's private output. The computation phase uses the private outputs from P_1 to perform a secure evaluation of the stage functions of G in a verifiably unique fashion by using the uniZK proofs introduced in [15]. Since G is finite, one can upper bound the total number of bits in all the theorems that need to be proved during the computation process. Hence, a bounded theorem uniZK proof system suffices for P_2. [10]

Physical Envelopes As we show in Section 4, some type of physical channel assumption is necessary to achieve a collusion-free protocol for general games. Thus, our protocol assumes the existence of physical envelopes that support the operations $\textsc{Send}(R, m)$ and $\textsc{Receive}(S)$ which satisfy the following properties: (1) *Binding* —when a receiver, R, calls $\textsc{Receive}(S)$, he learns the values m for all previous calls to $\textsc{Send}(R, m)$ made by sender S (2) *Hiding* —when a sender calls $\textsc{Send}(R, m)$, no one gains any information about m and when receiver calls $\textsc{Receive}(S)$ no one besides R gains any information about m (3) *Public* —when a sender calls $\textsc{Send}(R, m)$ everyone learns the string "$(\textsc{Send}, S, R)$" and when a receiver calls $\textsc{Receive}(S)$, everyone learns the string "$(\textsc{Receive}, S, R)$." We think of $\textsc{Send}(R, m)$ as corresponding to an action where the sender puts m in an envelope, writes his name on the envelope and hands it to R. We then think of $\textsc{Receive}(S)$ corresponding to the action where the receiver publicly opens all envelopes from S and privately reads their contents. [11]

The Function PRE. The function PRE is a probabilistic function that maps n private inputs $x_1, \ldots, x_n$ to n private outputs $y_1, \ldots, y_n$. However, we find it easier to explain PRE as a function from a single public input 1^k to a common public output *Common* and private outputs $o_1, \ldots, o_n$. That is, for each i, $x_i = 1^k$ and $y_i = (Common, o_i)$.

Let the polynomial $p(k)$ upper-bound the total number of bits in all of the theorems that need to be proven during the Computation phase of our protocol running on security parameter 1^k. Due to the structure of our protocol, $p()$ will only depend on the stage functions of G, and the security parameter k.

Let G be the generator for a public-key encryption scheme, G_{uzk} be the generator for a p-bounded-theorem uniZK system and $\textsc{com}(string, coins)$ be the commitment algorithm for a perfectly binding commitment scheme. The function PRE then operates by :

1. Running $G(1^k)$ n times to obtain matching public and secret encryption keys $(\textsc{pk}_1, \textsc{sk}_1), \ldots, (\textsc{pk}_n, \textsc{sk}_n)$.
2. Running $G_{uzk}(1^k, p(k))$ n times to obtain matching public and secret uniZK keys $(\textsc{upk}_1, \textsc{usk}_1), \ldots, (\textsc{upk}_n, \textsc{usk}_n)$.
3. Selecting random strings $\tau_1, \ldots, \tau_n$, $R_1, \ldots, R_n$, $r_1, \ldots, r_n$, $R'_1, \ldots, R'_n$, $r'_1, \ldots, r'_n$, and $v_1, \ldots, v_n$.
4. Computing commitments to the random tapes, $\textsc{com}(R_1, r_1)$, $\ldots, \textsc{com}(R_n, r_n)$, $\textsc{com}(R'_1, r'_n), \ldots,$ $\textsc{com}(R'_n, r'_n)$, and $\textsc{com}(\textsc{sk}_1|\textsc{usk}_1, v_1), \ldots, \textsc{com}(\textsc{sk}_n|\textsc{usk}_n, v_n)$.

The output *Common* is then the 6-tuple consisting of

- ▷ $(\textsc{pk}_1, \ldots, \textsc{pk}_n)$, the n-vector of public encryption keys
- ▷ $(\textsc{upk}_1, \ldots, \textsc{upk}_n)$, the n-vector of public uniZK keys
- ▷ $(\textsc{com}(R_1, r_1), \ldots, \textsc{com}(R_n, r_n))$, an n-vector of committed random tapes
- ▷ $(\textsc{com}(R'_1, r'_1), \ldots, \textsc{com}(R'_n, r'_n))$, an n-vector of committed random tapes
- ▷ $(\tau_1, \ldots, \tau_n)$, the n-vector of common random strings
- ▷ and $\textsc{com}(\textsc{sk}_1|\textsc{usk}_1, v_1), \ldots, \textsc{com}(\textsc{sk}_n|\textsc{usk}_n, v_n)$, the n-vector of committed secret keys.

[10]The original uniZK paper[15] provides a system attaining a stronger multi-theorem notion of uniZK but requires a specific number theoretic assumption instead of working for any trapdoor permutation. In the full version of this paper, we provide a p-bounded theorem uniZK system based on one-way permutations.

[11]Note that alternatively, a single invocation of a channel allowing simultaneous delivery of private messages would suffice for our protocol.

The private output o_i for player i is the 7-tuple consisting of $(\textsc{sk}_i, \textsc{usk}_i, R_i, r_i, R'_i, r'_i, v_i)$.

The Pre-Processing Phase

The pre-processing protocol P_1 proceeds as follows:

1. The players run Steps 1 - 4 of the GMW protocol to compute the function $\textsc{Pre}$ on input 1^k.
2. As in Step 5 of GMW, the players broadcast encryptions of their shares of the public output *Common* and provide zero-knowledge proofs that the encryptions of the shares are correct. However, departing from the GMW protocol, the players do not yet broadcast their shares of the private outputs.
3. Envelopes are used to exchange the shares of the players' private inputs. Each player i invokes $\textsc{Send}(j, m_j)$ once for each other player j where m_j is player i's share of private output o_j. Once each player has observed ("Send", i, j) for each pair i and j, each player then invokes $\textsc{Receive}(j)$ once for each other player j. If any player observes ("Receive", i, j) before observing ("Send", j, k) for all $k \neq j$ then that player broadcasts an abort flag. (That is, if player j opens his envelope from player i before sending all n of his envelopes, he is presumed to be cheating, and so each honest player is instructed to abort.) Each player then reconstructs his private output $o_i = (\textsc{sk}_i, \textsc{usk}_i, R_i, r_i, R'_i, r'_i, v_i)$. To verify that this private output is correct, player i computes $\textsc{com}(\textsc{sk}_i|\textsc{usk}_i, v_i)$, $\textsc{com}(R_i, r_i)$, and $\textsc{com}(R'_i, r'_i)$ and checks that the resulting strings match the corresponding commitment strings included in the *Common* public output. If any of these checks fail, the player broadcasts an abort flag.

Computation Phase

In the computation protocol, denoted P_2, the players run a sequence of secure computation protocols to evaluate the stage functions of G. Each of the secure computation protocols, which we refer to as UNI-GMW protocols, are based on GMW but differ as follows:

- ▷ A UNI-GMW protocol only works on finite functions (independent of the security parameter k).
- ▷ The players do not perform Step 1 of GMW, but instead use the keys $(\textsc{pk}_i, \textsc{sk}_i)$ computed during P_1.
- ▷ The players do not perform Step 2 of GMW, but instead use the committed random tapes $\textsc{com}(R_i)$ computed during P_1.
- ▷ The players perform Steps 3 - 5 as in GMW except that whenever player i is instructed to give a zero-knowledge proof that an encryption or commitment is correct he instead gives a single uniZK proof using key $\textsc{upk}_i$ and string τ_i that "The encryption or commitment is correct *and* that the encryption or commitment is computed properly using random coins committed in $\textsc{com}(R'_i, r'_i)$."

We now specify the Computation phase as follows:

1. The players run a UNI-GMW protocol on empty input to generate a random sharing of initial global state σ^1. Interpreter T_i initializes a variable, σ_i, to null, and maintains it throughout this phase by appending to it, all of the actions taken and partial information received by player $\mathscr{H}_i$.
2. Let $i = \textsc{turn}(j)$. In order to emulate stage j of G, interpreter T_i queries its oracle $\mathscr{H}_i$ on input σ_i to obtain an action x_i^j (this is the action that the human player would send in an ideal execution). All interpreters then engage in $n+1$ sequentially executed UNI-GMW protocols to compute the following functions (we are splitting the stage function g_j into $n+1$ smaller steps):
   - ▷ $\textsc{share}_j$: This function computes a sharing of σ^{j+1}. Player i's private input to this step consists of his share of the global state σ^j and his action x_i^j. All other players provide their own share of σ^j as their private input.
   - ▷ $\textsc{output}_{i,j}$: This function computes player i's output in stage j based on the stage function g_j. The private inputs to this function are the same as for $\textsc{share}_j$.

     For each party i (in order from 1 to n), the interpreters run a UNI-GMW protocol on the inputs to produce the private partial information y_i^j for player i. Each interpreter then updates its state, σ_i, by appending y_i^j.

   For all $j > 1$, the inputs to $\textsc{share}_j$ and $\textsc{output}_{i,j}$ include outputs produced during earlier steps. Therefore, when sharing these inputs, the player provides an additional uniZK proof that the input being shared is the same as the output received in the previous step.

4. IMPOSSIBILITY RESULTS

All of our impossibility results make use of the ideal game H, depicted in Figure 4, which is finite and has publicly observable actions.

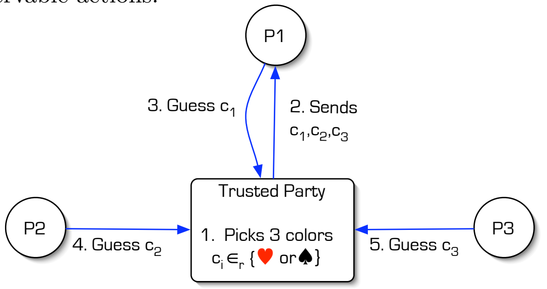


Figure 1: Game H. The trusted party privately hands Player 1 three cards, each of which is red with probability $\frac{1}{2}$ and black otherwise. Player 1 then (publicly) guesses the color of card 1, Player 2 (publicly) guesses the color of card 2 and Player 3 (publicly) guesses the color of card 3. The trusted party then reveals to everyone the identities of the players who guessed correctly.

Envelopes are necessary

Here we prove that a collusion-free protocol which works for any finite game must use some type of physically private channel.[12] The intuition is that unless the adversaries have distinct views of the protocol execution, they can easily simulate a monolithic adversary.

[12] Although our proof is with respect to our particular definition of collusion-freeness, the structure of the proof does not rely on the specific details of our model. A similar theorem could be proved with respect to any sufficiently strong definition of collusion-freeness.

THEOREM 2. *The game H (which is finite and has publicly observable actions) has no collusion-free protocol whose* only *communication channel is broadcast.*

LEMMA 1. *In the game H, a traditional (monolithic) adversary can perform attacks which cannot be simulated in a collusion-free ideal execution of H as defined in Section 2.*

Proof Sketch:
Recall that a traditional (monolithic) adversary receives all messages sent to any of the members of the coalition C and dictates all messages to be sent by any members of C. Consider a coalition $C = \{1, 2\}$. An adversary $A_{1,2}$ that receives all messages sent to player 1 can determine the color of cards 1 and 2. Therefore, $A_{1,2}$ can instruct player 1 and player 2 to each guess correctly in every execution.

Now consider a coalition-free ideal execution of H. In such an ideal execution, after player 1 learns the color of the cards, the only information from player 1 that player 2 receives from the trusted party is the guess player 1 makes about card 1. Therefore, if player 1 wins (i.e., correctly guesses the color of card 1) with probability 1, then player 2 receives no information about the color of card 2 (since the color of each card is chosen independently by the trusted party). In this situation, the mutual information between the color of card 2 and the guess of player 1 (conditioned on Player 1 winning) is zero. Therefore, player 2 cannot win with probability greater than $\frac{1}{2}$, and thus there cannot exist ideal adversaries $\mathscr{A}_1$ and $\mathscr{A}_2$ that cause player 1 and player 2 to each win with probability 1. $\square$

LEMMA 2. *If protocol P uses only broadcast channels, then any attack performed by a (traditional) monolithic adversary A can be simulated by a set of independent adversaries $\{A_c : c \in C\}$ in a collusion-free real execution of P (as defined in Section 2).*

Proof Sketch: With only broadcast, independent adversaries in a collusion-free real execution can emulate any monolithic adversary A, and hence collusion-free security is not possible. To show this formally, we construct new adversaries $\{A'_c : c \in C\}$ who operate in a collusion-free real execution of P and generate exactly the same distribution of messages as A.

Our set of adversaries, A'_c work as follows. Before the protocol starts, the adversaries A'_c agree upon a random tape, r, via a coin-flipping protocol. Once the protocol starts, each machine begins to execute an independent copy of A using the random tape r. During the protocol, each independent adversary feeds all of the broadcast messages into his personal copy of A. Whenever it is A'_c's turn to send a message, he broadcasts whatever message A would have sent on behalf of party c. Since each independent A'_c is supplying exactly the same inputs and random tape to his copy of A, the $|C|$ copies of A will always have the same state, and will generate exactly the same messages as the monolithic adversary, A, would generate in a traditional real execution. $\square$

Proof Sketch of Theorem 2: Lemma 2 states that when a protocol P uses only broadcast, then any attack by a traditional (monolithic) adversary can be simulated by independent adversaries in a collusion-free real execution. If P were collusion-free, then these attacks could also be simulated in a collusion-free ideal execution. If P worked for any finite game then it would contradict Lemma 1. Thus, a collusion-free broadcast protocol for any finite game cannot exist.

Non-finite Games are Impossible

Here we prove that there exist some non-finite games which do not admit collusion-free protocols. Keeping in mind the necessity of secret envelopes for collusion-free protocols, the intuition is that in some non-finite games, adversaries can repeatedly use game-instrinsic steganography to convey whatever information was previously exchanged in the secret envelopes.

THEOREM 3. *There exists a* non-finite *game H' with publicly observable actions that has no collusion-free protocol whose communication channels consist of broadcast and plain envelopes.*

Proof Sketch: Let H' be a repeated version of game H in which after each round of playing H, player 1 decides whether the game should continue or terminate. (That is, the game H is repeated an a priori unbounded number of times.) After player 1 decides to terminate, the trusted party announces which players guessed correctly in each round.

Let P be a collusion-free protocol for H'. From Theorem 2 we know that any collusion-free protocol for a finite game must use envelopes. The same is true for non-finite games (by the same argument) and therefore P must use envelopes. There are three cases to consider.

Case 1: The envelopes are used once *after* player 1 decides to terminate the game. At this point, all of the players' moves have to be determined by the transcript of P thus far. Otherwise, the players' in a real execution could condition their moves based on the knowledge of when H' terminates— something not possible in an ideal execution of H'. If the players' moves have already been determined, then the phase of P in which these moves were chosen used only broadcast. Therefore, by an argument similar to one presented in Lemma 2, during the entire period of P when actions are chosen, the bad players can simulate any monolithic adversary. This contradicts collusion security.

Case 2: The envelopes are used once *before* player 1 decided to terminate the game. Thus, there are an arbitrary number of rounds that occur after the use of the envelopes. Let M be an upper bound on the number of bits sent to any player over the envelope channel. (Note since the number of rounds is unbounded, M must be independent of the number of rounds). Consider a malicious player 1 and player 2 who agree on an M bit random string R prior to the protocol.[13] In each round $i \leq M$ of H, player 1 guesses "Red" if the i^{th} bit of the (concatenation of) the message(s) received by player 1 in the physically private channel is equal to the i^{th} bit of R and guesses "Black" otherwise. Note that after round M, player 2 has knowledge of all messages received by player 1 throughout the protocol. Therefore player 2 can compute any value that player 1 can compute. Thus in round $M + 1$ of the protocol, since player 1 can compute the color of card 2, player 2 must also be able to compute the color of card 2. This means that player 1 and player 2 can both correctly guess the color of their card in round M with probability 1. This is impossible in an ideal execution of H.

Case 3: The envelopes are used at multiple rounds throughout the protocol. If all uses of the envelope channel occur after player 1 decides to terminate H', then the

[13]Alternatively, in the case that M is not known prior to the start of the protocol players 1 and 2 could agree on the seed of a pseudo-random number generator.

argument from Case 1 yields a contradiction. Otherwise, consider a malicious player 1 and player 2 who agree before the game on a random bit b. Let round r be the first round of H in which the envelopes are used after the completion of round 1. (Note if such an r does not exist, then an argument similar to Case 2 yields a contradiction.) During each round $i < r$ of H, player 1 correctly guesses the color of card 1 and player 2 makes his guess randomly. Then, when the physical channel is used prior to round r, player 1 privately sends to player 2 (unobserved by the other players) the color of card 2 in round 1. Subsequently in round r, player 1 again correctly guesses the color of card 1 and player 2 makes a guess which is equal to the color of card 2 in round 1. Player 1 then terminates the game. In this game, player 1 has guessed correctly in every round and player 2's guess in round r is equal to the color of card 2 in round 1. This type of correlation is impossible in an ideal implementation of game H', thereby contradicting the collusion-free security of P. □

Private-Action Games are Impossible

Here we prove that games with private actions generally do not have collusion-free protocols. Intuitively, when a player j takes a private action in an *ideal execution*, player j "cannot" use his choice of action to convey information to a co-conspirator. However, in any *real execution*, the message traffic must somehow encode (via encryption, say) player j's private action. Since the messages sent by player j are seen by other players (as opposed to just the trusted party) j can choose his action so that the message traffic conveys some information to a co-conspirator.

Note, however, that one can easily transform a private action game G into a similar game G' with publicly observable actions in such a way that G and G' are identical with respect to what can be accomplished by a traditional monolithic adversary. This can be done as follows: Every time a player must take a private action in G chosen from set A, the trusted party in G' sends the player a random mapping f from A to $\{0,1\}^k$, the player publicly announces $f(a)$, and the trusted party updates the global state as though the player had chosen a. The game G' has more opportunities for game-intrinsic steganography than G. However, in general, the power of a maliciously coalition of players is much less in a collusion-free protocol for G' than in a traditional secure protocol for G, which in some sense, is the best one can hope for.

THEOREM 4. *There exists a finite game, H^*,* with private actions *that has no collusion-free protocol whose communication channels consist of broadcast and plain envelopes.*

Proof Sketch:

We choose a simple variation of H as a counterexample for this theorem, but our techniques apply to a broad class of games with private actions including the game *Poker′* discussed in the Introduction.

Let the private-action game H^* proceed as H does with the following modifications : Player 1 *privately* (instead of publicly) guesses the color of card one. Denote this guess g_1. Additionally, the trusted party sends a private message m_2 to player 2 such that, with probability 1/10, m_2 contains player 1's guess, i.e., $m_2 = g_1$, and otherwise is the symbol $\perp$. Likewise, the trusted party sends a private message m_3 to player 3, such that (independently) with probability 1/10, m_3 contains player 1's guess, i.e., $m_3 = g_1$, and otherwise is the symbol $\perp$.

Let T^i_R and T^i_B denote the set of honestly generated transcripts of protocol P whose local extraction yields a player i guess of "Red" and "Black" respectively. In any collusion-free protocol P for H^* both sets must be non-empty and statistically far (the latter being a consequence of the existence of the local extraction function).

Let k^i be the first protocol round in which the distribution of protocol-round k^i messages in T^i_B is statistically far from the distribution of round k^i messages in T^i_R.

We observe that at round k^1, player 1 must know the color of Cards 1, 2, and 3 (in the sense that they can be extracted from player 1's view) because the honest interpreter must send the colors of Cards 1, 2, 3 to it's corresponding human player before it can receive the action (guess) from its human player.

We first consider the case where $k^1 < k^2$. There are two sub-cases. The first sub-case is when Player 1's round k^1 message is observed by Player 2. If so, consider the following malicious strategy for players 1 and 2. Before the game, players 1 and 2 agree on a random string R. In round k^1, player 1 computes the message M_{red} that would be sent by the honest interpreter in a "Red" transcript and the message M_{black} that would be sent by the honest interpreter in a "Red" transcript. If $M_{red} \cdot R$[14] differs from $M_{black} \cdot R$ then player 1 chooses the guess g_1 such that $M_{g_1} \cdot R$ corresponds to the color of Card 2.[15] Otherwise, player 1 chooses g_1 to correspond to the color of Card 2. If player 2 receives an observable $m_2 \neq \perp$ then player 2 selects m_2 as his guess. Otherwise, player 2 selects the action corresponding to $M \cdot R$ (where M is player 1's round k message).[16] This malicious strategy allows player 2 to correctly guess card 2 with probability greater than $\frac{1}{2}(1 + \frac{1}{10})$, which violates the collusion-free property of the protocol since winning with such odds is impossible in an ideal implementation of H^*.

The second sub-case is when Player 1's round k^1 message is observed by Player 3 but not by Player 2 (that is, the message is sent privately in an envelope to Player 3). In this case, if Player 1 and Player 3 maliciously collude, Player 1 can send the color of Card 3 to player 3 in the envelope (in addition to the honestly generated message M) and player 3 can guess the color of card 3 correctly with probability greater than $\frac{1}{2}(1 + \frac{1}{10})$ (which is impossible in an ideal implementation of H^*.[17]

Next, consider the case where $k^1 \geq k^2$. Here we observe that at round k^1, player 2 must be able to extract m_2 from the transcript because the honest interpreter must send m_2 to the human player 2 before the interpreter can receive the action (guess) from the human player. We now consider the following malicious strategy for players 1 and 2. Before the game, players 1 and 2 agree on a random string R. In round k^2, player 2 computes the message M_{red} that would be sent by the honest interpreter in a "Red" transcript and the message M_{black} that would be sent by the honest interpreter in a "Red" transcript. If $M_{red} \cdot R$ differs from $M_{black} \cdot R$ then player 2 chooses his guess g_2 such that $M_{g_2} \cdot R = 0$ if, and

[14]Where $M \cdot R$ denotes the inner product of M and R.

[15]Here, 0 means "Black" and 1 means "Red".

[16]We pessimisticly assume that $\Pr[M_{black} \neq M_{red}] < 1/10$. If the probability were greater than 1/10, then player 2 would always choose the action corresponding to $M \cdot R$.

[17]In fact, even if one assumes stronger private channels such as envelopes which somehow limit the number of bits sent or magically ensure that the message sent is distributed correctly, Player 1 and Player 3 could still collude in the same way that Player 1 and Player 2 colluded in the proof of the first sub-case.

only if, $m_2 = \perp$. Let M be the round k^2 message sent by player 2. Player 1 then computes $v = M \cdot R$ and selects his action g_1 so that $g_1 =$ "*Black*$''$ if, and only if, $v = 0$. This malicious strategy allows player 1 to guess "Black" more often when Player 2 receives $\perp$. However, in the ideal game, player 1 has no information about whether player 2 receives $\perp$. This contradicts the collusion-free property of the protocol. $\square$

Acknowledgments. We would like to thank Oded Goldreich, Rafael Pass, and Chris Peikert for helpful comments.

5. REFERENCES

[1] M. Backes and C. Cachin. Public-key steganography with active attacks. In *TCC '05*, 2005.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for fault-tolerant distributed computing. In *Proc. of STOC '88*, pages 1–10, 1988.

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

[4] C. Cachin. An information-theoretic model for steganography. In *Proc. of Information Hiding '98*, pages 306–318, 1998.

[5] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd FOCS*, pages 136–145, 2001.

[6] R. Canetti, U. Feige, O. Goldreich, and M. Noar. Adaptively secure multi-party computation. In *Proc. 28th STOC*, pages 639–648, 1996.

[7] D. Chaum, C. Crépeau, and I. Damgård. Multi-party unconditionally secure protocols. In *STOC '88*, 1988.

[8] C. Crépeau. A secure poker protocol that minimizes the effects of player coalitions. In *Crypto '85*, volume 218 of *LNCS*, pages 73–86. Springer, 1986.

[9] Y. Dodis and S. Micali. Parallel reducibility for information-theoretically secure computation. In *CRYPTO '00*, pages 74–92, 2000.

[10] O. Goldreich. *Foundations of Cryptography*, volume 2, chapter 7 (General Cryptographic Protocols). Cambridge University Press, 2004.

[11] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87*, pages 218–229, 1987.

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

[13] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, Apr. 1988.

[14] S. Katzenbeisser and F. A. P. Petitcolas. Defining security in steganographic system. In *Security and Watermarking of Multimedia Contents IV*, pages 260–268, 2002.

[15] M. Lepinski, S. Micali, and abhi shelat. Fair-zero knowledge. In *TCC 2005*, pages 245–263, 2005.

[16] J. L. Nicholas Hopper and L. von Ahn. Provably secure steganography. In *Crypto '02*, 2002.

[17] A. Shamir, R. Rivest, and L. Adleman. Mental poker. Technical report, MIT, 1978.

APPENDIX

A. THE GMW PROTOCOL

The GMW protocol allows a set of n players to privately and correctly evaluate any probabilistic function F mapping private inputs $x_1, \ldots, x_n$ (one for each player) to private outputs $y_1, \ldots, y_n$. The GMW protocol consists of two components. The first component is a protocol, which we refer to as GMW', that uses private channels to privately and correctly evaluate F in the presence of "honest-but-curious" players. The second component is a compiler, which we refer to as GMW", that transforms any honest-but-curious protocol that uses private channels into a broadcast protocol that is secure even in the presence of malicious players who deviate in an arbitrary fashion from the prescribed protocol.

GMW': The protocol GMW' takes in a probabilistic circuit for the function F and outputs a set of interactive Turing machines $\hat{M}_1, \ldots, \hat{M}_n$ one for each player in the protocol. Each machine $\hat{M}_i$ operates by dividing its input x_i into n shares (one for each player) in such a way that all n shares can be used to reconstruct the input but any $n-1$ shares provide no information about the input (an XOR sharing is one such sharing). The machines $\hat{M}_i$ then perform computation on these shares to evaluate the probabilistic circuit for the function F. At the end of the protocol each $\hat{M}_i$ has a share of each of the outputs y_j. Each $\hat{M}_i$ then (for each j) sends its share of y_j to machine $\hat{M}_j$.

GMW": The compiler takes as input the machines $\hat{M}_1, \ldots, \hat{M}_n$ from the honest-but-curious protocol described above and outputs a set of interactive Turing machines $M_1, \ldots, M_n$ one for each player in the protocol. The compiled GMW protocol proceeds as follows:

1. Each machine M_i runs the generator for a public-key encryption system to obtain a public encryption key PK_i and a matching private decryption key SK_i. Machine M_i then broadcasts PK_i and provides a zero-knowledge proof of knowledge of SK_i.
2. The machines $M_1, \ldots, M_n$ run a coin flipping protocol which produces for each i a commitment, $\text{COM}(R_i)$, to a random string R_i. The protocol is such that all participants can compute $\text{COM}(R_i)$ but only machine M_i can compute R_i.
3. Each machine M_i broadcasts a commitment to the initial state of $\hat{M}_i$ with random tape R_i and provides a zero-knowledge proof that this commitment is correctly computed. Note that the statement being proven is in NP since anyone who knows the coins used to create the commitment can easily verify the truth of the statement.
4. Machine M_i now proceeds to simulate an honest execution of the machine $\hat{M}_i$. Throughout the execution, M_i maintains a commitment to the current state of $\hat{M}_i$. Every time $\hat{M}_i$ wants to send a private message to $\hat{M}_j$, machine M_i (A) computes the message based on the current state of $\hat{M}_i$, (B) broadcasts an encryption of the message in the key PK_j, (C) broadcasts a commitment to the new state of T_i (after sending the message) and (D) provides a zero-knowledge proof that the announced encryption and the announced commitment are correctly computed based on the commitment to the previous state of T_i. Note that this statement is also in NP since anyone who knows the coins used to construct the commitments and the encryption can easily verify the truth of the statement. Upon receiving the encrypted message, M_j decrypts the message and announces a commitment to the new state of $\hat{M}_j$ (after receiving the message). Machine M_j then provides a zero-knowledge proof that this commitment was computed correctly.
5. Once the simulated execution reaches the final stage of the honest-but-curious protocol, each M_i is able to compute a share of each output y_j. Machine M_i then encrypts his share of y_j in the key PK_j, broadcasts this encryption and provides a zero-knowledge proof that the encryption is properly computed.