Collusion-Free Protocols in the Mediated Model

Joël Alwen[1], abhi shelat[2], and Ivan Visconti[3]

[1] New York University
251 Mercer St. New York, NY, 10012, USA
jalwen@cs.nyu.edu
[2] University of Virginia
Charlottesville 22904, USA
shelat@virginia.edu
[3] Dipartimento di Informatica ed Appl., Università di Salerno
84084 Fisciano (SA), Italy.
visconti@dia.unisa.it

Abstract. Prior approaches [15, 14] to building collusion-free protocols require exotic channels. By taking a conceptually new approach, we are able to use a more *digitally*-friendly communication channel to construct protocols that achieve a stronger collusion-free property.

We consider a communication channel which can filter and rerandomize message traffic. We then provide a new security definition that captures collusion-freeness in this new setting; our new setting even allows for the mediator to be corrupted in which case the security gracefully fails to providing standard privacy and correctness. This stronger notion makes the property useful in more settings.

To illustrate feasibility, we construct a commitment scheme and a zero-knowledge proof of knowledge that meet our definition in its two variations.

Keywords: Secure Collusion Free, Mediated Communication, ZKPoK, Commitments

1 Introduction

The Federal Communication Commission in the United States just finished auctioning several bands of communication spectrum. This was the seventeenth such auction run by the FCC and the commission has gained quite a bit of experience in running them. In particular, based on behavior in prior auctions, the commission employed several ad-hoc rules in order to maximize the revenue generated by this auction. One major concern for them is the problem of *collusion* between bidders. As documented in [9], in a prior auction, although many rules prohibited explicit collusion between the bidders, bidders have nonetheless devised clever signaling strategies *during* the auction in order to cheaply divide the auctioned rights. Thus, it is safe to say that an issue at the forefront of FCC's auction design team is to prevent bidders from engaging in such collaborative bidding strategies.

1.1 Does Cryptography Help the FCC?

It has long been held that secure multi-party computation protocols provide the most robust and secure way for implementing any collaborative task, including the running of an auction. Indeed, on face, secure multi-party protocols would mimic the role of a trusted-third party who ran the auction—without actually needing to trust a third party. While in practice, the FCC can function as such a third-party, doing so is not optimal. The FCC is a huge organization and the participants in high-value auctions might naturally question whether FCC "insiders" are helping the competition, by say leaking bid information during the auction. Thus, from both a practical and theoretical perspective, we ask how cryptography can improve the security of an auction.

There is unfortunately a serious problem which complicates the use of cryptography in such auctions. As pointed out by Lepinski, Micali, and shelat [15], the use of a cryptographic protocol can undo all of the carefully planned measures designed by the auctioneer to prevent collaborative bidding. In particular, since secure cryptographic protocols are randomized, a protocol may unintentionally produce a steganographic channel which bidders may use to communicate illegally about their bidding strategies. Thus, a careless use of cryptographic protocols would undo all of the mechanism rules designed to avoid collusion. This is one of the first examples of how cryptographic protocols may in fact be *worse* than trusted-third parties from a conceptual (as opposed to efficiency) point of view.

Fortunately, the authors of [15] do suggest a solution. They define and construct *collusion-free protocols* in a model in which players can exchange *physical envelopes.* Their security notion guarantees that no new method for players to collude are introduced by the protocol itself. Thus, their solution represents the best of both worlds: all of the auction rules which prevent collusive bidding remain effective, and yet all of the privacy and correctness properties implied by good cryptography are enjoyed as well.

To achieve the collusion-free property, the authors of [15] design a *verifiably deterministic* protocol in which at each point, a participant can only send a single message that the other players will accept: while each protocol message is unpredictable by the others, it is nonetheless, verifiable as the only appropriate next message. This property has also been called *forced action* to capture the idea that a player has only one (forced) action which it can take to continue the protocol. All other messages are interpreted by the other players as a signal to abort. Later, Izmalkov, Lepinski, and Micali [14] also use the idea of forced action and, by employing a new assumption concerning a physical randomizing ballot box, are able to construct protocols which are both information theoretically secure and collusion-free.

Notice, however, that both of these solutions employ exotic physical communication channels: the envelope and the ballot box. A principle criticism with these assumptions is that they have no analogues in the digital world. A physical ballot box relies on the forces of nature to provide randomness. But no party can ever be trusted to implement this in a digital (remote) setting because the moment some entity is trusted with generating the ballot box randomness, that entity essentially becomes entrusted with all of the security properties. Thus it might as well act as the trusted party computing the ideal functionality. Another (potentially less problematic) argument is that envelopes are both perfectly binding and perfectly hiding which again can not be implemented in a digital setting. Thus we are left with no choice but to implement these protocols in a physical setting.

Yet the engineering requirements of implementing such a *physical* communication model without introducing a subliminal channel seem daunting. Every last detail would have to be precisely controlled, from how exactly envelopes are transported from one player to another player, the timing of all events, marks on envelopes must be avoided and even their ambient temperature must be regulated. In the past, attacks such as reconstructing computer screens from reflection off of walls, extracting secret keys from smart cards meters away via the electric fields from card readers, or listening to private conversations by measuring the vibrations of window panes with lasers have shown that physical security and in particular physical isolation can be a *very* difficult to achieve.

Despite the difficulties with implementing these physical channels, some extra communication channel are provably necessary [15] in order to achieve collusion-freeness. Nonetheless, in a digital world, the (im)practicality of such physical channels makes it worthwhile to explore other solutions to the problem of collusion in cryptographic protocols, and our approach does precisely this.

1.2 Our New Approach

This paper addresses the problem of building collusion-free protocols without using physical channels.

Our insight comes from studying the *opposite* of verifiable determinism. Recall, the motivation behind forced-action was to remove the inherent entropy of cryptographic messages. Removing the entropy was necessary to remove the possibility of using steganography to collude during a protocol execution. Instead of taking this approach, we consider *adding* more randomness to each message so that any hidden message is smothered.

A protocol step in our approach may allow many acceptable messages and may require a lot of randomness—even randomness chosen by the prover and verifier. To avoid the previous problems of steganographic channels, our protocols only allow communication via a *mediator* communication channel. This channel has a very simple task: it removes any steganographic information from the messages sent by players by "rerandomizing" the messages. The mediator is similar to the "warden" model introduced by Simmons [20]. A first step in this direction was done in [1].

At first, it may seem that a mediator requires a substantial trust investment on behalf of the players. Let us first note that the amount of trust placed in the channel is no more than that placed in other such secure protocols. In particular, the private channel model used by Ben-Or, Goldwasser and Widgerson [4], the ballot box [14] and the communication model in [16] also assume that the adversary does not control or modify the messages sent by players. In fact, many protocols make implicit assumptions about their communication channels, be they private channels, common reference strings, synchronous channels, synchronous broadcast, envelopes [16], or ballot boxes [14]. Often the assumptions made on the communication channel are neither made explicit, nor understood explicitly.

While our channel is incomparable to these other ones, it remains plausible since any modern day router can implement the protocol instructions we require of the mediator. In particular, many modern Internet routers implementing IPsec already do many similar cryptographic operations on the data that they relay. In this sense, our mediator channel seems to be a *digital* rather than physical way to achieve the collusion-free property. As such we eliminate the need for stringent engineering requirements as players may be physically separated by great distances and any contact is on a purely informational basis.

Most importantly, unlike other protocols that employ exotic communication channels, we model the mediating communication channel as a party that can be corrupted just like other participants. (Indeed, in real life, the mediator would probably be an entity like the FCC and thus it is natural to consider what happens when it is corrupted.) As it turns out, our protocols become secure with respect to traditional multi-party protocol security notions when the mediator is corrupted. Thus, we only rely on the mediator to guarantee the collusion-free property—this is yet another reason our model is a more palatable one. If the FCC were the mediator, it is their natural interest to act honestly so as to prevent collusion. Even so, a cheating FCC would be unable to affect the privacy or correctness of the auction thereby mitigating insider threats.

Why Universal Composability Does not address the Issue A natural question is to consider why the strong notion of universally composable security [6] is not sufficient for our purposes. Here the problem is one of modelling. Like prior security notions, UC (and its various extensions) continues to model the adversary as a monolithic entity. Thus, the security model already assumes that the corrupted parties can collude at-will among one-another during the protocol execution.

Our Contributions To formalize our ideas, we present a new security definition for collusion-free security which models the communication channel as a corruptible participant. As discussed, our definition captures "graceful failures" with respect to the channel. Since this is an important part of our contribution, we discuss the communication model and formally define and discuss the security notions in Section 2. To illustrate the feasibility of our notions, we present a collusion-free commitment scheme in Section 3. The proof for this protocol illustrates technical issues of collusion-free security and serves as a warm-up for the secure collusion-free zero-knowledge proof of knowledge protocol we present in Section 4.

2 Model and Security Definition

We use the ideal versus real experiment indistinguishability paradigm introduced in [11] to formalize our new ideas about secure computation. Real world players are denoted with a san serif font like P, ideal players and functionalities are denoted with a calligraphic font like $\mathcal{P}$, and malicious (i.e., adversarial) players have a tilde like $\widetilde{\mathsf{P}}$ and $\widetilde{\mathcal{P}}$. We introduce a new player M (and $\mathcal{M}$) who controls all communication between the players.

The standard security notion formalizes the idea that regardless of their input to f, for any set of corrupted real world players there exists a *single* ideal world simulator which can reconstruct everything the set of real players might learn during the execution of the secure protocol. This means *no* information beyond that exchanged in the ideal world can be exchanged between colluding real world parties.

We use a different adversarial model than the traditional monolithic approach. In particular we consider malicious parties $\widetilde{\mathsf{P}}_1 \ldots \widetilde{\mathsf{P}}_t$ who are playing a real world protocol with each other via the mediator M. In the ideal model, we place restrictions on the way in which $\widetilde{\mathcal{P}}_1 \ldots \widetilde{\mathcal{P}}_t$ can communicate with each other. In particular they can only communicate via a call to $\mathcal{F}$ or by using $\widetilde{\mathcal{M}}$ as in intermediary. (This idea comes from [15], but they do not model $\widetilde{\mathcal{M}}$.)

We wish to capture the intuition that the *distinct* adversaries $\widetilde{\mathsf{P}}_1 \ldots \widetilde{\mathsf{P}}_t$ should not be able to use an execution of the secure real world protocol as a means of computing any joint functionality beyond the intended ideal functionality implemented by the protocol. Clearly this goal will require a *steganography-free* real world protocol. Further we require that the real world protocol emulates $\mathcal{F}$ even in the face of a *set* of *coordinated malicious players*. Traditional security guarantees for multi-party computation are incomparable in this sense as they only constrain a monolithic adversary and how it can affect other players. For example although commitments are hiding if the sender is honest, a UC commitment can be used to compute any other functionality if sender and receiver are both corrupt and coordinated.

If mediator $\mathcal{M}$ is honest, it never sends messages to a player, and thus $\mathcal{F}$ is the only means of inter-player communication. If, on the other hand, the mediator $\widetilde{\mathcal{M}}$ is corrupt, then all other corrupt parties in both the ideal and the real world models are able to perfectly coordinate their actions. Specifically, they (in the worst case) can use $\widetilde{\mathsf{M}}$ in the real world and $\widetilde{\mathcal{M}}$ in the ideal world as perfectly secret and authenticated channels. Thus, in this case our security notion essentially collapses to the traditional monolithic model.

Per this discussion, our definition of a collusion free protocol comes in two parts. The first captures the case when the mediator is honest, and the second part adds the property of "secureness" by requiring that when the mediator is corrupt then any real world attack can also can be simulated in the ideal world by a monolithic adversary. In other words the mediator should only be trusted for avoiding collusions between players. If it is dishonest then the security of the protocol is (almost[4]) the same as traditional SFE security.

Authenticated Channels Without a setup stage, it would be impossible for any player to establish the authenticity of information which supposedly originated from another player. This is particularly true in our model since all players communicate through M, and so a corrupted M could mount a classic man in the middle attack, hijack an identity mid protocol or route messages to the wrong destinations.

We note that [15] also requires a pre-processing phase to setup keys. In particular, they require a preprocessing round which must take place before inputs are distributed. Arguably this is an even greater problem as in certain settings (such as some auctions) it is unrealistic to expect players to meet before inputs are ever distributed. The protocol in [14, 13] implicitly uses authenticated channels since all of the participants are in physical proximity.

There are several approaches to handling this problem. The first is to assume a Public Key Infrastructure (PKI) and assume that players have registered keys and proven knowledge of the secret key. This is an approach implicitly taken by many protocols; however it is unsuitable in our work for two reasons. First, it implicitly requires that all parties perform some global action before receiving their private inputs. And secondly, it creates a technical problem for simulation. To handle a corrupt mediator, a simulator must feed messages on behalf of the honest players. If these honest players have registered public keys, then we must assume the mediator (who represents the communication channels) must also know the public keys of the honest users, and will therefore expect signatures under those keys for all messages. (Note, much like the GUC-model, we assume that the mediator can probe the *real* PKI and retrieve the real public keys of the players.) Thus, the simulator will need to know the secret keys of the honest players which seems unreasonable.

A second method is to add a setup phase to a protocol during which keys are shared. Since players know their inputs, to remain collusion-free, all communication must be passed through the mediator. In this case, however, a corrupt mediator can "fork" the honest parties into separate groups by creating several different public keys for some players. There is no broadcast channel when the mediator is corrupt, so an honest player must take on faith that the key it has received for some other player j is the one actually created and sent by player j. The authors of [3] study this forking model and suggest protocols and definitions for the situation. The protocol is complicated because it must handle non-malleability issues. We are currently working on using these tools for this task.

However, because our goal is to focus on the collusion-free property, we resolve this issue in this preliminary version by only considering broadcast-honest

[4] Even in the ideal world the mediator is allowed to choose a forced abort set. We have chosen to make this explicit but we point out that in any model where the adversary is given full control of the network this power is implicit.

mediators. These are mediators who cheat arbitrarily, but perform one broadcast operation during the protocol honestly. Another way to consider such a mediator is to assume that there is a public bulletin board that all players can see, and that the mediator posts everyone's public key on this board. Such a notion is already considered in the electronic voting literature [7, 5, 18, 8]. Overall, restricting attention to these simpler cheating mediators clarifies the collusion-free issues; investigating ways to remove this restriction is a task for future work.

Aborts An overlooked problem when considering collusion free protocols is signaling via aborts. The easiest solution is to simply assume no party will ever abort. For game theoretic applications (with punishment strategies) this remains an interesting security notion. Another approach might be to say that if one party aborts then all parties are forced to abort and no subsequent protocol is played. However in the interest of more general applications and a more robust security notion we take a different approach. We explicitly model aborts in the ideal world with greater detail.[5]

To pinpoint the issue, in prior ideal models for SFE, a party can simply abort; but in the real model instead a party can abort at a given round. Thus, it seems necessary that the ideal model allows a "round number" to be communicated during an abort in order to properly capture the full power of a real-world adversary. To specifically accommodate this in the ideal world, every abort message is accompanied by an integer. Let us emphasize that this phenomena only becomes an issue when considering the problem of steganographic communication and colluding parties since it might allow a second adversary to benefit. So we point out that this is not actually a weakening of our ideal model with respect to previous ones but rather a more detailed specification of the real security achieved. We also mention that abort messages in the ideal world are handled instantly. That is normally $\mathcal{F}$ will wait for all inputs before computing f and returning output. However if it receives an abort message it immediately sends this to all players and terminates. This corresponds to players knowing immediately (at least at the end of the current communication round) when a player has aborted in the real world rather then only finding out once the entire computation is complete. In the interest of clarity we will describe the effects of this design choice on our proofs (in the full version) in general terms. So as not to get lost in unnecessary details we make almost no further mention of aborts in the proofs and refer only to this section.

2.1 Notation

Let f be an n-input and n-output function $f : D \times D_1 \times \ldots \times D_n \to R \times R_1 \times \ldots \times R_n$. Here D_i is the domain of player i's private input and D is the domain of the public input. For example for zero knowledge D is the set of statements and D_{P} is the set of witnesses. Similarly R is the range of the public output

[5] In particular in [15] aborts are modeled via a simple "abort flag" both in the ideal and real games.

while R_i is the range of player i's private input. (The public input and public output are just a special component of all players inputs and outputs which is the same.)

We denote by $\Pi = \langle \mathsf{P}_1(x_1, \mathtt{aux}_1), \ldots, \mathsf{P}_n(x_n, \mathtt{aux}_n)\rangle$ a protocol where player i has input x_i to f. (Note that honest players will faithfully use these as their inputs, but malicious parties may not.) The strings $\mathtt{aux}_1$ through $\mathtt{aux}_n$ model prior information that players have at the beginning of an execution. Let $[n]$ to denote the set $\{1, 2, \ldots, n\}$. If $A \subseteq [n]$ then we write e_A to refer to the arbitrary output of all players in A. We write $\bar{A}$ to refer to the compliment of A in $[n]$.

A subscript of $\mathbb{I}$ denotes an ideal world protocol while a subscript of $\mathbb{R}$ denotes a real world protocol. For ideal world protocols a super script of $\mathcal{F}$ implies that all parties have access to the ideal functionality $\mathcal{F}$. Thus $\langle \mathsf{S}, \mathsf{R}, \mathsf{M}\rangle_{\mathbb{R}}$ is a real world protocol between S and R with mediator M while $\langle \mathcal{S}, \mathcal{R}, \mathcal{M}\rangle_{\mathbb{I}}^{\mathcal{F}}$ is an ideal world protocol between $\mathcal{S}$ and $\mathcal{R}$ who have access to ideal functionality $\mathcal{F}$ (and ideal mediator $\mathcal{M}$). We will use $\mathtt{VU}(\mathsf{P}(x))$ to denote the entire view of P for an execution with input x (including random tape, inputs and messages recieved) and we use $\mathtt{IO}(\mathsf{P}(x))$ to denote only the input and output pair of an execution.

Ideal execution In the ideal execution, all players, including the ideal mediator $\mathcal{M}$, have private channels to the ideal functionality $\mathcal{F}$. In addition, $\mathcal{M}$ has bi-directional channels to all ideal players.

A round of an ideal protocol begins with all players $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and $\mathcal{M}$ sending their input to $\mathcal{F}$. The mediator's input to $\mathcal{F}$ is a tuple $(S_1, \ldots, S_\ell, I_i, \ldots, I_\ell)$ which allows the corrupt mediator to "fork" the different players into separate groups. Namely, $\mathcal{M}$ inputs a list of non-empty disjoint subsets $S_1, \ldots, S_\ell$ that partition the set of players $\{1, \ldots, n\}$. In addition, for each S_i, $\mathcal{M}$ specifies the inputs I_i to $\mathcal{F}$ for the other players $\bar{S}_i$.

For each S_i, the ideal functionality evaluates $\mathcal{F}$ on the inputs provided by the players in S_i and the inputs for the players in $\bar{S}_i$ specified by I_i. It then begins to deliver the outputs to the players in S_i in some canonical order. When it is player j's turn to receive output, the ideal functionality queries $\mathcal{M}$ about whether to deliver the output or whether to broadcast an integer $a \in [\mathtt{poly}(k)]$ and halt. If $\mathcal{M}$ agrees to delivery, then the ideal functionality sends the output to that player and continues to the next player in order. The output values for the players in $\bar{S}_i$ are sent to $\mathcal{M}$. If instead $\mathcal{M}$ chooses to broadcast an integer a, then the ideal functionality sends $(\perp, a)$ to all players and halts.

$\mathcal{F}$ can maintain internal state between rounds allowing for such functionalities as commitment schemes. In the ideal model:

- an honest player never sends a message to $\mathcal{M}$;
- an honest $\mathcal{M}$ never sends a message to a player, always inputs $S_0 = [n]$ and $I_0 = \emptyset$ to $\mathcal{F}$ and always agrees to output delivery;
- a corrupt ideal mediator $\widetilde{\mathcal{M}}$ does not see the private outputs of $\mathcal{F}$ (at least for honest players). The only values related to the computation of honest parties it learns is the public input and public output and the output values for each of the corrupted players.

Real execution In the real world, there is no $\mathcal{F}$ and instead players have authenticated private channels connecting them to a special party M called the (real world) mediator. Thus M sits at the center of a star network connected to P_1 through P_n “mediating” all communication. As a result, M controls many aspects of the computation which necessitates the complicated provisions given to the ideal mediator described above.

Real world executions begin with each player and the mediator selecting fresh independent uniform random strings and jointly executing an interactive protocol. At the end of both a real and an ideal execution all players privately output an arbitrary string which captures information they have learned. The honest mediator always outputs the special message $\perp$. Honest parties output the results of computing f, but corrupted players can output whatever they wish.

2.2 Collusion Free Protocols

As usual with ideal/real paradigm definitions, a protocol is considered secure if the probability ensemble representing a real execution is indistinguishable from an ensemble representing an ideal execution. The definition below, however, changes the order of some quantifiers to capture a specific concern.

Definition 1. *Let $\Pi = \langle \mathsf{P}_1, \ldots, \mathsf{P}_n, \mathsf{M} \rangle_{\mathbb{R}}$ be an n player protocol with security parameter k and let $\mathcal{F}$ be an ideal functionality computing the n-input and n-output function f. A protocol Π is a* collusion free protocol for $\mathcal{F}$ *if for any efficient real player $\widetilde{\mathsf{P}}$ (including those playing the honest protocol) there exists an efficient ideal simulator $\widetilde{\mathcal{P}}$ with access to a common random tape R such that for any vector of players $(\widetilde{\mathsf{P}}_1, \ldots, \widetilde{\mathsf{P}}_n)$ for all inputs $x_i \in \{0,1\}^*$ and for all $\mathtt{aux}_i \in \{0,1\}^*$, the following two ensembles are indistinguishable:*

$$\left\{ \left\langle \left\{ \mathtt{VU}(\widetilde{\mathsf{P}}_i(x_i, \mathtt{aux}_i)) \right\}_{i \in [n]}, \mathtt{IO}(\mathsf{M}(\mathtt{aux}_{\mathsf{M}})) \right\rangle_{\mathbb{R}} \right\}_k$$

and

$$\left\{ \left\langle \left\{ \mathtt{VU}(\widetilde{\mathcal{P}}_i(x_i, \mathtt{aux}_i, R)) \right\}_{i \in [n]}, \mathtt{IO}(\mathcal{M}(\mathtt{aux}_{\mathsf{M}})) \right\rangle_{\mathbb{I}}^{\mathcal{F}} \right\}_{R,k}$$

Further, we call Π a secure *collusion free protocol if in addition, every mediator $\widetilde{\mathsf{M}}$ also has a simulator $\widetilde{\mathcal{M}}$ for which the indistinguishability holds.*

In the above definition, the ensembles are taken over the values of k, R and the random choices of the players and mediator. By “efficient” adversaries we mean a probabilistic polynomial time one in the size of the input and security parameter.

Comments Since the experiments' outputs include the complete vector of arbitrary output strings, the components must have a similar *joint* distributions in both experiments.

For the case of collusion freeness, M is honest, so the ideal adversaries must be able to produce the same joint output distribution as in the real world, but only by communicating via at most one call to $\mathcal{F}$. Therefore these sets of *distinct* adversaries can not jointly compute anything more then what is revealed by f. In particular they can not use the real world protocol to exchange information steganographically or jointly compute anything beyond what one call to f reveals.

Secure collusion-free protocols are more general. If mediator is corrupt then since honest parties output their results from f, even a monolithic adversary (i.e., distinct adversaries perfectly coordinating through the corrupt mediator) can not alter the output of the joint functionality nor learn anything about it beyond what their own output reveals. (The worst they can do is cause aborts as $\widetilde{\mathcal{M}}$ gets to choose the forced abort set.) This guarantees the traditional security properties such as privacy and correctness for general SFE in much the same way the UC framework captures these notions. (Though with weaker composability properties since a distinguisher for multi-party collusion freeness only sees inputs and outputs but does not have all the powers of the environment in UC definitions.)

Another subtlety is a consequence of the order of the quantifiers and the intuitive attack scenario we model. For every corrupt player, there exists a simulator with access to a common random tape R shared amongst all such simulators. This means that the ideal simulators do not know which other parties are corrupt. The only joint information they are given is the random tape R. One might argue that it is meaningful for simulators to know the complete set of corrupt parties and their code. After all colluding parties may well be perfectly coordinated going into the protocol. However such a definition would not preclude real world player from using an execution of Π to exchange information unrelated to the current execution. So for example it might be that a pair of corrupt parties which have had no previous contact could use an execution to exchange phone numbers. Although they might have done this anyway in advance we still would like to *force* them to have exchanged this information before hand. This allows for applications such as the online poker house or security in a large intelligence agencies network where individual players may never had contact before.

By requiring that all simulators only share the (read only) random tape R, we model the fact that in the real world protocols, players observe common randomness although they can not influence what this randomness is (via secure secret sharing for example). However beyond this publicly visible randomness no information is exchanged. We note that this is a strengthening with respect to Definition 1 in [15]. Further, the fact that players can agree on common randomness after a real execution is also a property of the ballot-box protocol in [14] and so technically breaks the central theorem of perfectly emulating the ideal game. In the context of game theoretic applications this may in fact present a real problem as it is known that correlated randomness can be used to achieve equilibria with better expected payoffs then otherwise possible [2]. Thus at a minimum perfect composability is lost. The problem in [14] has been addressed in follow up work [13]; we also resolve it for this protocol in the full version.

Another improvement over the collusion free definition in [15] is that we no longer require a "reconciliation function" to compare the real and ideal world outputs. This simplifies both the definition of collusion-free protocols and simplifies the security proof strategy.

2.3 Authenticated Channels

Typically, key registration is handled by a trusted-third party. As mentioned above, since we do not want to assume such a PKI and also want to model security when external trust assumptions fail, we describe a pre-processing protocol using the mediator. Naturally, our process provides a somewhat weaker form of security then might be achieved with a trusted key registration server. In particular there is no guarantee that an honest player will successfully register his key when the mediator has been corrupted. But on the other hand even if identities were established perfectly once computation starts $\widetilde{\mathsf{M}}$ can always cut off a player i and no other player j can tell whether this is because of a corrupt $\widetilde{\mathsf{M}}$ or corrupt player i. In some sense, this is unavoidable.

The other weakness of our preprocessing is that $\widetilde{\mathsf{M}}$ can fake the presence of a player by including a public key for them. However this too is unavoidable since even with perfect authentication $\widetilde{\mathsf{M}}$ could simply setup a dummy player with an identity and take part in the authentication and joint computation stages through the dummy. As such, this is a problem which is outside the scope of this work.

Initialization of the set of parties. A first approach to key-setup might be to have parties generate a key pair, register their public key with M and run a $\mathtt{ZKPoK}$ to prove they know the corresponding secret keys. However, as noted by [15], if public keys are selected by the parties, the keys may not be chosen honestly. For example, they can be chosen from intentionally "weak" or special distributions which allow other parties to break the key. This could allow parties to exchange information during the protocol. To solve this problem, a more elaborate registration procedure is executed. First each player runs a coin-flipping protocol with M (without the final decommitment) such that only the player knows the result of the coin flipping. The player then uses this randomness as input to the algorithm that generates the key pair. Then the public key is sent to M along with a $\mathtt{ZKPoK}$ that it used the result of the coin flipping to generate the pair. Once all public keys have been registered with M it broadcasts the set and the joint computation phase can begin. We call the resulting initialization stage Δ_n^{M}.

There is one technicality to handle. We require that the $\mathtt{ZKPoK}$ enjoys the concurrent zero knowledge property [10, 19]. Indeed, an adversarial M could run as verifier of the $\mathtt{ZKPoK}$s by coordinating the executions in an adaptive and malicious way with all honest parties. In order to avoid such attacks we require a concurrent $\mathtt{ZKPoK}$ protocol [19, 17].

Collusion Free Authentication. Once keys have been shared, there are various ways in which they can be used to emulate authenticated channels. Since it is not our principle focus, we describe the following correct but not necessarily most efficient method for doing so. Let us briefly note the subtle requirements which prevent standard protocols from being adopted. On the one hand, we cannot allow one player to send a signature to another (since that would enable steganography), but we must provide enough evidence to the second player to accept authenticated messages.

At the beginning of each real world collusion free-protocol execution e each player P_i sends a random k-bit string r_e to M along with a signature σ of r_e (where k is the security parameter). The string r_e will act as P_i's session identifier for e. M sends $\mathtt{Com}(r_e)$ and $\mathtt{Com}(\sigma)$ to all other players P_j and proves in ZK that $\mathtt{Com}(\sigma)$ decommits to a signature of the decommitment of $\mathtt{Com}(r_e)$ which verifies under $\mathtt{svk}_i$. A subsequent round t, message m_e to player j of the execution e is accompanied by a signature of $\sigma' = \mathtt{Sig}_i(r_e, t, \mathtt{svk}_j, m_e)$. M proves in zero knowledge to player j that there is a signature σ' of a message with the following prefix. σ (i.e., the decommitment of the first message $\mathtt{Com}(\sigma)$) is a valid signature for the first part of the prefix, and the second part of the prefix t, is the current round number, and the third part of the prefix equals player j's verification key. Thus the player j knows that m_e came from P_i playing a session with the current random identifier, that it is the t-th round message in such a session and that it is the intended recipient. With overwhelming probability (in k) there will only be a single session with this session identifier so the message must be from P_i during this session.

Calling Subroutines One issue remains with this approach. The collusion free protocol e may use another collusion free protocols e' as subroutines. To make sure that M can not run replay attacks emulating P_i for the entire l-th call of the subroutine (by playing an old $r_{e'}$ and σ) the subroutine is essentially treated as a message in the calling protocol. In particular all signatures (including the first σ of the subroutine) are of messages prefixed by l, the number of the call and r_e, the session identifier of the calling protocol. For each message in e' M additionally proves in ZK that this extra prefix corresponds to the session identifier of e and the current call number. If M can substitute any previous sub-protocol for P_i then M can be used for existential forgery attacks against the signature scheme since either r_e or l will be different (with overwhelming probability in k).

For most of the rest of this paper we will not explicitly discuss the authentication mechanism as this will needlessly complicate exposition. The only exception to this is when we discuss how ideal world simulators internally emulate honest players when running their experiments as this will require them to be able to create signed messages.

3 Collusion Free Commitments

The first primitive we consider is a commitment scheme. That is we consider the functionality $\mathcal{F}_{\texttt{Com}}$ which works in two rounds. In the first round it takes a message m as input from the (ideal world) sender $\mathcal{S}$ and produces output *committed* for the (ideal world) receiver $\mathcal{R}$. In the second round it takes input *decommit* from $\mathcal{S}$ and gives the output m to $\mathcal{R}$. All other inputs and outputs are the special message $\bot$.

The Difficulty Before continuing, let us discuss why this is a non-trivial task. Players can not be allowed to directly exchange messages as this would enable steganography. So the mediator is used to rerandomize communication. However there is a more subtle attack. Consider the last round of the real world protocol to be played by a particular player. Suppose they can tell the value of even just two (non-constant) bits in another parties view (or even a function there of). Then by choosing whether or not to abort depending on the value of those bits in the current execution the player can signal information via the real world protocol (beyond the mere abort message modeled in the ideal world). Therefore having the mediator ensure messages are randomized correctly is not enough. In fact there can not be any (computable) correlation—be it chosen in advance or random but fixed—between players' views. I.e. it is not enough to ensure one player can not fix a function of some bits in another's view, but it must even be impossible for one player to subsequently guess any function of some bits of another's view after they have been fixed.

Yet at the same time we do not want to require the mediator to be honest for the security of the protocol (i.e., hiding and binding for a commitment scheme). In the following protocol we resolve these conflicting requirements.

The Protocol In figure 1 we give a protocol which makes use of a statistically binding commitment scheme $(\texttt{Com}, \texttt{Dec})$ and a zero knowledge proof of knowledge $\texttt{ZKPoK}$. We then prove it to be a collusion free protocol for the $\mathcal{F}_{\texttt{Com}}$ functionality. In particular we prove that corrupt $\widetilde{\mathsf{R}}$ and $\widetilde{\mathsf{S}}$ can be simulated by a pair $\widetilde{\mathcal{R}}$ and $\widetilde{\mathcal{S}}$ in the ideal world. Further we prove the protocol to be hiding if S is honest and statistically binding if R is honest[6]. We note that one-way permutations suffice to construct all three primitives.

Theorem 1. *Assuming the existence of one way-functions the protocol in figure 1 is a statistically binding commitment scheme. Further it is a collusion free protocol for the functionality* $\mathcal{F}_{\texttt{Com}}$.

Proof. (High-level proof idea) The idea behind the following proof can be summarized as follows. The hiding property of the protocol is reduced to the hiding of $(\texttt{Com}, \texttt{Dec})$ and the zero knowledge property of $\texttt{ZKPoK}$ in steps C.1 and C.2. The

[6] We note that if a statistically hiding commitment scheme $(\texttt{Com}', \texttt{Dec}')$ is used instead of $(\texttt{Com}, \texttt{Dec})$ and the zero-knowledge argument $\texttt{ZK}$ is statistical zero knowledge then the resulting protocol is statistically hiding. The proof remains largely the same.

Common Input: Signature verification key $\mathtt{svk}_{\mathsf{S}}$.
Committer Input: Signature key $\mathtt{ssk}_{\mathsf{S}}$ and message m

Commitment Phase

C.1 S selects random coins r, and sends $c = \mathtt{Com}(m, r)$ to M.
C.2 S runs $\mathtt{ZKPoK}$ with M proving knowledge of m and r such that $c = \mathtt{Com}(m, r)$.
C.3 M selects random coins r' and sends $c' = \mathtt{Com}(c, r')$ to R.

Decommitment Phase

D.1 S sends $(m, r) = \mathtt{Dec}(c, r)$ to M.
D.2 M sends m to R.
D.3 M runs $\mathtt{ZKPoK}$ with R proving that there exist coins r and r' such that $c' = \mathtt{Com}(\mathtt{Com}(m, r), r')$.

Fig. 1. Collusion-Free Perfectly-Binding Commitments

binding property of the protocol in figure 1 is reduced to the binding property of $(\mathtt{Com}, \mathtt{Dec})$ and the soundness of the $\mathtt{ZKPoK}$ in step D.3. Both of these proofs appear in the full version.

To prove collusion freeness we construct two simulators, one for a corrupt sender and one for a corrupt receiver. The simulator for the Sender extracts the values m, r and forwards appropriate messages to the ideal functionality. The simulator for the Receiver sends a commitment to 0 during the commit protocol, sends the committed message (received from the ideal functionality) during the decomit protocol and uses the simulator for the $\mathtt{ZKPoK}$ to equivocate the commitment. The hiding property of the commitment scheme is necessary to prove that these simulators meet the definition.

Lemma 1. *Assuming the existence of one-to-one one-way functions, the protocol in figure 1 is a collusion-free protocol for* $\mathcal{F}_{\mathtt{Com}}$.

Proof. Omitted for space.

Achieving Secure Collusion-Free Commitments For lack of space we only mention here how to achieve secure collusion-free commitments. First of all, the setup phase requires that each player knows the public key of each other player, where the public key consists of a pair of public keys, one for signature, and one for rerandomizable CPA encryption (e.g. El Gamal).

In the protocol, M receives a commitment of m from S, sends to R the commitment of the commitment and proves to R in zero knowledge that the committed message has been sent by S. This ends the commitment phase.

The opening phase is instead more complex and is played as follows. M and S play a simulatable coin-tossing protocol such that only M obtains the output.

M receives from S an encryption of m under R's public-key. The ciphertext is rerandomized by M and sent to R, using as randomness the output of the coin-tossing protocol. M also proves to R that the ciphertext is a rerandomization of a ciphertext sent by S, using as randomness the one established by the coin-tossing protocol. So far, m has been read by R and not by M, however a consistency check with the commitment phase has still to be played. The opening phase continues with S that proves in zero-knowledge to M, using a specific implementation of Blum's protocol that the encrypted message corresponds to the committed one. The specific implementation goes as follows. Consider a triple of messages (a, e, z) of Blum's protocol. After receiving a, M computes a commitment c of a and sends it to R. Then M plays again with R a simulatable coin-tossing protocol, so that only it obtains the output e, and this is played in the second round of Blum's protocol. Then M receives z from S and proves to R in zero-knowledge that it knows an accepting third message z for Blum's protocol, where the first message is committed in c and the second round is the output of the coin-tossing protocol.

4 Secure Collusion-Free ZKPoK

The second primitive we construct is a *secure* collusion free zero knowledge proof of knowledge for any language in NP. We consider the protocol $\langle \mathcal{P}, \mathcal{V}, \mathcal{M} \rangle_{\mathbb{I}}^{\mathcal{F}_{\mathtt{ZKPoK}}}$ where the functionality $\mathcal{F}_{\mathtt{ZKPoK}}$ works in one round. As public input it receives a graph G and from $\mathcal{P}$ it takes private input a 3-coloring w. If w is a valid 3-coloring of G it produces public output *true* and otherwise the public output is $(false, \lambda)$. All other inputs and outputs not specified above are the special message $\perp$. Intuitively λ models the specific round at which the proof failed. As with aborts this is important since we need to capture every bit of communication in the real world between prover and verifier and failing a proof in the i-th round can be used to signal $\log(i)$ bits.

The Protocol In figure 2 we give a protocol which is based on the zero knowledge proof of knowledge for graph 3-colorability (G3C) in [12]. To do this we make use of a statistically binding commitment scheme $(\mathtt{Com}, \mathtt{Dec})$ and a zero knowledge proof of knowledge ZKPoK. We then prove this protocol be a secure collusion free protocol for $\mathcal{F}_{\mathtt{ZKPoK}}$. Note that apart from collusion freeness this also implies all the usual properties of a zero knowledge proof of knowledge. In particular the protocol is complete; if V is honest then it is a proof of knowledge with respect to any $(\widetilde{\mathsf{P}}, \widetilde{\mathsf{M}})$; and if P is honest then it is zero knowledge with respect to any $(\widetilde{\mathsf{M}}, \widetilde{\mathsf{V}})$. We also note that one way permutations are sufficient for constructing all primitives used by this protocol.

We prove secure collusion freeness by showing that for any non-empty subset in $\{\widetilde{\mathsf{P}}, \widetilde{\mathsf{M}}, \widetilde{\mathsf{V}}\}$ of corrupt parties we can construct a set of simulators such that their joint output together with that of the honest ideal parties is (jointly) indistinguishable from that of all real world players (corrupt and honest).

Common Input: Verification keys $\texttt{svk}_{\mathsf{P}}$ and $\texttt{svk}_{\mathsf{V}}$ (from file F), graph $G = (V, E)$.
Prover P **Input:** Signature key $\texttt{ssk}_{\mathsf{P}}$, 3-coloring w of G.
Verifier V **Input:** Signature key $\texttt{ssk}_{\mathsf{V}}$.

ZK.1 M sends $\phi = \texttt{Com}(\pi, r)$ to V where π is a random permutation over $[|E|]$.
ZK.2 P selects a random permutations of $\{red, green, blue\}$ and applies it to w resulting in the 3-coloring w'. It colors G according to w' resulting in set $E' = w'(E)$ of the colors for the end points of each edge in G. P computes $\nu = \{\nu_e = \texttt{Com}((c_1, c_2)_e, r_e)\}_{e \in E'}$ where $c_1, c_2 \in \{red, green, blue\}$. P sends the set of commitments of the 3-coloring ν to M.
ZK.3 M sends $\nu' = \{\nu'_e = \texttt{Com}(\nu_e, r'_e)\}_{e \in E'}$ to V.
ZK.4 V sends $\eta \in E$, a challenge edge of G to M.
ZK.5 M sends $\eta' = \pi(\eta)$ to P.
ZK.6 P sends $((c_1, c_2)_{\eta'}, r_{\eta'}) = \texttt{Dec}(v_{\eta'})$, decommitment to the coloring of edge η' to M.
ZK.7 If $c_1 = c_2$ or either one is not in $\{red, green, blue\}$ then M sends $failed$ to V. Then it uses witness $(r_{\eta'}, r'_{\eta'}, \pi)$ to run ZKPoK with V proving that $\nu'_{\pi(\eta)}$ is a commitment of a commitment of an edge with different (valid) colored ends and moreover the edge is the one selected by V according to the permutation π committed in ϕ. If V is not convinced by the proof it rejects.
ZK.8 Repeat steps ZK.1-ZK.7 $|E|^2$ times with new random tapes. V accepts G is 3-colorable if and only if it accepts all proofs in steps ZK.7.

Fig. 2. Collusion-Free Zero Knowledge

Theorem 2. *Assuming the existence of one-way permutations the protocol in figure 2 is a secure collusion free protocol for the functionality* $\mathcal{F}_{\texttt{ZKPoK}}$.

If all parties are corrupt then there is no security proof to give. Thus now concentrate on the remaining 6 cases.

The rest of the proof is structured as follows (with some intuition behind each step in the bullets):

1. We construct 3 simulators, one for each corrupt party. The simulator for a corrupt verifier is based on the zero knowledge simulator for underlying ZKPoK for G3C. The simulator for a corrupt prover is based on the knowledge extractor for the underlying ZKPoK. The simulator for the mediator is a combination of the previous two.
2. We describe how to handle aborts and failed proofs. Essentially these are coordinated via messages to and from $\mathcal{F}_{\texttt{ZKPoK}}$ and the value λ which indicates at which round aborts and failure have occurred.
3. We prove a lemma stating that assuming the existence of one-way permutations if the mediator and either verifier or prover is corrupt then the pair of simulators and honest party have the desired output. The proof resem-

bles the proofs of correctness of the underlying simulator and knowledge extractors for the `ZKPoK` of G3C.
4. We prove a lemma stating that assuming the existence of one-way permutations, if one party is corrupt then the simulator and honest parties have the correct output distribution. These cases are special cases of the lemma proven in the previous step.
5. We prove a lemma stating that assuming the existence of one-way permutations the protocol is collusion free. This lemma covers the final case when verifier and prover are corrupt playing through an honest mediator. The proof works because an honest mediator "re-randomizes" messages which causes the view of any one corrupt party to be relatively unchanged by modifications to any other (non-aborting) parties algorithm. Thus, the simulator for the sender can use the honest receiver algorithm in its internal emulation.
6. We draw on all three lemmas to conclude the proof of theorem 2. We point out that all cases of possible sets of corrupt parties have been dealt with and so the theorem holds.

The detailed proof can be found in the full version.

5 Acknowledgments

We wish to thank Giuseppe Persiano for his invaluable contributions which helped kick start this paper. In particular the idea of a mediated communication model is due to him. Also we would like to thank Yevgeniy Dodis and Daniel Wichs for there many thoughtful discussions on the topic. Finally we are grateful for Jesper Nielsen's comments concerning the intricacies of applying MPC to game theoretic situations.

We thank the anonymous reviewers for their suggestions. The work of the authors has been supported in part by the European Commission through the IST program under Contract IST-2002-507932 ECRYPT and the one of the last author through the FP6 program under contract FP6-1596 AEOLUS.

References

[1] M. Alparone and G. Persiano. Staganography-free implementation of yao's protocol. Technical report, Unpublished, 2006.

[2] R. J. Aumann. Subjectivity and Correlation in Randomized Strategies. *Journal of Mathematical Economics*, 1(1):67–96, March 1974.

[3] B. Barak, R. Canetti, Y. Lindell, R. Pass, and T. Rabin. Secure computation without authentication. In *Advances in Cryptology - Crypto '06*, Lecture Notes in Computer Science, pages 361–377. Springer-Verlag, 2005.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1988. ACM.

[5] D. Boneh and P. Golle. Almost Entirely Correct Mixing with Applications to Voting. In *CCS'02*, pages 68–77, 2002.

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

[7] D. Chaum, P. Ryan, and S. Schneider. A Practical Voter-Verifiable Election Scheme. In *ESORICS'05*, pages 118–139, 2005.

[8] R. Cramer, R. Gennaro, and B Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *EUROCRYPT'06*, pages 103–118, 2006.

[9] P. Cramton and J. Schwartz. Collusive bidding in the fcc spectrum auctions. Technical Report 02collude, University of Maryland, Department of Economics, December 2002. available at http://ideas.repec.org/p/pcc/pccumd/02collude.html.

[10] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In *30th ACM Symposium on Theory of Computing (STOC '98)*, pages 409–418. ACM, 1998.

[11] O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM Press.

[12] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing but their Validity or all Languages in NP have Zero-Knowledge Proof Systems. *J. ACM*, 38(3):690–728, 1991.

[13] S. Izmalkov, M. Lepinski, and S. Micali. Verifiably secure devices. In R. Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 273–301. Springer, 2008.

[14] S. Izmalkov, S. Micali, and M. Lepinski. Rational Secure Computation and Ideal Mechanism Design. In *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 585–595, Washington, DC, USA, 2005. IEEE Computer Society.

[15] M. Lepinksi, S. Micali, and a. shelat. Collusion-Free Protocols. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 543–552, New York, NY, USA, 2005. ACM.

[16] M. Lepinski, S. Micali, C. Peikert, and a. shelat. Completely Fair SFE and Coalition-Safe Cheap Talk. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 1–10, New York, NY, USA, 2004. ACM Press.

[17] Daniele Micciancio and Erez Petrank. Simulatable Commitments and Efficient Concurrent Zero-Knowledge. In Eli Biham, editor, *EUROCRYPT'03*, volume 2656 of *Lecture Notes in Computer Science*, pages 140–159, Warsaw, Poland, May 2003. IACR, Springer-Verlag.

[18] C. Neff. A Verifiable Secret Shuffle and its Application to e-Voting. In *CCS'01*, pages 116–125, 2001.

[19] M. Prabhakaran, A. Rosen, and A. Sahai. Concurrent Zero-Knowledge with Logarithmic Round Complexity. In *43th IEEE Symposium on Foundations of Computer Science (FOCS '02)*, pages 366–375, 2002.

[20] G. J. Simmons. The prisoners' problem and the subliminal channel. In *CRYPTO'83*, page 5167, 1983.