Steganography and Collusion in Cryptographic Protocols

by

Matthew Lepinski

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Sept 2006

Author .....
Department of Electrical Engineering and Computer Science
August 7, 2006

Certified by .....
Silvio Micali
Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Steganography and Collusion in Cryptographic Protocols

by

Matthew Lepinski

Submitted to the Department of Electrical Engineering and Computer Science
on August 7, 2006, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

Steganography, the hiding of covert messages inside innocuous communication, is an active area of cryptographic research. Recent research has shown that provably undetectable steganography is possible in a wide variety of settings. We believe that the existence of such undetectable steganography will have far reaching implications. In this thesis, we investigate the impact of steganography on the design of cryptographic protocols. In particular, we show that that all existing cryptographic protocols allow malicious players to collude and coordinate their actions by steganographicly hiding covert messages inside legitimate protocol traffic. Such collusion is devastating in many settings, and thus we argue that it's elimination is an important direction for cryptographic research.

Defeating such steganographic collusion requires not only new cryptographic protocols, but also a new notion of protocol security. Traditional notions of protocol security 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! We therefore put forward the notion of a collusion-free protocol which guarantees that no set of players can use the protocol to maliciously coordinate their actions. As should be expected, such a strong notion of security is very difficult to achieve. We show that achieving collusion-free security is impossible in a model with only broadcast communication and that even with physically private communication (e.g. physical envelopes) there are still many ideal functionalities that have no collusion-free protocols.

Fortunately, under natural assumptions collusion-free protocols exist for an interesting class of ideal functionalities. Assuming the existence of trapdoor permutations, we construct collusion-free protocols, in a model with both broadcast messages and physical envelopes, for every finite ideal functionality in which all actions are public.

Thesis Supervisor: Silvio Micali
Title: Professor

Acknowledgments

The thesis is based upon joint work produced by Silvio Micali, Abhi Shelat, and myself. An earlier version of this work [17] appeared in the proceedings of STOC 2006. I'd like to thank Abhi Shelat for being a joy to work with on this project and Silvio Micali for all of his advice and guidance throughout my graduate studies. Additionally, I'd like to thank Ran Canetti, Shafi Goldwasser, Ron Rivest, Chris Peikert and Rafael Pass for helpful comments and discussions at various stages of this work. I greatly appreciate the financial support of the National Science Foundation. This thesis is based upon work supported under an NSF Graduate Research Fellowship.

The successful completion of my graduate studies would have been impossible without the help and support of many others. I'd like to thank Seth Gilbert, Sergei Izmalkov, April Rasala Lehman, David Liben-Nowell, and Chris Peikert for working with me on other research projects unrelated to this thesis. I'd also like to thank my dear friends Michael Edwards, Michael Treaster, David Hafvenstein and Michael Thomas for frequently reminding me that there's more to life than theoretical computer science. And finally, I surely would have given up long before the completion of this thesis if not for the unconditional love of my wife, Liv Coleman, my parents, Steve and Ellen Lepinski, and our cats Patience and Mirth.

Contents

Chapter 1

Introduction

1.1 The Problem: Steganographic Collusion

Steganography

Steganography is the art of conveying a hidden message via a public and apparently innocuous one. For example, a photographer in a heavily censured 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 in various forms dates back to the ancient Greeks, but, in the past few years, it has become an active area of search within the cryptography community. Recent work in Steganography (e.g., [3, 19, 21, 20, 1]) has shown that in a wide variety of settings and models

> *Steganographic communication is provably impossible to detect whenever there is a minimum amount of entropy*

At a high level, the intuition behind these results is as follows. Suppose a distribution of "innocuous messages" has a fair amount of entropy. Then a sender can sample innocuous messages from the distribution until he comes up with one that hashes to his desired covert message. This means that anyone who knows what hash function the sender used can determine the covert message, but, for appropriately chosen hash functions, anyone who doesn't know the hash function will see a random-looking innocuous message.

Secure Multi-Party Computation

Secure *multi-party* computation [10] (SMC, for short) traditionally posits that *any* ideal interaction with a trusted party (also know as a *game with imperfect information*) can be tightly simulated by the players themselves. That is, even if a proper subset of the players collude and deviate from their prescribed instructions in an arbitrary and coordinated manner, SMC guarantees that the game unfolds with the same privacy and correctness as if an external trusted party handled all the crucial details of the interaction.

In the past two decades, SMC has been extensively investigated and improved; in particular, by (a) solely relying on physical assumptions [2, 6], (b) tolerating the dynamic corruption of players [5], and (c) guaranteeing its safe use as a subroutine within arbitrary and larger protocols [9, 4]. However, no prior SMC protocol could prevent malicious players from colluding; that is, from successfully coordinating their actions in order to disrupt the protocol's desired correctness or privacy constraints. Whether based on computational or physical assumptions, all prior results simply guarantee that "executing the ideal functionality with a subset of colluding players or executing the protocol with the same colluding players yields the identical results."

Collusion in SMC protocols

Let us now consider the impact of steganography on SMC protocols and see why malicious collusion is possible in all prior SMC protocols.

The key to collusion is communication. A *coalition* 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). Recall that Steganographic communication is provably impossible to detect whenever the distribution of acceptable messages has a minimum amount of entropy. Additionally, it is well known to cryptographers that

> *any secure protocol must have a lot of entropy* (as shown by [15] even in the case of simple encryption).

Therefore, due to the entropy in the messages of all secure protocols, malicious players can use steganography to hide covert messages within the traffic of protocol to collude by coordinating their actions. Moreover, such collusion is provably undetectable by the honest players!

In many settings, such collusion (even among a set of just two players) would be disastrous. In poker, for example, two players secretly sharing information about their hands have a tremendous —and illegal!— advantage. For example, consider a game of draw poker in which malicious player 1 has a pair of Aces and malicious player 2 has both four hearts and a pair of Kings. If the two players are playing independently (without knowledge of each other's hands) then will both surely keep their high pairs when it comes time to draw. However, if, via steganography, the players can collude and share information about their hands then player 2 will realize that his pair of Kings will always lose to player 1's pair of Aces. Therefore, player 2's best move is to discard his non-heart king and draw to a flush. It is easy to see that such collusion during a poker game will result in the honest players losing much more often than they should. Therefore, no existing SMC protocol is capable of guaranteeing a fair game of poker. Indeed, the ability to cryptographicly play a fair game of poker has been an elusive goal for the past 20 years. In 1985, Crépeau [7] introduced techniques for limiting the power of a coalition of bad players in Mental Poker, 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.

1.2 Our Solution: Collusion-Free Protocols

Understanding Our Goal

Of course steganography is not the only way for malicious players to coordinate their actions during the execution of an SMC protocol. Indeed, the very actions of the game itself may intrinsicly enable some amount of communication between players. For example, in poker, if a player folds then others can infer that he did not have four aces. Such intrinsic communication is clearly desirable: no one would play a game of poker in which a player's actions must be totally independent from his cards. As another example of intrinsic communication, in the game of bridge players are allowed to communicate a limited amount of information to their partners via their bids in the auctions. Such communication during the auction is

an integral part of a bridge game and yet any communication beyond what is legitimately allowed in the auction would give a partnership a massive and unfair advantage. Therefore, our goal is to prevent coalitions (to the maximum possible extent) by eliminating all illegitimate, extra communication between malicious players while preserving the legitimate, game-intrinsic communication.

It is the undetectability of steganographic collusion that makes it so difficult to eliminate. If good players in a protocol execution detect any *extra* (i.e., non protocol-specified) communication, then they should abort since its very presence indicates that bad players are at work. Yet the undectability of steganography is inherent in any secure protocol due to the entropy required for protocol security. We resolve this apparent contradiction by proving that

> *The presence of steganographic communication is provably impossible to detect, but its very presence is provably preventable in Poker, Bridge, and all similar games.*

The Notion of Collusion-Free Protocols

To help elucidate our new notion, we choose to defer all other complexities and introduce collusion-free protocols in the simplest possible setting: the static, stand-alone model without "fairness." Informally, this means that we shall focus on a single execution, where the bad players are fixed in advance, only one player is active at any given time, and any player can cause an abort. The number of bad players, however, can be arbitrarily high.

To define collusion-free protocols, we rely once more on the traditional Ideal-Real paradigm. We actually find it convenient (for variety of discourse, as well to be in line with game-theoretic tradition[1]) to refer to the abstract specification of a protocol as a *game*, and then consider two separate implementations for it, the ideal and the real.

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

  For clarity, we may refer to a player in the ideal implementation as a *human* player. Such a player communicates to the trusted party in a private and specific manner. Simplifying a bit (see Section 2 for more details), when it is his turn to play, a

[1] Technically, since in our cryptographic setting the players receive no payoffs, a game theorist might refer to such abstract specifications as a *game-form* and not a game.

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, as usual, 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 captures 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 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* 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.[2] Thus, even in an ideal hand of Poker, 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 (e.g., whether he bluffed or not).

[2]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!

- *Real Implementation.* This is the traditional implementation of a game G via a protocol P executed by just the players themselves, with again the caveat that *all extra communication channels are removed.*

  In addition, the real implementation preserves the "reactive nature" of game playing.[3] That is, a real player i has two distinct components: a human player, H_i, and an interactive Turing machine, T_i, specified by P. Human player H_i 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 H_i's *interpreter:* essentially, it transforms an action of H_i into a protocol message, processes "the message traffic," and presents H_i with proper partial information. In between 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, collusion-free protocols are actually quite tricky to define properly, *even assuming familiarity with the usual subtleties of secure computation.* Personally, we regard the very definition of collusion-freeness as a principal 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. Computer scientist have already started studying protocols in game-theoretic settings, and our framework may boost the robustness of such game-theoretic protocols.

[3] 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 to compute their own description."

Remarks.

- Let us observe that communication occurring before or after the protocol is not a problem: though both "extra and undetectable" by definition, it is already incorporated in the ideal implementation.[4] The problem is in the extra communication occurring *during* the protocol. 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 demand that, at least during critical portions, all communication is via broadcast only. But the use of broadcasting is not sufficient, even with our model's guarantee that no other channels are available to any players, whether good or bad. This is so because bad players may *subliminally* use the very channels specified by the protocol (including broadcast ones) to secretly coordinate themselves. It is thus the designer's responsibility to guarantee that *the protocol itself cannot be used to provide any additional power to a coalition.*

- Our work bars a main avenue of subliminal communication: that introduced by the protocol itself. Other *physical* avenues for such communication might, however, 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.) Barring 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* mechanisms.

- 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, not eliminate them!) Let us now uncover the difficulties in constructing collusion-free protocols.

[4]Notice, however, that it is the protocol designer's responsibility to ensure that bad players cannot use post-execution communication to disrupt the protocol's privacy constraint in any way.

Our Theorems

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[5] 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 *infinite* 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 provides solves Crépeau's open problem [7], because Poker is a finite game whose actions are publicly observable: the only actions are publicly announcing a

[5]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).

subset of $\{1, \ldots, 5\}$ (representing the cards in his own hand which the player wants to replace[6]) or announcing bets, calls, and folds.

- 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.[7] This concrete example highlights that collusion-free security is very subtle and depends, as to be expected, on the precise details of the game's specification.

High-Level View of Our Construction

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

As already mentioned, collusion-free protocols for non-trivial 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.

[6]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.

[7]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.

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) [8] 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,

[8] The original definition and construction of uniZK proofs is presented in [18]. However, for our application, a weaker version (requiring weaker assumptions) suffices.

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 cryptographic primitives, such a small amount of information about USK_i is provably useless for any adversary to violate our collusion-free definition.

Chapter 2

Defining Collusion-Free Protocols

We follow the original paradigm of [10]: 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 are that (1) our ideal adversaries do not communicate during an ideal execution —to capture their independence— and (2) an efficient function f "reconciles" their separate and contrasting views after the game —to guarantee *forever* that honest player privacy is maintained. More specifically, the reconciling function f combines the *separate* outputs of the our ideal adversaries, and produces a single output that is indistinguishable from the *joint* views of the real adversaries. This ensures that whatever malicious players can learn by comparing notes after a real execution, they can also learn after an ideal execution. This is the best we can hope for. Of course, it would be preferable if the malicious players, separated during the game, remain separated forever after, but it is not realistic to expect that real players live forever in a Faraday cage!

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). [1] We tightly couple a real execution to its corresponding ideal one by requiring that all information about 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 ideal state, we require that player's view of the message traffic can be *efficiently* mapped into that player's ideal state.

[1] Our definition generalizes to cases where multiple ideal players take actions at the same time, but this requires our protocols to make frequent use of simultaneous broadcast channels which we find undesirable.

2.1 Preliminaries

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

1. Σ, a finite set of possible game states,
2. X_i^j, a finite set of possible actions for player i for each stage j,
3. Y_i^j, a finite set of possible outputs for player i for each stage j, and
4. TURN : $[1...s] \rightarrow [1 \ldots n]$, a function which maps each stage to the player who must take an action during that stage.
5. $g^1, \ldots, g^s$, 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, a honest human player, H_i, is a sequence of probabilistic (strategy) functions, $\{\mathrm{H}_i^1, \ldots, \mathrm{H}_i^s\}$[2] and a malicious player, A_i, is an efficient interactive Turing machine which accepts a unary input (and as usual retains state between stages). Such 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, σ^j, is a string encoding all actions taken by all players prior to stage j, all information sent to the players prior to round j and any private randomness used by the stage functions. [3] The local state of a player i, σ_i^j consists of the all of the actions taken by i, and all of the outputs received from the trusted party prior to round j. The local state of a set of players C, σ_C^j, consists of the concatenation of σ_c^j for all $c \in C$. Letting Σ_i^j be the set of all possible local states for player i at the beginning of stage j, then each 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 x_i^j in stage j, then every player k receives x_i^j as part of his private output y_k^j at the end of stage j.

IDEAL EXECUTIONS. Let C be a subset of $[1, n]$. In an ideal execution of G with security parameter 1^K and trusted party T_G, the players are partitioned into a set of ideal adversaries, $\{\mathrm{A}_c : c \in C\}$, and a set of human players, $\{\mathrm{H}_i : i \notin C\}$.

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

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

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. Also prior to the first stage, the ideal adversaries may exchange an arbitrary number of private messages.

For stage j, suppose the trusted party has global state σ^j and let $i = \text{TURN}(j)$. If $i \notin C$, then the honest player function $\text{H}_i^j(\sigma_i^j)$ is invoked on input σ_i^j to compute an action x_i^j which is sent to T_G. If $i \in C$, then the ideal adversary A_c is given unary input 1^K and (running from his previous state) computes an action x_i^j which is delivered to T_G. The trusted party then computes the partial information vector $(\sigma^{j+1}, y_1^j, \ldots, y_n^j) = g^j(\sigma^j, x_i^j)$. The players receive their partial information, in lexicographic order, by means of the following n-step process (per player): for the ith player, T_G queries all the players in order to determine if player i should receive his partial information. If no player objects, T_G delivers y_i^j to player i. Else, if a player objects, then 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 $e \leftarrow \langle\ (\text{H}_i : i \notin C) \overset{1^k}{\longleftrightarrow} (\text{A}_c : c \in C)\ \rangle$ to denote that e is a random ideal execution of G with security parameter 1^K, ideal adversaries $\{\text{A}_c : c \in C\}$, and human players $\{\text{H}_i : i \notin C\}$. If e is an ideal execution, the auxiliary output of A_c is denoted $\text{OUT}_c(e)$. When indexed by a set C, $\text{OUT}_C(e)$ consists of the concatenation of $\text{OUT}_c(e)$ for $c \in C$.

PROTOCOLS. An r-round, n-player protocol Π for a game G is defined by (1) a set of n efficient oracle ITMs (Interactive Turing Machines), $\{T_i, \ldots, T_n\}$, each capable of making only s oracle calls, (2) r communication channels, $\{B_1, \ldots, B_r\}$, the last of which are broadcast channels, and (3) a (not necessarily efficient) extraction function EXT described below as well as 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 Π with human players $\text{H}_1, \ldots, \text{H}_n$ 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 $T_i^{\text{H}_i}$ otherwise. By $T_i^{\text{H}_i}$, we mean the interpreter T_i runs with oracle access to human player H_i such that the jth oracle call of T_i is answered by the probabilistic function H_i^j.

Prior to round 1, all parties are given a security parameter 1^K as input. Also prior to round 1, all malicious players may exchange may exchange an arbitrary number of private messages. For each round t of Π, all communication occurs via channel B_t, for honest and malicious players alike.

As before, we use the notation $e \leftarrow \langle\, (T_i^{\mathrm{H}_i} : i \notin C) \overset{1^k}{\longleftrightarrow} (A_c : c \in C) \,\rangle$ to denote that e is a random execution of Π with human players $\{\mathrm{H}_i : i \notin C\}$ and adversarial algorithms $\{A_c : c \in C\}$.

The message traffic in a real execution e is denoted $\textsc{traffic}(e)$ and consists of all messages sent during an of execution of Π. Similarly, $\textsc{view}_i(e)$ denotes the view of a 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, $\textsc{view}_C(e)$ is the concatenation of $\textsc{view}_c(e)$ for $c \in C$. Finally, the (possibly exponential time) function $\textsc{ext} : \textsc{traffic} \to \Sigma$ maps the traffic to a global game state in G and the efficient function $\textsc{ext}_i : \textsc{view}_i(e) \to \Sigma_i$ maps a player's view to a human player state.

2.2 Definition

Definition 1 (Collusion-free Protocol) *A protocol* Π *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* $\{\mathrm{A}_c : c \in C\}$ *such that for any set of honest human players* $\{\mathrm{H}_i : i \notin C\}$*, for all* $c \in C$*, there exists an efficient function* f *so that the following pair of ensembles are computationally indistinguishable*

$$\left\{\; e \leftarrow \langle\, (T_i^{\mathrm{H}_i} : i \notin C) \overset{1^k}{\longleftrightarrow} (A_c : c \in C) \,\rangle : (\textsc{view}_C(e), \textsc{ext}_c(\textsc{view}_c(e)), \textsc{ext}(\textsc{traffic}(e))) \;\right\}_k$$

$$\left\{\; e \leftarrow \langle\, (\mathrm{H}_i : i \notin C) \overset{1^k}{\longleftrightarrow} (\mathrm{A}_c : c \in C) \,\rangle : (f(\textsc{out}_C(e)), \sigma_c^s(e), \sigma^s(e)) \;\right\}_k$$

Remarks

- 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 $(\text{OUT}_C(e), \ldots)$ be indistinguishable from $(\text{VIEW}_C(e_I), \ldots)$. Let us informally explain why such an optimistic definition is impossible to achieve.

Such an optimistic definition would imply that for each $c \in C$, 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! Therefore, the optimistic definition would thus obligate each A_i to generate the exactly same broadcast traffic. However, this broadcast traffic together with player j's private view allows player $j \in C$ to deduce his private outputs. Thus, A_i must generate broadcast traffic that properly encodes j's private outputs without knowledge of these private outcomes —A task which is information-theoretically 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.

- We follow the definition of Dodis and Micali [9] and require that the game state of G is embedded in any real execution and could be extracted if one had unlimited computational resources. (The state of G is analogous to the effective inputs and outputs of the trusted party in the definition of Dodis and Micali.) An alternative approach to defining secure computation is to specify that the each honest player i outputs σ_i (in both the ideal and real executions) and that the in the definition the (extracted) state of G is replaced by the outputs of the honest players (see for example, [4]).

  Such an approach leads to a weaker (more permissive) notion of protocol security. In particular, it allows a protocol to be secure if the inputs (or outputs) of a player exist only in that player's mind and are not actually embedded in the traffic. (For example, perhaps a player i's output is a function of i's private coin tosses as well as the traffic of the protocol.) Which of these two approaches yields a more "natural" notion of protocol security is a philosophical question beyond the scope of this thesis.

  Since our main result is a positive one, we prefer to use the stronger (less permissive)

notion of security. (Indeed, since our construction achieves the the stronger notion we might as well claim the strongest result possible.) However, it is worth noting that all of the impossibility results in Chapter 5 continue to hold even if one adopts the weaker approach to protocol security.

- We think it is quite natural to model a setting in which malicious players can communicate before and after a protocol execution and are only isolated from each other during the execution. However, one can perhaps imagine settings in which malicious players are somehow prevented from coordinating their strategies before an execution or pooling their information after an execution.

  It is easy to formulate a definition which captures real and ideal adversaries cannot communicate after a protocol execution. To do so, in the above definition, simply replace $f(out_C(e))$ with $out_c(e)$ and replace $\text{VIEW}_C(e)$ with $\text{VIEW}_c(e)$. It is easy to see that the impossibility results from Section 5 still hold with respect to this modified definition and that our protocol presented in Section 4.1 also satisfies this modified definition.

  However, it is not clear how to formulate a definition which both (1) captures that real and ideal adversaries cannot communicate before an execution; and (2) is achievable for some class of non-trivial games using only broadcast and physical envelopes. Indeed, the ability of the ideal adversaries to coordinate their strategies is essential to our construction, and we do not see how this coordinate can be removed —even in the case where the real adversaries are prevented from communicating before hand.

- It is explicit in our ideal execution th any player can abort the execution at any time. This is necessary if we wish to tolerate an arbitrarily large coalition of malicious players. As in other secure computation settings, if we were willing to assume an honest majority, then we could remove the ability to abort from our ideal executions. In such a case, we would need to modify our protocol (in Section 4.1) to use a threshold secret sharing scheme in place of an XOR secret sharing scheme.[4]

[4]See [10] or [11] for more information on secure multi-party computation with an honest majority.

Chapter 3

Tools for Our Construction

In this paper, we use a number of standard cryptographic tools. For more information on Public-Key Encryption see [15]. For more information on Perfectly Binding Commitments see [11]. For more information on Zero-Knowledge proofs see [14, 13]. Below, we describe in some detail the classic GMW Protocol for Secure Multi-party Computation and our novel bounded uniZK proofs.

3.1 A Classic Tool: 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.[1] 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

[1] An honest-but-curious player sends every message according to the prescribed protocol, however honest-but-curious players may share information with one another and attempt to deduce information about other players' inputs and outputs.

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 $\hat{M}_j$. The GMW' protocol guarantees that after any honest execution, each player is able to compute an output y_i such that (1) the n-tuple $y_1, \ldots, y_n$ is a (possibly random) output of F on inputs $x_1, \ldots, x_n$ and (2) for any subset $C \subset \{1..n\}$, anything that can be efficiently computed from the transcripts of machines $\{T_c : c \in C\}$ can be efficiently computed from the input-output pairs $\{(x_c, y_c) : c \in C\}$.

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[2] 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

[2]When we say that a machine M_i "provides a zero-knowledge proof" of some NP statement $x \in L$, we mean that for each other machine M_j, machine M_i acts as the prover in a zero-knowledge proof that $x \in L$ in which M_j acts as the verifier and all messages in the proof are sent over the broadcast channel. If an honest verifier would reject *any* of these ZK proofs then *every* machine M_j aborts.

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.

GMW SIMULATOR Let C be a static set of adversaries. The GMW" simulator, $S_C(\{A_c | c \in C\})$, works for any set of real execution adversaries, A_c for $c \in C$, and does so in three stages: first, the simulator extracts the private input of the malicious players, it then feeds these private inputs to an Oracle for the functionality being computed and receives the corresponding outputs for the set of bad players, and finally produces fake transcripts and simulated proofs which induce those outputs. Informally, these three stages work as follows:

1. Pick random tapes, $\lambda_1, \ldots, \lambda_n$ for each of the n parties in the protocol. Whenever party i requires random bits, use λ_i.

2. Begin running an execution of the GMW protocol with algorithms A_c for $c \in C$ in which all honest party messages are generated as specified below. Whenever the protocol calls for an adversary to generate a message, run the corresponding machine A_c to generate the message. Feed all messages into the A_c algorithms as they would be fed in a real execution of GMW".

3. Complete Step 1 of the GMW" protocol for each of the honest parties using random tapes λ_i for all $i \notin C$.

4. Use the ability to rewind each of the adversary machines in order to extract secret keys SK_c during step 1. At this point, the simulator knows all of the secret encryption keys for all of the parties.

5. Complete Step 2 of GMW". Use the secret key of each of the bad players extracted in the previous step to determine R_c for each $c \; inC$. Complete Step 3 of GMW" by generating each honest message as specified in GMW".

6. Begin executing Step 4 of GMW". Recall that the first step of the GMW' protocol is for each machine to produce an XOR sharing of its input. When generating shares for the honest players, the simulator produces a random sharing of the input 0. For each party $c \in C$, use use knowledge of the SK_c and R_c in order to extract x_c for all $c \in C$.

7. Query the Oracle with the x_c values that were extracted in the previous step. Receive the corresponding function values for the bad players, y_c for $c \in C$.

8. Continue simulating the GMW" protocol. When a message for an honest player must be produced, send an encryption of a random message, along with a simulated ZK proof that the random string was correctly computed as in GMW". Verify each of the adversary messages as the honest player would. If any message from the set of bad players does not verify, signal an abort.

9. Once the execution reaches Step 5 of GMW", the simulator knows all of the shares of the output values that each of the adversary machines have computed. The simulator then produces fake messages and proofs for the honest party such that the final value computed by each of the A_c machines is y_c.

3.2 A Novel Tool: Bounded Unique Zero-Knowledge Proofs

We first provide a definition of Bounded-Theorem uniZK. As with the standard definition of uniZK presented in [18], our bounded-theorem uniZK definition guarantees —in addition to the standard properties of a zero-knowledge proof system— that with overwhelming probability (over the choice of public proving keys) the honest prover algorithm is a bijection between witnesses for the theorem and proofs the verifier will accept. In particular, this implies that for any theorem with a unique witness, there is only a single proof string that will cause the verifier to accept.[3] However, our bounded-theorem uniZK definition is weaker

[3]In our collusion-free protocol presented in Section 4.1 we shall use uniZK proofs only for theorems that have a single witness.

than the standard uniZK definition not only because the number of theorems being proved is bounded, but also because a dishonest prover is forced to use a correctly generated public and secret key pair.[4] However, this weaker formulation suffices for our purposes because we generate the public and secret keys for all players using a secure function evaluation and thus even a dishonest prover cannot cause his keys to be generated incorrectly. We then show that unlike the standard uniZK definition, this definition can be easily achieved using any one-way permutation.[5]

Let L be an NP language, and R_L be its corresponding, polynomial-time relation. We say that a sequence of pairs of strings, $(x_1, w_1), (x_2, w_2), \ldots, (x_\rho, w_\rho)$, is a *p-bounded theorem-witness sequence for* L for some polynomial p if each $x_i \in L$ and $w_i \in R_L(x_i)$ and the total length of all the x_i's is less than $p(k)$.

Definition 2 *A triple of efficient algorithms, (G, P, V), where P is deterministic, is a* bounded theorem uniZK proof system *for an NP-language L if there exists a positive constant c and a negligible function μ such that the following properties are satisfied:*

COMPLETENESS: $\forall$ *p-bounded theorem-witness sequences* $(x_1, w_1), (x_2, w_2), \ldots, (x_\rho, w_\rho)$ *for* L, *and security parameters* $\forall k > 2$

$$\Pr\left[\begin{array}{l}(\mathrm{UPK}, \mathrm{USK}) \leftarrow G(1^k, \rho);\ \pi_1 = P(x_1, w_1, \mathrm{USK}, 1); \\ \pi_2 = P(x_2, w_2, \mathrm{USK}, 2);\ \ldots;\ \pi_\rho = P(x_\rho, w_\rho, \mathrm{USK}, \rho) :\ \bigwedge_i V(x_i, \mathrm{UPK}, \pi_i, i) = 1\end{array}\right] = 1$$

SOUNDNESS: $\forall k > 2$ *and* $\forall$ *algorithms* P^*,

$$Pr\left[(\mathrm{UPK}, \mathrm{USK}) \leftarrow G(1^k, \rho); (x^*, \pi^*, i) \leftarrow P^*(\mathrm{UPK}, \mathrm{USK})\ :\right.$$
$$\left. x^* \notin L \wedge\ V(x^*, \mathrm{UPK}, \pi^*, i) = 1\right] < \mu(k)$$

ZERO-KNOWLEDGENESS: $\exists$ *an efficient algorithm S, such that for all p-bounded theorem-witness sequences* $(x_1, w_1), (x_2, w_2), \ldots, (x_\rho, w_\rho)$ *for L, the following two ensembles are com-*

[4] In the stronger definition of [18] the soundness and zero-knowledgeness of a proof must hold no matter how a dishonest prover maliciously chooses his public key. They achieve this stronger guarantee by including a common random string which the prover uses to certify that his chosen key is "valid".

[5] The construction of [18] requires the quadratic residuosity assumption. No constructions of standard uniZK proof systems under any other assumption are currently known.

putationally indistinguishable:

$$\left\{ \begin{array}{l} (\text{UPK}, \text{USK}) \leftarrow G(1^k, \rho);\ \pi_1 = P(x_1, w_1, \text{USK}, 1); \\ \pi_2 = P(x_2, w_2, \text{USK}, 2);\ \ldots;\ \pi_\rho = P(x_\rho, w_\rho, \text{USK}, \rho)\ :\ (\text{UPK}, \pi_1, \pi_2, \ldots, \pi_\rho) \end{array} \right\}_k$$

$$\left\{ \begin{array}{l} (\text{UPK}', \text{USK}') \leftarrow S(1^k); \pi'_1 \leftarrow S(\text{USK}', x_1, 1); \\ \pi'_2 \leftarrow S(\text{USK}', x_2, 2);\ \ldots;\ \pi'_\rho \leftarrow S(\text{USK}', x_\rho, \rho) : (\text{UPK}', \pi'_1, \pi'_2, \ldots, \pi'_\rho) \end{array} \right\}_k$$

UNIQUENESS: $\exists$ *an efficient deterministic algorithm* P^{-1} *such that* $\forall x \in L$, $\forall i > 0$, *with probability greater than* $1 - \mu(k)$ *(over the random choice of* (UPK, USK) *according to* $G(1^k, \rho)$*)* $P(x, \cdot, \text{USK}, i)$ *is a bijection between* W_x *and* $\Pi^i_{\text{UPK}}(x)$ *and* $P^{-1}(x, \cdot, \text{USK}, i)$ *is a bijection between* $\Pi^i_{\text{UPK}}(x)$ *and* W_x*, where* $W_x = \{w : w \in R_L(x)\}$ *and* $\Pi^i_{\text{UPK}}(x, \sigma) = \{\pi : V(x, \sigma, \text{UPK}, \pi, i) = 1\}$.

Theorem 1 (uniZK) *The existence of one-way permutations implies a p-bounded theorem* uniZK *proof system for any NP-language L and for any polynomial p*().

Proof: Our construction of a bounded theorem uniZK system is based heavily on the protocol of DMP protocol[8] which allows a prover, after an interactive pre-processing stage, to non-interactively prove a set of theorems of bounded length.

In particular, we define G on input 1^k, to perform an honest execution of the DMP pre-processing protocol and output as a public key, the honest verifier's view of this execution, and output as a secret key, the honest prover's view of this execution. Additionally we take P to be the DMP theorem proving algorithm and take V to be the DMP verifier algorithm. Completeness, Soundness and Zero-Knowledgeness follow directly from the correctness of the DMP protocol. Uniqueness follows from the observation that an honest execution of pre-processing induces a bijection between NP witnesses that $x \in L$ verifier acceptable proofs of $x \in L$ (and this bijection is easily computed in both directions by anyone who knows the coin tosses used by the honest prover during pre-processing).

Chapter 4

Constructing Collusion-Free Protocols

4.1 Construction

In this section, we present our Collusion-free protocol. The corresponding security proofs, which complete Theorem 1, can be found in the following section.

Our protocol is based on the original GMW [10] 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. Therefore, we provide an overview of the important pieces of the GMW protocol in Section 3.1. (A more thorough explanation of the GMW protocol can be found in [11].)

As noted in the Introduction, our protocol, P, for playing any finite game G consists of two subprotocols, a *pre-processing subprotocol*, P_1, immediately followed by a *computation subprotocol*, P_2. The pre-processing phase computes a function PRE in such a way that no player has knowledge of another player's private output, even if players maliciously collude with one another. 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 [18]. 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. In Section 3.2 we give a bounded

uniZK proof system based on one-way permutations.[1]

Physical Envelopes

As we show in Section 5.1, some type of physical channel assumption is necessary to achieve a collusion-free protocol for general games. Our protocol assumes the existence of physical envelopes that support the operations SEND(R, m) and RECEIVE(S) which satisfy the following properties: (1) *Binding* —when a receiver, R, calls RECEIVE(S), he learns the values m for all previous calls to SEND(R, m) made by sender S (2) *Hiding* —when a sender calls SEND(R, m), no one gains any information about m and when receiver calls RECEIVE(S) no one besides R gains any information about m (3) *Public* —when a sender calls SEND(R, m) everyone learns the string "(SEND, S, R)" and when a receiver calls RECEIVE(S), everyone learns the string "(RECEIVE, S, R)." We think of 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 RECEIVE(S) corresponding to the action where the receiver publicly opens all envelopes from S and privately reads their contents. [2]

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 stage 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 *Gen* be the generator for a public-key encryption scheme, Gen_{uzk} be the generator for a p-bounded-theorem uniZK system and COM$(string, coins)$ be the commitment algorithm for a perfectly binding commitment scheme. The function PRE then operates by

:

[1]The original uniZK paper[18] 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.

[2]Note that a single invocation of a channel allowing simultaneous delivery of private messages would suffice for our protocol, but we prefer to use physical envelopes to emphasize that fact that simultaneity is not required.

1. Running $Gen(1^k)$ n times to obtain matching public and secret encryption keys: $(\text{PK}_1, \text{SK}_1), \ldots, (\text{PK}_n, \text{SK}_n)$.
2. Running $Gen_{uzk}(1^k, p(k))$ n times to obtain matching public and secret uniZK keys: $(\text{UPK}_1, \text{USK}_1), \ldots, (\text{UPK}_n, \text{USK}_n)$.
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, $\text{COM}(R_1, r_1)$, ..., $\text{COM}(R_n, r_n)$, $\text{COM}(R'_1, r'_n)$, ..., $\text{COM}(R'_n, r'_n)$, and $\text{COM}(\text{SK}_1|\text{USK}_1, v_1)$, ..., $\text{COM}(\text{SK}_n|\text{USK}_n, v_n)$.

The common output *Common* is then the 6-tuple consisting of (1) The n-vector of public encryption keys $(\text{PK}_1, \ldots, \text{PK}_n)$ (2) The n-vector of public uniZK keys $(\text{UPK}_1, \ldots, \text{UPK}_n)$ (3) The n-vector of committed random tapes $(\text{COM}(R_1, r_1), \ldots, \text{COM}(R_n, r_n))$ (4) The n-vector of committed random tapes $(\text{COM}(R'_1, r'_1), \ldots, \text{COM}(R'_n, r'_n))$ (5) The n-vector of common reference strings $(\tau_1, \ldots, \tau_n)$ and (6) The n-vector of committed secret keys $\text{COM}(\text{SK}_1|\text{USK}_1, v_1)$, ..., $\text{COM}(\text{SK}_n|\text{USK}_n, v_n)$. The private output o_i is the 7-tuple: $(\text{SK}_i, \text{USK}_i, R_i, r_i, R'_i, R'_i, v_i)$.

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 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, unlike in GMW, 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 $\text{SEND}(j, m_j)$ 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 $\text{RECEIVE}(j)$ 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 this private output o_i. Each

player then verifies the secret keys in o_i correspond to the public keys which are part of *Common* by opening $\text{COM}(\text{SK}_i|\text{USK}_i, v_i)$ using v_i and comparing the values. Similarly, each player checks that the secret coins in o_i correspond to the public commitments in *Common*. 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 (whose size is independent of the security parameter k).
- The players do not perform Step 1 of GMW, but instead use the keys $(\text{PK}_i, \text{SK}_i)$ computed during P_1.
- The players do not perform Step 2 of GMW, but instead use the committed random tapes $\text{COM}(R_i)$ 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 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 $\text{COM}(R'_i, r'_i)$."

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 the 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 by player H_i and all of the partial information it receives.

2. Let $i = \text{TURN}(j)$. In order to emulate stage j of G, interpreter T_i queries its oracle H_i on input σ_i to obtain an action x_i^j (this is the same action that the human player sends

in an ideal execution). All interpreters then engage in $n+1$ UNI-GMW protocols to compute the following functions (we are essentially splitting the stage function g_j into $n+1$ parts):

- 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' private inputs are their own shares of σ^j.
- $\text{OUTPUT}_{i,j}$: 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 above.

  For each party i (in order from 1 to n), the interpreters run a UNI-GMW protocol on the inputs described above 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 to it.

  For all $j > 1$, the inputs to SHARE_j and $\text{OUTPUT}_(i,j)$ include shares of the global state produced as output from the UNI-GMW protocol computing SHARE_{j-1}. 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 stage.

4.2 Proof of Correctness

Here we describe how to map any set of corrupted players, C, who use adversarial algorithms $A = \{A_c\}, c \in C$, into a set of independent ideal adversaries, $\{\mathrm{A}_c | c \in C\}$ where each A_c corresponds to A_c. We also describe a reconciliation function f which takes the output from said ideal adversaries, and produces a transcript of an execution of P which is indistinguishable from a real execution.

At a high level, each of our ideal adversaries independently simulates an execution of P against adversary algorithms $\{A_c | c \in C\}$ as follows. During the simulation of P_1, the ideal adversary uses its rewinding ability in order to control the set of $(Common, o_i)$ outputs generated for each of the honest players in their own simulation. By controlling these keys, A_c can use the secret keys of the players in $C - \{c\}$ to generate messages for these bad players in each of the UNI-GMW protocols in P_2. Additionally, A_c can use the uniZK simulator to "fake" messages from the honest players in order to force particular outputs for player c.

At the end of the ideal execution, each ideal adversary A_i outputs the ideal state of player i consisting of all messages that A_i sent and received from the trusted party during the ideal execution. The key goal of our simulation is to ensure that the ideal state produced during such an ideal execution is statisticly indistinguishable from the ideal state extracted from a real execution. We then construct a reconciliation function f that takes as input the ideal states produced by A_i and outputs simulated views for all of the ideal players. To implement this high-level approach, we must overcome the following three challenges:

Simulating Dishonest Player Messages: The primary challenge that differentiates our collusion-free simulation from other protocol simulations is that we must simulate the view of a malicious adversary interacting with other *independent* malicious adversaries.[3] In particular, this means that A_i must simulate messages messages sent by A_j. Naively, one might suggest that A_i generate A_j's messages by simply running A_j. However, this is not possible because A_i does not know the messages that player j received from the trusted party.

It is the uniqueness of our uniZK proofs and the fact that G is a public-action game

[3]Traditionally, in protocol simulations, all players not controlled by the malicious adversary are honest. Such a traditional setting is easier since the honest players faithfully follow the protocol instructions —which are, of course, carefully designed so as to make honest players easy to simulate.

which enables us to overcome this challenge. Whenever player j chooses an action a in the game, that action is transmitted to A_i[4]. The uniqueness of our proofs then guarantees that there is only one possible way for player j to encode action a (as a sequence of encrypted shares) and A_i —knowing the secret keys of all malicious players— can compute this proper encoding of a. Similarly, whenever player j must send a message that does not encode his choice of action, he has, by uniqueness, only possible message to send and A_i can compute this message. Thus, by severely limiting the choices available to each player, we enable the ideal adversary A_i, who does not have enough information to run the code of the real adversaries, to nonetheless generate the messages sent by the other malicious players.

Simulating Honest Player Messages: The second challenge is to simulate the messages sent by the honest parties. What makes this challenging in our setting is that the messages sent by the honest players essentially encode the outputs sent by the trusted party to the malicious players. Indeed, a standard simulation technique is to "fake" the messages sent by the honest parties to ensure that the output encoded in these messages is the correct output obtained from the trusted party. The difficulty in our setting is that the honest player messages encoding player j's output are necessarily different in A_i's simulated execution than they are in A_j's simulated execution —because A_i doesn't know player j's output! This can be problematic if A_j were, for example, to hash all honest player messages he's seen in order to select some future action. In essence, the problem is that later actions chosen by A_j can provide A_i with additional information about the honest-player messages that encoded earlier outputs for player j.

To overcome this challenge, A_i must update the view that he presents to A_i at each stage based on the actions chosen by the other malicious players. In a real execution, for any global state σ^k at stage k, A_i sees a random transcript consistent with σ^k. Therefore, to obtain the same distribution of player i actions in an ideal execution, A_i should endeavor to present A_i with a random transcript consistent with σ^k. This is clearly impossible, but fortunately —due to the privacy properties of our cryptographic primitives— A_i cannot distinguish a random transcript consistent with σ^k from a random transcript consistent with σ_i^k. Thus at each stage k, A_i randomly samples simulated transcripts until it finds one consistent with $state_i^k$. This sampling can be performed

[4]Recall that our protocol works only for public-action games.

in expected polynomial time since the number of possible states is constant.[5] Such sampling is also used by the reconciliation function f to generate a simulated transcript which is consistent with σ_C^k.

4.2.1 Ideal Adversaries

In the ensuing discussion, we denote by S_C, the GMW simulator for malicious players C as defined in Section 3.1. This simulator expects black-box access to the algorithms A_c for $c \in C$ and produces a transcript of the interaction between honest players and the coalition C.

The Function PRE′

The ideal adversaries for any set of corrupted parties compute a function PRE'_C, which is very similar to the function PRE, except that it uses a uniZK simulator in order to generate the public and secret uniZK keys and common random strings for the honest players. The function is defined as follows.

On input 1^k,

1. Run step 1 of the function PRE.

2. In place of step 2, run $S_{uzk}(1^k)$, the uniZK simulator, $n-|C|$ times to obtain matching public and private uniZK keys $(\text{UPK}'_i, \text{USK}'_i)$ and simulated random strings τ'_i for each $i \notin C$. Run the normal uniZK generator, $G_{uzk}(1^k)$ $|C|$ times to generate uniZK public and private keys, $(\text{USK}_c, \text{UPK}_c)$ for each $c \in C$.

3. Run steps 3 and 4 of the function PRE, except that instead of randomly choosing the τ_i strings for $i \notin C$, use τ'_i from the previous step. Generate all τ_c, $c \in C$ as in PRE.

These values are organized as specified in the PRE function into a common output $Common'$ and a private output o'_i.

[5]The difficulty in doing such sampling is that σ_i^k may occur with negligible probability and thus the sampling procedure may require more than polynomial time. We therefore achieve expected polynomial time sampling by using the techniques of Goldreich and Kahan [12]. That is, we approximate the probability that σ_i^k occurs and then terminate the execution if we fail to find a transcript consistent with σ_i^k within the anticipated number of trials. Although the work of Goldreich and Kahan only considers a setting with two possible states (success and abort), their techniques immediately generalize to settings with a constant number of states.

Producing transcripts for P_1 (pre-processing phase).

For this part, A_c makes black-box access to all bad player algorithms, $\{A_c | c \in C\}$ in order to run S_C.

1. **Share coins.** Before the protocol beings, the ideal adversaries agree upon a random tapes, λ_{pre}, λ_S and, for all $c \in C$, λ_c which they will use during the ideal execution. Recall that this is a legitimate step, since our ideal model allows adversaries to share arbitrary information before the game starts.

2. **Generate known keys.** Each A_c uses $\lambda_{pre'}$ to run the function PRE$'$ in order to generate outputs $(Common, o_i')$ for each player i.

3. **Start** P_1 Each A_c runs an independent copy of S_C, using random tape λ_S as the coins of the simulator and random tape λ_c as the coins for algorithm A_c (for each $c \in C$), until Step 9 of S_C, which corresponds to Step 5 of the GMW" protocol. A_c prints all the messages output by S_C and the adversaries A_c into its transcript.

   When S_C queries the oracle for the function values (during its own Step 7), each A_c supplies S_C with the strings, $(Common', o_c')$ (for all $c \in C$), generated in Step 2.

   At some point, S_C produces shares of $(Common, o_c)$ sent by the honest players to player c (where all shares sent to player c are encrypted in c's public key). It is understood that these shares can be split into the shares of the public output, $Common$, and the private output o_c. At this point A_c prints the portion of the shares corresponding to $Common$ into the transcript (along with the ZK proofs generated by S_C that these shares are correctly computed). The shares of o_c are used in the next step.

4. **Print traffic.** Finally, using the shares of o_c, A_c prints into the transcript envelope traffic as described in Step 3 of the pre-processing.

UNI-GMW Simulator

To simplify the description of our ideal adversaries below we describe the following chunk of code (which is executed repeatedly by the ideal adversary) as the "UNI-GMW Simulator" $\mathcal{S}$. (Note that this is not a simulator in the strict sense of any definition, but is simply a sub-routine that the ideal adversary uses to generate some transcripts.) Whenever, $\mathcal{S}$ is invoked, it inherits all of the state of the ideal adversary which includes (a) The output of

the PRE′ function, (b) the stage j (which allows it to determine which section of the tapes R and R' are being used), (c) the set of malicious players and their code, (d) the function being computed (either $share_j$ or $\text{OUTPUT}_{i,j}$, (e) the output, y_c for player c.

The "Simulator" $\mathcal{S}(A_c, y_c)$ works as follows:

1. First, the simulator must produce commitments to the initial state of the honest-but-curious machine for each player. For each $d \in C$, the simulator honestly generates this message based on R_d (using the section of R_d reserved for this instance of the protocol). The simulator produces correct uniZK proofs that these commitments are correctly computed. For each honest player $i \notin C$, the simulator generates a commitment to a random initial state for the honest-but-curious machine and simulates a uniZK proof that this commitment is correct. Use the black-box access to A_c to generate player c's commitment to the initial state.

2. As in Step 6 of the GMW simulator, produce messages corresponding to the XOR sharing of the player's inputs. For honest players, produce a random sharing of the input 0 and give a simulated uniZK proof that the sharing is correct. For player c, use black-box access to A_c to produce the sharing messages. If a player $d \in C$ must give an input other than his share of σ_j, the ideal adversary invoking $\mathcal{S}$ will supply messages to be inserted into the transcript on behalf of player d. When a player d's only input is his share of σ_j, the simulator uses secret key SK_d and R_d (both in o_d) to determine σ_j, and then honestly computes a new sharing of this same value using tape R_d (and of course gives a proper uniZK proof that this was done correctly).

3. Generate a simulated transcript of the remaining portion of the UNI-GMW execution. During the protocol, whenever a player $d \in C$ must generate a message, generate it honestly based on the messages received by player d (which can be read using SK_d) and the current committed state of d's (honest-but-curious) machine. Whenever an honest player must generate a message, generate an encryption of a random message (as done by the GMW simulator), and use the uniZK simulator to generate the proof of correctness. Use the black-box for player A_c to generate messages (and uniZK proofs) for player c. The simulator ensures that player c's output is y_c by generating fake shares (and corresponding proofs) for the honest players in the final round (as in Step 9).

Sampling Consistent Transcripts

When simulating stage j of game G, the ideal adversary A_c will have a sequence of actions $x_c^1, \ldots, x_c^{j-1}$ selected by player c, a sequence of outputs $y_c^1, \ldots, y_c^{j-1}$ received by player c and (since the game G has public actions) a sequence of actions $x_d^1, \ldots, x_d^{j-1}$ for each other malicious player $d \in C$. To ensure a proper distribution of stage j actions for player c, A_c must present A_c with a random (simulated) transcript of the first $j-1$ stages consistent with these actions and outputs.

To accomplish this, we adopt the techniques of Goldreich and Kahan [12].[6] Letting 1^K be the security parameter, we randomly sample K^2 genuine transcripts. This allows us to estimate, to within a constant factor, the probability that $x_c^1, \ldots, x_c^{j-1}$, and $x_d^1, \ldots, x_d^{j-1}$ for each $d \in C$, all simultaneously occur. Call this estimate α.[7]

The ideal adversary A_c now runs the UNI-GMW simulator $\mathcal{S}$ once for every UNI-GMW protocol executed during the first $j-1$ stages of P_2.[8] (Ideal adversary A_c always runs $\mathcal{S}$ with adversary A_c. Additionally, for each UNI-GMW protocol that produces a share of the global state, A_c picks that produced share at random and for the UNI-GMW protocol that produces the stage k output for player c, A_c runs the UNI-GMW simulator with output y_c^k.) Once he has produced such a simulated transcript, A_c checks the transcript for consistency. That is, for each $d \in C$ (including c), A_c checks that at each stage $k < j$, A_d does not abort and selects action x_d^k (whenever it is d's turn to act in stage k).

If A_c finds that the simulated transcript is not consistent with all of the given actions, then A_c generates a new simulated transcript (as described above) and continues until either it finds a consistent transcript or else it finds K^2/α inconsistent transcripts. In the latter case, A_c aborts the protocol and outputs the special symbol $\perp$.[9]

---

[6] Although the work of Goldreich and Kahan only considers a setting with two possible states (success and abort), their techniques immediately generalize to settings with a constant number of states.

[7] Of course, in order to sample genuine transcripts we would need to know the human players H_i, for all $i \notin C$. Since we don't know the human players, we can randomly select actions for the human players and, because the game has constant size, this only makes our estimate α worse by a constant factor.

[8] Namely, once before stage 1 to generate shares of the initial global state, and then $n+1$ times per stage —once to compute a sharing of the new global state and n times to deliver outputs to each of the n players.

[9] Note that the (as in [12]) the indistinguishability of simulated transcripts from genuine transcripts implies that A_c outputs $\perp$ with negligible probability and that the above procedure for randomly sampling a consistent transcript runs in expected polynomial time.

Producing transcripts for P_2 (Computation phase).

Recall that in this phase, A_c cannot make use of a GMW-style simulator because A_c does not know the outputs of any malicious player other than c. Notice that A_c makes repeated calls to the UNI-GMW simulator, $\mathcal{S}(A_c)$, and that $\mathcal{S}$ has access to all values computed by A_c.

Throughout this simulation, A_c uses string λ_c as the random coins for A_c and (when needed) uses λ_d as the random coins of A_d for all other $d \in C$. Additionally, for all $d \in C$, let x_d^j be the action selected by player d in stage j (and let x_d^j be empty if $d \neq \textsc{turn}(j)$), and let y_c^j be the output selected received from the trusted party in stage j. (Recall that since G is a game of public actions that A_c obtains x_d^j via y_c^j).

First A_c obtains transcript T_1 as follows. Select a random value ω^0 as player c's share of the initial global state. Run $\mathcal{S}(A_c, \omega^0)$ to generate a transcript T_1 for the protocol $\textsc{share}_0$ which computes shares of the initial share of the global state. (Note, that T_1 will be completely independent of the global state maintained by the trusted party.)

For each stage j, A_c will have previously computed a transcript T_j during the previous stage. Let $i = \textsc{turn}(j)$. If $i = c$, then A_c must send an action x_c^j to T_G. Select a random share ω^j and run $\mathcal{S}(A_c, \omega^j)$ for the function $share_j$. Use the transcript from this step (along with o_c) to extract player c's action x_c^j.[10] The ideal adversary now forwards x_c^j to T_G. If $i \neq c$ then A_c merely waits for the output y_c^j. Ideal adversary A_c must still determine whether to abort prior to the players receiving outputs in stage j. To do so A_c selects a random share ω^j and runs $\mathcal{S}(A_c, \omega^j)$ for the function $\textsc{share}_j$. If A_c aborts in this simulated execution, then A_c requests an abort prior to any player receiving stage j output.

Similarly, A_c must determine whether to abort prior to player x receiving output for each $x \leq c$. To do so, the ideal adversary runs $\mathcal{S}(A_c)$ for the function $\textsc{output}_{x,j}$. If A_c aborts in this simulated execution, then A_c requests an abort prior to player x receiving output in stage j.

Once A_c receives y_c^j, the ideal adversary learns x_i^j and now samples a transcript T_{j+1} that is consistent with $x_d^1, \ldots, x_d^j$, for all $d \in C$. Once all s stages have been completed, A_c prints his local state consisting of all actions and outputs sent or received from the trusted

[10] This is possible because the transcript contains a commitment to x_j^c using coins specified in the random tape R_c'. Since the random tape is known, A_c enumerate all possible commitments to the finitely many actions and determine which one was chosen by A_c.

party.

Generating Reconciled Transcripts.

The reconciliation function f takes in the local states produced by the ideal adversaries and randomly samples a transcript consistent with all of these local views. This process is quite similar to the consistent transcript sampling for ideal adversary A_c and uses the techniques of Goldreich and Kahan [12] (see description above).[11] However, unlike A_c, function f has access to the outputs of all players in C. Therefore, we make use of the fact that our protocol satisfies the traditional notion of protocol security against a single monolithic adversary.[12] Let A be the (monolithic) adversary who selects a random tape λ_d for each $d \in C$, and then computes messages from player d in protocol P by running algorithm A_d with random tape λ_d. Let SIM_A be the traditional simulator for adversary A and protocol P.

On input the local states σ_d (for each $d \in C$), the function f obtains simulated transcripts by running SIM_A, and whenever SIM_A requires an output y_d^j from the trusted party, f provide the simulation with the appropriate y_d^j output by A_d at the end of the ideal execution. The function f checks consistency of a transcript by extracting the local state σ'_d of each player $d \in C$. If $\sigma_d = \sigma'_d$ for each $d \in C$ then the transcript is consistent with the outputs of the ideal adversaries and f outputs the transcript. Otherwise, f repeatedly samples transcripts until it either finds one that is consistent or obtains K^2/α inconsistent transcripts —where 1^K is the security parameter and α is an estimate of the probability that a genuine transcript is consistent with $\{\sigma_d : d \in C\}$. In the latter case, function f outputs the special symbol $\perp$.[13]

4.2.2 Proof Sketch of Collusion-Freeness

The key component of the proof is showing that the global state produced by executing the ideal adversaries $\{\mathrm{A}_c : c \in C\}$ along with any set of human players $\{\mathrm{H}_i : i \notin C\}$ yields a

[11] Here it is again essentially that G is finite and thus that there are only a constant number of possible global states of the game. Otherwise, we would not be able to invoke the techniques of Goldreich and Kahan to obtaining an expected polynomial-time sampling procedure.

[12] Since our protocol closely follows the structures of the GMW protocol, that our protocol satisfies traditional notions of security is an immediate corollary of [10].

[13] Note that in the case that some A_c outputs $\perp$, then f has no hope of finding a consistent transcript and will also output $\perp$. However, this occurs with only negligible probability.

global game state that is computationally indistinguishable[14] the extracted global states of a real execution with $\{A_c : c \in C\}$ and $\{\mathrm{H}_i : i \notin C\}$. In essence, this result, whose proof is outlined belowed, shows that *during* the protocol, the real adversaries have no more power to affect the global state than they would in an ideal execution.

Given this result, the collusion-freeness of the protocol follows immediately from the fact that f is a properly distributed transcripts in expected polynomial time. This in turn follows from (1) the fact that our protocol is secure in the traditional sense against monolithic adversaries and (2) the techniques of Goldreich and Kahan.

Lemma 1 *For each malicious player $c \in C$ and for all possible human players corresponding to the players $i \notin C$, the following ensembles are indistinguishable.*

$$\left\{ e \leftarrow \langle\, (T_i^{\mathrm{H}_i} : i \notin C) \overset{1^k}{\longleftrightarrow} (A_c : c \in C) \,\rangle : \mathrm{EXT}(\mathrm{TRAFFIC}(e)) \right\}_k$$

$$\left\{ e \leftarrow \langle\, (\mathrm{H}_i : i \notin C) \overset{1^k}{\longleftrightarrow} (\mathrm{A}_c : c \in C) \,\rangle : \sigma^s(e) \right\}_k$$

PROOF SKETCH OF LEMMA 1

We refer to the top ensemble as REAL, and the bottom ensemble as IDEAL. First, for clarity, we define $REAL^{good}$ to be the distribution REAL conditioned on the event that no adversary A_d in the execution produces an acceptable uniZK proof of a false theorem. Similarly, we define $IDEAL^{good}$ to be the distribution IDEAL conditioned on the fact that for all $d \in C$, the copy of A_d being run inside A_d does not produce an acceptable uniZK proof of a false theorem. Since false theorems can only be proved with negligible probability, it is clear that $REAL \overset{c}{\equiv} REAL^{good}$ and $IDEAL \overset{c}{\equiv} IDEAL^{good}$.[15]

At a high level our technique is to bridge the gap between the $IDEAL^{good}$ and $REAL^{good}$ executions with a series of four hybrid executions.

First, we consider a Hybrid A where instead of A_c having secret keys for the players in C and honestly generating messages on their behalf, A_c only has knowledge of the secret keys of player c, and must instead rely on an external party to honestly generate message

[14] Since the number of possible global states is constant, in this case the notion of computational indistinguishability coincides with the notion of the statistical indistinguishability.

[15] Of course, this assumes that the uniZK keys for the players in C are generated properly. In an ideal execution, it is clear from the code of the ideal adversary that the uniZK are properly generated. However, to prove this formally in a real execution requires using the security of the GMW protocol used to generate the uniZK keys during pre-processing.

for the players in $C - \{c\}$. Additionally, A_c relies on this external party to check the consistency of the simulated transcripts it generates. This hybrid is indistinguishable from $IDEAL^{good}$ because the perfect secrecy of the envelopes used in pre-processing ensures that no information about the secret keys of player $d \in C$ can be deduced from player c's view (and by design, the messages for the players in $C - \{c\}$ are generated in exactly the same manner in both $IDEAL^{good}$ and Hybrid A).

In a Hybrid A execution: (1) the ideal adversary only knows the secret keys of player c (2) the committed random strings in the simulated UNI-GMW transcripts are completely independent from the global state and the outputs received by player c and (3) some of the players other than c are sending "fake" messages to c to force c to receive certain outputs. This setting is quite similar to a traditional ideal GMW execution with only a single malicious player c. This motivates a second Hybrid B in which A_c's UNI-GMW simulator "magically" receives committed strings that correspond to the true global state and all honest player messages are honestly generated in a fashion consistent with this global state. (However, the proofs that these messages are correct are still simulated by A_c.) Hybrid B is indistinguishable from the Hybrid A due to the correctness of the traditional GMW protocol (which tells us that "fake" honest player messages and committed strings that are independent of the global state are indistinguishable from real honest player messages and committed string consistent with the global state).

In a Hybrid B execution, the ideal adversary still generates simulated uniZK keys for all of the honest parties and simulates uniZK proofs that the honest player messages were correct. We next introduce a Hybrid C in which the ideal adversary no longer generates simulated uniZK keys for the honest players but instead "magically" receives correct uniZK keys for the honest players. In addition to "magically receiving correctly generated honest player messages," the ideal adversary also "magically" receives proper uniZK proofs that the honest player messages are correctly generated. This hybrid is indistinguishable from the Hybrid B because of the zero-knowledge property of the uniZK proof system.

In a Hybrid C execution we observe that (1) each of the ideal adversaries receives (the same) committed tapes consistent with the global state and (2) each of the ideal adversaries receives (the same) correctly computed honest player messages consistent with the global state. Additionally, we observe that the consistency of A_c's transcripts ensures that the actions taken in A_c's transcripts to correspond perfectly with the actions taken in A_d's

transcripts (for all $d \in C$). For these reasons, and because no copy of A_d proves a false theorem, the theorems being proven in A_c's transcripts correspond *perfectly* to the theorems being proven in A_d's transcripts. However, note that in A_c's transcript, A_c produces uniZK proofs that messages sent by player c are correct, while in A_d's transcript these uniZK proofs that player c's messages are correct are produced by some external party.

We now consider a Hybrid D in which each A_d magically receives the uniZK proofs (and corresponding messages) generated by A_c's copy of A_c. This hybrid is indistinguishable from Hybrid C since the uniqueness of the uniZK proof system (along with the fact that all of the theorems that are proven have only a single witness) ensures that A_c must be generating the same uniZK proofs as the external party used in Hybrid C.[16]

We now observe that during the computation-phase of a Hybrid D execution, A_c's transcript contains committed tapes consistent with the global state, properly generated honest player messages consistent with the global state, properly generated honest player uniZK proofs and messages on behalf of player d (for all $d \in C$) which are actually generated by A_d. That is, the computation-phase of a Hybrid D execution proceeds exactly as the computation-phase of a real execution. Therefore, the only difference between $REAL^{good}$ and Hybrid D is that (during pre-processing) $REAL^{good}$ uses a real execution of a GMW protocol to compute Pre, while Hybrid D uses a simulated execution of this pre-processing protocol. (Although we note that the pre-processing phase of both a $REAL^{good}$ execution and a Hybrid D execution produce properly distributed outputs for the function Pre). Therefore, Hybrid D and $REAL^{good}$ are indistinguishable because of the security of the GMW protocol used during pre-processing.

[16] Of course this assumes that A_d is giving valid uniZK proofs. However, if A_d ever gives an invalid proof, A_d would request an abort and the trusted party would immediately notify A_c.

Chapter 5

Impossibility Results

All of our impossibility results make use of the following ideal game H (observe that H is a finite game with publicly observable actions).

Figure 5-1: Game H. First, 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 all players the identities of the players who guessed correctly.

5.1 Impossibility of Broadcast-Only Protocols

Here we prove that a collusion-free protocol which works for any finite game must use a physically secure channel.[1] Since the protocol we present in Section 2 invokes a physically

[1] Although our proof is with respect to our particular definition of collusion-freeness, the intuition behind this point 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.

private channel only in the final round of pre-processing, its use of physically private channels is in some sense optimal.

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 2 *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:

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 A_1 and A_2 that cause player 1 and player 2 to each win with probability 1.

Lemma 3 *If protocol Π 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 Π (as defined in Section 2).*

Proof: 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 Π 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 messages that are broadcast into A. Whenever it is A'_c's turn to broadcast a message, it 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 copy of A it is running, the $|C|$ copies of A will always have the same state, and will generate exactly the same messages as the monolithic adversary, A, when run in the standard model.

PROOF OF THEOREM 2: Lemma 3 states that when a protocol Π uses only broadcast, then any attack by a traditional (monolithic) adversary can be simulated by independent adversaries in a collusion-free real execution. If Π were collusion-free, then these attacks could also be simulated in a collusion-free ideal execution. If Π worked for any finite game then this would contradict Lemma 2. Thus, a collusion-free broadcast protocol for any finite game cannot exist.

□

5.2 Impossibility of Infinite Games

Here we show that not all infinite games have collusion-free protocols. Although we choose a simple variation of H as our counter-example, our techniques apply to a large class of games. For example, a similar proof could show that there is no collusion-free protocol for the game "keep playing hands of poker as long as all players wish to continue".

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

Proof:

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.

From Theorem 2 we know that any collusion-free protocol for a finite game must use a physically private channel. The same is true for infinite games (by the same argument). Therefore, suppose a collusion-free protocol Π for H' were to use the physically private channel only once. There are two cases to consider.

If the physically private channel is used *after* player 1 decides to terminate the game, then at this point, all of the players' moves have to be determined by the transcript of Π 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 Π in which these moves were chosen used only broadcast. Therefore, by an argument similar to one presented in Lemma 3, during the entire period of Π when actions are chosen, the bad players can simulate any monolithic adversary.

Suppose the physical channel is used before player 1 indicates that the game should terminate. In this case there are an arbitrary number of rounds that occur after the use of the physically private channel. Let M be an upper bound on the number of bits sent to any player over the physically private 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.[2] 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.

We now consider the case where the physically private channel is used multiple times throughout the protocol. If all uses of the channel occur after player 1 decides to terminate H', then the argument from the first case above applies. Otherwise, consider a malicious player 1 and player 2 who agree before the game on a random bit b. We focus our attention

[2]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.

on the first time that the physically private channel is used after the completion of round 1. (Note if the physically private channel is used only prior to the end of round 1, then the argument from the second case above applies.) Let round r be the first round of H that begins after this use of the physically private channel. 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. The trusted party's announcement allows everyone to conclude that player 1 guessed correctly in every round and player 2's guess in round r was equal to the color of card 2 in round 1. The ability for player 2 to correlate his guesses this way is impossible in an ideal execution of H'.

5.3 Impossibility of Games With Private Actions

Here we prove that not all games with private actions have collusion-free protocols. Although we choose a simple variation of H as are counterexample, our techniques apply to a broad class of games with private actions including the game *Poker'* discussed in the Introduction.

Although there do not exist collusion-free protocols for games with private actions, 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. Note that G' has more opportunities for game-intrinsic communication than G. However, in general, the power of a set of maliciously colluding players is much less in a collusion-free protocol for G' than in a traditional secure protocol for G. (In light of Theorem 4, this is in some sense 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: Let $H*$ be the following variation of H. First, the trusted party privately hands *each* of the three players a card which is red with probability $\frac{1}{2}$ and black otherwise. Player 1 then *privately* guesses the color of card 1, Player 2 (privately) guesses the color of card 2 and Player 3 (privately) guesses the color of card 3. The trusted party then reveals the guesses of all players as well as the colors of all three cards.

Assume, for the sake of contradiction, that Π^* is a collusion-free protocol for $H*$. Assume that when the security parameter is 1^K, that Π^* has R_K rounds. Note that R_K must be polynomial in K.

Consider the following two strategies for player 1: Red_1 is the strategy "play honestly and guess red", and $Black_1$ is the strategy "play honestly and guess black". Notice that in any execution e of Π^* (with security parameter 1^K) there must be a first round $R^1(e)$ where the message prescribed by Red_1 differs from the message prescribed by $Black_1$. We can analogously define strategies Red_2 and $Black_2$ for player 2 and let $R^2(e)$ be the first round where the message prescribed by Red_2 differs from the message prescribed by $Black_2$. Finally, we analogously define Red_3, $Black_3$ and $R^3(e)$ for player 3.

Notice that there must be some player i such that for infinitely many values of K, the probability over executions e with security parameter 1^K that $R^i(e) = \min(R^1(e), R^2(e), R^3(e))$ is at least $1/3$. Therefore, there is some player $j \neq i$ such that for infinitely many values of K, with probability at least $1/6$, $R^i(e) < R^j(e)$ and player j observes player i's round-$R^i(e)$ message.

Now consider the following malicious strategies of players i and j. Before the game, players i and j agree on a (sufficiently long) random string X. In round $R^i(e)$, player i computes the message M_{red} that would be sent by strategy Red_i and the message M_{black} that would be sent by strategy Red_j. If $M_{red} \cdot X$[3] differs from $M_{black} \cdot X$ then player i chooses the guess g such that $M_g \cdot X$ corresponds to the color of Card i.[4] Otherwise, player i chooses his guess at random. In round $R^j(e)$, player j randomly selects a round $r < R^j(e)$ such that player j observed a message m from player i in round r. (If no such round r exists —e.g., if round $R^j(e)$ is the first round— then let player j selects message m at random.) player j then selects the guess corresponding to $m \cdot X$.[5]

[3]Where $M \cdot X$ denotes the inner product of M and X.

[4]Here we can use the convention that 0 means "Black" and 1 means "Red".

[5]Again, we can use the convention that 0 means "Black" and 1 means "Red".

With these malicious strategies, let us consider the probability that Player j's guess is equal to the color of Card i. For infinitely many K, with probability at least a $1/6$, $R^i(e) < R^j(e)$ and player j observes player i's round-$R^i(e)$ message. In those cases, with probability $1/2$ $M_{red} \cdot X \neq M_{black} \cdot X$ and with probability at least $1/R_K$ player j guesses $r = R^i(e)$. Notice that when all these events occur, player j correctly guesses the color of card i and in all other cases player j guesses the color of card i with probability $1/2$. So for infinitely many values of K with probability at least $\frac{1}{2} + \frac{1}{24R_K}$, player j correctly guesses the color of card i. Thus, since R_K is polynomial in K, this contradicts the fact that Π^* is collusion free. This is because in an ideal execution of H^* all strategies of player j cause j to correctly guess the color of card i with probability exactly $1/2$.

Chapter 6

Extensions and Future Directions

A Stronger Notion

In essence, our notion of a collusion-free protocol captures that when the malicious players have no side channels available then the protocol prevents the creation of subliminal side channels. However, the assumption that no pair of malicious players has any side channels available during protocol execution is quite strong and may not be applicable in many settings. Therefore, it is desirable to consider a stronger notion of collusion-freeness which provides meaningful guarantees even when some of the malicious players have side channels available.

Consider the following (informal) definition of a *strongly collusion-free protocols.* For all functions f (mapping pairs of players to integers), protocol executions in which each player i has access to an undetectable side channel that allows him to secretly send (up to) $f(i,j)$ bits of information to each player j can be simulated by ideal executions in which the ideal adversaries have access to the same side channels.

In essence, such *strongly collusion-free protocols* captures that no matter what secret side channels the players have available, the protocol does not enhance the capacity of existing side channels. In particular, this guarantees that if malicious players 1 and 2 have formed a coalition using an undetectable side channel and malicious players 3 and 4 have formed a coalition using an undetectable side channel, then the protocol prevents the former coalition from colluding with the latter.

Fortunately, our proof that the protocol presented in Section 4.1 is collusion-free protocol

can be easily modified to show that the same protocol is also strongly collusion-free.[1]

Circumventing Impossibility Results

In Section 5 we showed that in a model with broadcast messages and physical envelopes it is impossible to achieve collusion-free protocols for every game. We believe results such as this motivate the study of alternative models of communication. The work of Izmalkov, Lepinski and Micali [16] shows that in a model with physical envelopes and a simple device for randomizing the order of envelopes (namely, a ballot-box) one can achieve collusion-free protocols for any game --including games infinite games with private actions! Although their ballot-box model is a natural model of communication used in conducting secure elections for hundreds of years, it is unlikely that their model is the only one permitting collusion-free protocols for all games. It thus interesting to ask, in what other communication models can one construct collusion-free protocols for every game?

One should also note that the impossibility results in Section 5 rule out collusion-free protocols for *many* infinite (or private-action) games. However, they do not rule out collusion-freeness for *all* such games. Indeed, the notion of Fair Zero-Knowledge put forward in [18] is essence a collusion-free protocol for the infinite game in which a prover proves to a verifier, in zero-knowledge, an arbitrary number of theorems. Given that collusion-free protocols exist for this specific zero-knowledge game, it is natural to ask what other infinite (or private-action) games have collusion-free protocols in our model of broadcast messages and physical envelopes.

Improving Efficiency

An undesirable feature of our construction is that for many games (including poker) the running time of the simulators is exponential in the natural representation of the game. This

[1] Essentially, there are two cases. If the capacity of the side channel from player c to player d has constant capacity (independent of the security parameter), then whenever A_c sends a side message to A_d, A_c sends the same side message to A_d. Additionally, when checking the consistency of simulated transcripts, A_D does a further check to ensure that in the simulated transcript A_c would have actually sent the observed side channel messages to A_d. (The transcript sampling procedure can still be performed in expected polynomial time because in this case the side channel messages are all of constant size.)

Alternatively, if the side channel from player c to player d has capacity that grows asymptoticly, then for all sufficiently large security parameters, the capacity of the side channel is greater than the size of G. In this case, A_c forwards all of the ideal messages he receives to A_d. In this case, A_d has all the information needed to run A_d and can run a slightly modified UNI-GMW simulator that treats has black-box access to bothA_c and A_d.

isn't technically a problem for us since all the games we deal with have constant size. Still this inefficiency is undesirable since it necessitates choosing such large security parameters as to make a collusion-free game of Mental Poker impractical. Therefore, it is interesting to consider how efficiency one can make the simulation of a collusion-free protocol. In particular, even for our protocol, it is very possible that improved analysis could yield more efficient simulation.

In general, the proof of Theorem 3 implies some limit on the efficiency of a collusion-free protocol. (That is, if the security parameters that one uses are too small then the game-intrinsic communication can be used to transmit a substantial amount of information about the players' secret keys and violate collusion-freeness.) However, there seems to be a gap between the size of the security parameters required by our construction and the lower bounds implied by Theorem 3.

Composability

It is easily seen that universally composable collusion-free protocols are impossible to construct (at least in our model of broadcast messages and physical envelopes[2]). Intuitively, malicious players can share the secret keys, created during the preprocessing of a collusion-free protocol, using inherent communication in other protocols running concurrently.

However, although arbitrary composition of collusion-free protocols is impossible, it is interesting to consider more limited settings of protocol composition. For example, is it possible for a set of players to play to two fair games of poker concurrently?

[2]The ballot-box protocol of [16] is both collusion-free and universally composible.

Bibliography

[1] Michael Backes and Christian 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] Christian Cachin. An information-theoretic model for steganography. In *Proc. of Information Hiding '98*, pages 306–318, 1998.

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

[5] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Noar. Adaptively secure multi-party computation. In *Proc. 28th STOC*, pages 639–648, 1996.

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

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

[8] Alfredo DeSantis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge with preprocessing. In *CRYPTO 1988*, pages 269–282. 1991.

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

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

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

[12] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for np. *Journal of Cryptology*, 9(3), 1996.

[13] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proofs. *Journal of the ACM*, 38(3), 1991.

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

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

[16] Sergei Izmalkov, Matt Lepinski, and Silvio Micali. Rational secure function evaluation and ideal mechanism design. In *Proceedings of FOCS '05*, 2005.

[17] Matt Lepinski, Silvio Micali, and abhi shelat. Collusion-free protocols. In *Proceedings of STOC '05*, 2005.

[18] Matt Lepinski, Silvio Micali, and abhi shelat. Fair-zero knowledge. In *TCC 2005*, pages 245–263, 2005.

[19] Nicholas Hopper Luis von Ahn and John Langford. Provably secure steganography. In *Crypto '02*, 2002.

[20] Leonid Reyzin Nenad Dedic, Gene Itkis and Scott Russell. Upper and lower bounds on black-box steganography. In *TCC '05*, 2005.

[21] Luis von Ahn and Nicholas Hopper. Public-key steganography. In *Eurocrypt '04*, 2004.