Collusion-Preserving Computation

Joël Alwen[1], Jonathan Katz[2,*], Ueli Maurer[1], and Vassilis Zikas[2,**]

[1] ETH Zürich
{alwenj,maurer}@inf.ethz.ch
[2] University of Maryland
{jkatz,vzikas}@cs.umd.edu

Abstract. In *collusion-free* protocols, subliminal communication is impossible and parties are thus unable to communicate any information "beyond what the protocol allows." Collusion-free protocols are interesting for several reasons, but have specifically attracted attention because they can be used to reduce trust in game-theoretic mechanisms. Collusion-free protocols are impossible to achieve (in general) when all parties are connected by point-to-point channels, but exist under certain physical assumptions (Lepinksi et al., STOC 2005) or when parties are connected in specific network topologies (Alwen et al., Crypto 2008).

We provide a "clean-slate" definition of the stronger notion of collusion *preservation*. Our goals in revisiting the definition are:

- To give a definition with respect to *arbitrary* communication resources (including as special cases the communication models from prior work). We can then, in particular, better understand what types of resources enable collusion-preserving protocols.
- To construct protocols that allow no *additional* subliminal communication when parties *can* communicate via other means. (This property is *not* implied by collusion-freeness.)
- To support *composition*, so protocols can be designed in a modular fashion using sub-protocols run among subsets of the parties.

In addition to proposing the definition, we explore implications of our model and show a general feasibility result for collusion-preserving computation of arbitrary functionalities. We formalize a model for concurrently playing multiple extensive-form, mediated games while preserving many important equilibrium notions.

1 Introduction

Subliminal channels [28,29,30] in protocols allow parties to embed "disallowed" communication into protocol messages, without being detected. (For example, a party might communicate a bit b by sending a valid message with first bit equal to b.) The existence of subliminal channels is often problematic. In a large-scale distributed computation, for instance, subliminal channels could allow two

[*] Research supported in part by NSF grant #1111599.

[**] Supported in part by a fellowship from the Swiss National Science Foundation (Project No. PBEZP2-134445).

R. Safavi-Naini and R. Canetti (Eds.): CRYPTO 2012, LNCS 7417, pp. 124–143, 2012.

parties to coordinate their actions (i.e., to collude) even if they were not aware of each other in advance. In other settings, parties may be disallowed or otherwise unable to communicate "out-of-band," and it would be undesirable if they could use the protocol itself to convey information.

More recently, subliminal channels have arisen as a concern in the context of cryptographic implementations of game-theoretic mechanisms. Here, informally, there is a game in which parties send their types/inputs to a trusted party which then computes an outcome/result. One might naturally want to replace the trusted party with a cryptographic protocol executed by the parties [13,15,8,23,24,25,21,1,2,19,20]. Using protocols for secure multi-party computation (e.g., [17]) can preserve Nash equilibria; however, such protocols do *not* suffice for implementing general equilibria precisely because they have subliminal channels and thus enable collusion in the real world even if such collusion is impossible in the original game.

This realization has motivated the investigation of *collusion-free* protocols that do not allow subliminal communication [23,24,25,21,19,5,3,20]. Collusion-free protocols for computing non-trivial functions are impossible when parties are connected by pairwise communication channels, and so researchers have turned to other communication models. Collusion-free computation of arbitrary functionalities is possible if parties have access to a semi-trusted "ballot box" and can communicate publicly via (physical) envelopes [24,21,19,20], or if parties are connected (via standard communication channels) to a semi-trusted entity in a "star network" topology [5,3].

1.1 A New Definition: Collusion Preservation

The works of Izmalkov et al. [24,21,19,20] and Alwen et al. [5,3] give incomparable definitions of collusion freeness, each tailored (to some extent) to the specific communication models under consideration. We revisit these definitions, and propose a stronger notion called *collusion preservation.* Intuitively, collusion-free protocols ensure that parties can communicate no more with the realizing protocol then using *only* the ideal functionality, whereas collusion-preserving protocols provide a stronger guarantee: the parties can communicate no more when running the protocol and using arbitrary fixed external channels, than they could using the ideal functionality and the same external channels. Our aim here is to provide a *clean, general-purpose* definition that better handles *composition*, both when collusion-preserving protocols are run as sub-routines within some larger protocol, as well as when collusion-preserving protocols are run concurrently with arbitrary other protocols (whether collusion-preserving or not). In what follows, we give an overview of our definition and expound further on the above points.

Overview of our definition. We follow the simulation-based definitional paradigm used by Alwen et al. [5,3]: In the real-world execution of a protocol, different adversaries can corrupt different parties. Importantly, these adversaries cannot communicate directly; instead, all parties are connected in a "star network" topology with a semi-trusted mediator. (This is in contrast to usual cryptographic definitions, which assume a "monolithic" adversary who controls all corrupted parties and can coordinate their actions.) Two notions of stand-alone security [16] are then defined, depending on whether or not the mediator is honest:

1. Conditional collusion freeness: When the mediator is honest, collusion freeness is required.
2. Fallback security: When the mediator is dishonest, we cannot hope for collusion freeness any more. Nevertheless a strong notion of security can be achieved; namely real/ideal adversaries are allowed to communicate arbitrarily, and the protocol is required to satisfy the standard (stand-alone) security definition [16].

We strengthen and extend the definition of collusion freeness in several ways. Firstly, rather than considering a specific "star network" topology with a special party (the mediator) in the center [5,3], or the specific physical assumptions of [24,21,19,20], we consider a general *resource* to which the parties have access in the real world. This resource is the only means of "legal" communication in the real world (though as we will see in a moment there may be other "illicit" means of communication available). In addition to being more general, our definition allows us to characterize the minimal properties of resources which achieve collusion-preserving computation. Secondly, we formulate our definitions in a universally composable (UC) [9] fashion, where there is an environment controlling the entire execution.[1] This has significant ramifications, since the environment *itself* can now act as a communication channel for the adversaries. If the environment chooses to allow no communication between the adversaries, then our definitions essentially "default" to the previous setting of collusion freeness. Crucially, however, if the environment allows the adversaries to communicate c bits of information "for free", then a collusion-preserving protocol ensures that the adversaries cannot communicate more than c bits (on top of the communication allowed by the ideal functionality) by running the protocol in the presence of the stated resource. (We show below a simple counter-example demonstrating that collusion freeness does not imply collusion preservation.) Moreover, we prove a universal composition theorem, thereby improving upon the results of [5,3], which do not claim nor realize any form of composition, as well as the results of [24,21,19,20] which obtain only a limited sort of composition; see below.

Collusion preservation is stronger than collusion freeness. We motivate the need for a composable definition with an example: Consider a protocol π that is collusion-free in the mediated (star network) setting of [5,3]. We obtain a new protocol π', identical to π except for the following two modifications to the mediator's behavior (where λ is the security parameter):

1. The mediator takes a special message $m \in \{0,1\}^{2\lambda}$ from P_1. In response, the mediator chooses a random $r \in \{0,1\}^{\lambda}$, sends it to P_1, and stores (r, m).

[1] Actually, we use the *generalized* UC (GUC) framework [10] as our starting point.

2. The mediator takes a special message $r' \in \{0,1\}^\lambda$ from P_2. If the mediator has a stored tuple of the form (r', m), it sends m to P_2 (and otherwise simply ignore r).

It is not hard to see that π' remains collusion-free: intuitively, this is because P_2 can guess $r' = r$ with only negligible probability.[2] However, π' is *not* collusion preserving. Specifically, if P_1 and P_2 have access to a λ-bit channel then they can use π' to communicate 2λ bits!

The above counter-example can be interpreted in several ways. One could imagine that π' is run in a setting in which P_1 and P_2 have access to a physical channel that only allows communication of λ bits. Alternately, the parties might be running π' while they are concurrently running some *other* protocol that is not collusion-free and enables the parties to (subliminally) communicate λ bits. Either way, the implication is the same: a collusion-free protocol may potentially allow additional communication than the corresponding ideal specification.

In the following we give an overview of the main results on the paper.

Protocol composition. The protocols of Izmalkov et al. [24,21,19,20] are collusion-free only when *at least one party running the protocol is honest*. The reason is that in their communication model parties have *the ability* to communicate arbitrary information (but their protocols guarantee that if an honest party is watching then it will detect any such communication). This limitation may not appear to be a problem, since one typically does not care to provide any guarantees once all parties are malicious. It becomes a problem, however, when collusion-free protocols are used as sub-routines within some larger protocol. Consider, for example, a collusion-free protocol Π for three parties $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$ in which each pair of parties runs some collusion-free sub-protocol π between themselves. If P_1 and P_2 are malicious, then π may provide no guarantees which means that they may now be able to communicate an unlimited amount of information; this could clearly be problematic with regard to the "outer protocol" Π.[3]

Necessary assumptions. The main criticism on the mediated model of [5,3] concerns the strength of the mediator as an assumption. In particular, an open question is whether or not one can reduce the power/complexity of the mediator without sacrificing collusion-freeness. Justifying the tightness of our model, we show that any recourse which allows for realizing arbitrary (well-formed) functionalities in a collusion preserving (or even collusion-free) manner satisfies three essential properties referred to as *isolation, independent randomness*, and *programmability*. Intuitively, the mediator is one of the simplest digital resources having all these properties, simultaneously. Note that the physical assumptions of [23,24,25,21] also implicitly satisfy these properties.

[2] In particular the simulators for π' can behave just as for π with the only modification that the simulator for P_1 responds with a random message r when it receives the special message m from it's adversary.

[3] Izmalkov et al. implicitly avoid this issue by having *every* party observe all sub-protocols that are run.

General feasibility with GUC fallback. Complementing and motivating our definitional work, we provide a completeness result for strong realization of a large class of functionalities. More concretely, for any functionality in this class we provide a protocol compiler and a particular resource which satisfies a universally composable version of the security definition from [3]: the compiled protocol provides Collusion Preserving (CP) security when executed with this particular resource, and, as a strong fallback, when executed with an arbitrary resource, it achieves GUC-type security, i.e., emulation by "monolithic" simulators.

Synchronization pollution. The types of correlation ruled out by collusion-free protocols fall into two categories: the first is due to "subliminal channels" (i.e. means of communication not present in the ideal world) and the second is due to "randomness pollution" (i.e. publicly visible correlated randomness generated during the protocols execution). We observe and mitigate, a new type of correlation between split adversaries (called *synchronization pollution*) which is crucial to achieving concurrent (and thus universal) composability. Synchronization pollution arises for example, in settings where a multi-round (or, more generally, a *multi*-stage[4]) protocol is used to CP-realize a non-reactive functionality but the parties have no means, external to the protocol, of synchronizing their actions, e.g., there are no synchronized clocks. In fact, the problem can arise for any security notion with split adversaries, where the relative order of events occurring in different parties interfaces is observable. Intuitively, an environment witnessing the execution of such a protocol could keep track of these events *as they occur* (say by instructing the adversaries to corrupt all players, run the honest protocol and announce each event as it takes place). However, if the ideal functionality is used instead, a priori the simulators have no means for coordinating the simulation of the events in the correct order as the functionality is good only for a single round of communication and otherwise the simulators have no further means of synchronizing their outputs. In the full version we demonstrate how the above issue can lead to real attacks when protocols are executed concurrently.

We resolve this issue by considering ideal functionalities with a means to provide a (minimal) amount of synchronization, namely output-synchronization. That is for an interactive CP protocol, at the very least, simulators are allowed to coordinate when they produce the final output of the computation for the environment. Surprisingly we are able to show that this is the *only* additional synchronization required for good split-simulation in the ideal world. Intuitively this is because our protocol hides all other properties *and events* in the execution (such as at which stage of the protocol the execution is currently in) from adversaries even if they corrupt all parties and their ongoing views are combined by the environment. To the best of our knowledge, this is the first protocol to exhibit such behavior. For example in the case of the sequence of works [24,21,19,20] all protocols contain multiple publicly verifiable (i.e. visible) events. Therefore

[4] Intuitively, by multi-stage we mean a protocol which results in at least one triplet of events (E_1, E_2, E_3) such that each is noticeable by at least one party and which can only occur in a fixed order $E_1 \to E_2 \to E_3$.

adversaries present *during* the execution of such protocols obtain significantly more then mere output synchronization. Moreover, to attain concurrent composability the ideal worlds would have to be augmented with some means for simulators to have the same level of synchronicity afforded by those events (similar to the synchronization wrapper). On the other hand, in previous works dealing with the mediated model [5,3] the current stage of the protocol execution is not hidden from corrupt players. We remark that for both lines of work, the protocols remain sequentially composable because once the execution has completed, the simulators can generate appropriate transcripts atomically. The problem arises only if the transcripts are inspected by an on-line distinguisher.

Implications for game theory. We show how to use the results of the CP framework to adapt the traditional models of algorithmic game theory [27] and mechanism design to a more practical setting. First we generalize the standard stand-alone model of game play for (extensive form computational) games to a concurrent setting in which multiple games with different player sets using different mechanisms are being played in an arbitrarily interleaved manner. Next we introduce a strong notion of equivalence between sets of concurrent games which we call *isomorphic game sets*[5]. Intuitively a pair of isomorphic game sets have equivalent strategies in that they induce approximately the same outcome (and payoffs). Moreover for any strategy a player may use in one game set and any action taken in the course of that strategy it is easy for the player to compute the equivalent action(s) in the isomorphic game set regardless of which strategies are used by all other players involved and even of the strategy which dictated to play the original action. We provide evidence that this transformation between actions of isomorphic game sets preserves, in particular, many desirable notions of stability such as k-resilience [7], correlated equilibria [6], dominant strategies and t-immunity [2].

Next we apply our feasibility results for the CP framework to make a second important step in bringing the field of mechanism design closer to a practical setting. Leveraging the GUC fallback property we show how to significantly reduce the type of trust placed in a certain class of mechanisms. Traditionally game theorists place complete trust in mechanisms not only to enforce such properties as the isolation of players and fairness of output distribution, but also to compute outcomes correctly and preserve the players' privacy.[6] We use the results from the cryptographic section of this work to show how to completely remove the latter two types of trust while still obtaining an isomorphic game set. Roughly speaking, this is done by replacing the original mechanism with a CP secure protocol running over a "less trusted" mechanism such that the resulting game is equivalent to the original even with respect to an arbitrary set of concurrently running games. The fact that the isomorphic strategies can be computed locally and automatically by each player with out considering other

[5] Our notion of isomorphic game sets is closely related to that of [26].

[6] For example in a poker game it is implicitly assumed that the dealer deals from a uniform random deck and that they don't reveal the cards exchanged by one player to another.

players' strategies permits both mechanism designer and player to operate in the more traditional (presumably cleaner) "fully trusted" setting while actual game play occurs in the more practical setting with reduced trust.

In comparison to results of [18,19,20] which provide information theoretic-equivalence between games using an unconventional model of computation, the results in this paper provide only computational equivalence but use a standard computational model. However, their notion of composition is weaker in two ways. Conceptually it is not scalable but more concretely it seems to allow for only rather limited notion of concurrency. In particular protocols that implement a mechanism must be run atomically with respect to actions in any concurrent games. In contrast, the notion obtained in this paper is fully UC composable in the more traditional sense. On the other hand, while our protocols prevent signaling via aborts as in [18], they do not provide the full robustness to aborts of [19,20]. Finally, the amount of randomness in the public view of our protocols is limited to a single pre-computation round which can be run before types are distributed. From that point on there is no further "randomness pollution". This is similar to [22], better then [18] (where even executions of a protocol produce randomness pollution) but weaker then [19,20] which do not produce any randomness pollution at all.

Relation to abstract cryptography [26] and to UC with local adversaries [12]. Collusion-preserving computation can be described in the Abstract Cryptography (AC) framework by Maurer and Renner [26], but this is beyond the scope of this paper. Apart from being stated at an abstract level, two relevant aspects of AC in our context are that there is no notion of a central adversary (and simulators are local) and that all resources are modeled explicitly, which allows to capture the absence of resources (e.g. communication channels). In concurrent and independent work with ours, Canetti and Vald [12] also consider the question of extending the notion of collusion freeness (local adversaries/simulators) to the universally composable setting. Although many of our results can be proved also in their framework, the two models have considerable conceptual and formal differences. We refer to the full version of our work [4] for a discussion of the main diferences between the two approaches.

2 Collusion-Preserving Computation

In this section we define our framework for investigating universally composable collusion freeness, namely *collusion-preserving* computation. On the highest level the idea is to combine the strong composability properties of the GUC framework of [10] with the model of split simulators along the lines of [3].

2.1 Preliminaries and Notation

We denote by $[n]$ the set $\{1,\ldots,n\}$ (by convention $[0]=\emptyset$) and for a set $\mathcal{I}\subseteq[n]$ we denote by $\overline{\mathcal{I}}$ the set $[n]\setminus\mathcal{I}$. Similarly, for element $i\in[n]$ we write $\overline{i}$ to denote the set $[n]\setminus\{i\}$. Using this notation we denote by $\mathsf{A}_{\mathcal{I}}$ a set of ITMs $\{\mathsf{A}_i\}_{i\in\mathcal{I}}$.

For input tuple $x_{\mathcal{I}} = \{x_i\}_{i \in \mathcal{I}}$ we write $\mathsf{A}_{\mathcal{I}}(x_{\mathcal{I}})$ to denote that for all $i \in \mathcal{I}$ the ITM A_i is run with input x_i (and a fresh uniform independent random tape). We assume a passing familiarity with the GUC framework and refer to the full version of this paper [4] for a description of the main features we use.

An intuitive description. We include an informal description of our CP framework and discuss its basics; a complete description can be found in [4]. Starting from the GUC model we make the following modifications:

Split adversaries/simulators: Instead of a monolithic adversary/simulator we consider a set of n (independent) PPT adversaries $\mathsf{A}_{[n]} = \{\mathsf{A}_i \ : i \in [n]\}$, where A_i correspond to the adversary associated with the player i (and can corrupt at most this party). Moreover, we ask that for each $\mathsf{A}_i \in \mathsf{A}_{[n]}$ there exists an (independent) simulator Sim_i.

Corrupted-set independence: We also require that the simulators do not depend on each other. In other words the code of simulator Sim_i is the same for any set of adversaries $\mathsf{A}_{[n]}$ and $\mathsf{B}_{[n]}$ as long as $\mathsf{A}_i = \mathsf{B}_i$.

Resources, shared functionalities, and exclusive protocols. The main difference between a CP functionality $\mathcal{R}$ and a GUC one is that besides the n interfaces to the (honest) parties it also has interfaces to *each* of the n adversaries $\mathsf{A}_1, \ldots, \mathsf{A}_n$ (see Figure 1). In other words rather then n interfaces a CP functionality has $2n$ interfaces.[7] Moreover, similar to the GUC framework (but in contrast to plain UC) we distinguish between two types of functionalities: *resources* (i.e., functionalities that maintain state only with respect to a single instance of a protocol) which we denote with capital calligraphic font as in "$\mathcal{R}$" and *shared functionalities* (i.e., functionalities that can maintain state across protocol instances) which we denote with an additional over-line as in "$\bar{\mathcal{G}}$".

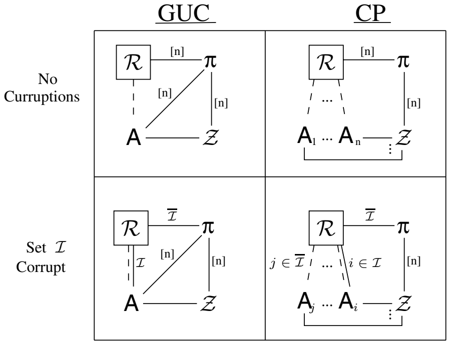
GUC
CP
No Curruptions
Set $\mathcal{I}$ Corrupt
$\mathcal{R}$
[n]
π
A
$\mathcal{Z}$
A_1 ... A_n
$\mathcal{I}$
$\bar{\mathcal{I}}$
$j \in \bar{\mathcal{I}}$
$i \in \mathcal{I}$
A_j ... A_i

Fig. 1. Corruption of player set $\mathcal{I} \subseteq [n]$: GUC model vs. CP model. (Setup functionalities are left implicit.)

The $\mathcal{R}$-hybrid world. A CP execution in the $\mathcal{R}$-hybrid world is defined via a straightforward generalization to the analogous GUC execution. We denote the output of the environment $\mathcal{Z}$ when witnessing an execution of protocol $\pi := \pi_{[n]}$

[7] Intuitively, the reason for having n additional interfaces (one for each adversary) is that adversaries should not communicate with each other, therefore they cannot share the same interface; nevertheless, as in all composable frameworks, each of them should be able to communicate with the assumed resource, e.g., for scheduling delivery of messages to the corresponding party.

attacked by adversaries $\mathsf{A} := \mathsf{A}_{[n]}$ in the $\mathcal{R}$-hybrid model as $\text{CP-EXEC}^{\mathcal{R}}_{\pi,\mathsf{A},\mathcal{Z}}$. Finally, we say a protocol π is *$\mathcal{R}$-exclusive* if it makes use of no other resources (shared or otherwise) then $\mathcal{R}$.

Bounding the number of calls to resources. A primary difference between how executions in a $\mathcal{R}$-hybrid world are modeled in the GUC and CP frameworks is that in CP parties can communicate with at most a *single* instance of $\mathcal{R}$. This is in contrast to all other UC like models where the $\mathcal{R}$-hybrid world is understood to mean that parties can make as many calls as they wish to $\mathcal{R}$ instantiating a new copy for each new invocation of $\mathcal{R}$. Note that, for a composable notion with split adversaries fixing the number of instances of functionalities/resources available to adversaries is in fact *crucial* for capturing the desired intuition of collusion freeness. For example a primary motivation of this work is to provide a way for reducing trust on the mediators used in mechanism design by providing protocols which can be used to replace the interaction with the mediator. If we do not restrict the number of instances of the mechanism with which parties can interact then there is no meaningful way to capture a game which calls for only a single instance. Unless explicitly stated otherwise, in the present work, whenever we write a functionality we assume a single instance of it. For a longer discussion we refer to the full version [4].

Definition 1 (Collusion-Preserving Computation). *Let $\bar{\mathcal{G}}$ be a setup, $\mathcal{R}$ and $\mathcal{F}$ be n-party resources, π be a $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol and ϕ be a $\{\bar{\mathcal{G}}, \mathcal{F}\}$-exclusive protocol (both n-party protocols). Then we say that π* collusion-preservingly (CP) emulates ϕ in the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-hybrid world*, if there exists a collection of efficiently computable transformations $\mathsf{Sim} = \mathsf{Sim}_{[n]}$ mapping ITMs to ITMs such that for every set of adversaries $\mathsf{A} = \mathsf{A}_{[n]}$, and every PPT environment $\mathcal{Z}$ the following holds:* $\text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\pi,\mathsf{A},\mathcal{Z}} \approx \text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{F}}_{\phi,\mathsf{Sim}(\mathsf{A}),\mathcal{Z}}$.

Realization, reductions, and the "$\sqsubseteq$" notation. We use the following notation (for details see [4]). If for functionality $\mathcal{F}$, an $\mathcal{R}$-hybrid protocol π CP-emulates $\mathcal{F}$ [8] then we say that π *realizes $\mathcal{F}$ (in the $\mathcal{R}$-hybrid world)*, and denote it by $\mathcal{F} \sqsubseteq^{CP}_{\pi} \mathcal{R}$, which can intuitively be read as "$\mathcal{F}$ CP-reduces to $\mathcal{R}$ via protocol π". By omitting π in this notation we denote simply the existence of some protocol for which the relation holds. We also use "$\sqsubseteq^{GUC}$" to denote the analogous relation but for *GUC*-realization. To simplify notation and maintain consistency with previous UC-type works, whenever an explicit protocol for the honest players is missing in the CP-EXEC notation then it is implicitly assumed that they are running the dummy $\mathcal{F}$-hybrid protocol $\mathcal{D}^{\mathcal{F}}$ that forwards all it inputs from $\mathcal{Z}$ to $\mathcal{F}$ and vise-versa. For example we might write $\text{CP-EXEC}^{\mathcal{F}}_{\mathsf{A}_{[n]},\mathcal{Z}}$ when the honest players are running $\mathcal{D}^{\mathcal{F}}_{[n]}$.

Composition theorem. As a main motivation for the CP model we put forth the goal of providing a formal and rigorous notion of composability for collusion-free security. We formalize a strong (universally) composable property of CP security

[8] Formally we would write $\mathcal{D}^{\mathcal{F}}$ instead of $\mathcal{F}$.

in the following theorem. The proof can be found in the full version [4]. As part of the proof we provide a useful tool for proving CP security of protocols in the form of a much simplified security notion which proves to be almost as powerful yet far easier to work with.

Theorem 1 (Composition). *Let $\mathcal{R}$ be an arbitrary resource and $\bar{\mathcal{G}}$ be a global setup (i.e. shared) functionality. Let ρ, π and ϕ be n-party protocols in the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-hybrid world such that π and ϕ are $\bar{\mathcal{G}}$-subroutine respecting [10]. If π CP-emulates ϕ and ρ uses ϕ as a subroutine then $\rho^{\pi/\phi}$ CP-emulates ρ in the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-hybrid world.*

Relations to existing security notions. The weaker notion of collusion free computation [4,22] can be described as the special case of CP which assumes off-line environment, i.e., the environment does not interact with the adversaries during the computation. In the full version [4] we prove a pair of lemmas relating CP results to matching GUC results. The first formalizes the intuitive claim that CP security is at least as strong as GUC security via a lemma stating that CP realization essentially implies GUC realization. A bit more precisely, we describe a natural mapping of CP functionalities to analogous GUC functionalities $\mathcal{F} \longmapsto [\mathcal{F}]$. Then we show that if a protocol CP realizes $\mathcal{F}$ in the $\mathcal{R}$-hybrid world, then *the same* protocol executed in the analogous GUC $[\mathcal{R}]$-hybrid world GUC realizes the analogous GUC functionality $[\mathcal{F}]$. The second establishes the other direction, i.e., translating GUC security statements to statements in CP, and is useful as a primary building block for our feasibility. To this end we define the $\mathcal{R}_{\mathsf{ins}}$ functionality which is an adaptation to the CP setting of the standard complete network of UC insecure channels. The main difference is that messages from say player P_i to P_j (as depicted in Figure 2) are first given to A_i and then A_j before it is delivered to P_j . Adversaries can modify (or not deliver) the message at will.

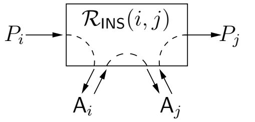

Fig. 2. Insecure Channel with split adversaries

3 Necessary Assumptions for Collusion Preservation

Having defined the CP framework and verified it's composition properties, we turn to the next major goal of this work: to provide a resource with which we can (constructively) CP-realize as many functionalities as possible. Ideally we would like to obtain a CP-complete resource, namely one from which *any* reasonable functionality can be realized. Indeed, in the next section we describe just such a resource which we call the *mediator*. However, we must first justify the seemingly strong assumptions we will make when defining the mediator by showing their necessity. To this end, we demonstrate three necessary properties a given resource must have for it to be CP-complete. As corollaries to these results we rule out realizing large classes interesting functionalities using virtually all common communication resources such as fully connected networks and broadcast channels. Beyond this, due to their generality, we believe that given a target ideal functionality $\mathcal{F}$ (such as an auction mechanism or voting functionality), these results provide significant insights into the minimal assumptions about real world communication channels which can be used to CP realize $\mathcal{F}$. In the following we sketch these properties, and refer to the full version of this paper for a formal description using the language of our framework

Correlation is not free. Fundamentally, models with a monolithic adversary already allow *perfect* coordination between all interfaces connected to corrupt players. However, when adversaries are split (and a priori isolated) such coordination is not given to adversaries for free anymore. So the security requirements in executions where all players are corrupt are still non-trivial. Instead bounds may still be required on the amount of coordination between the behavior on different interfaces.[9] By analyzing the implications of the security requirements in such settings we show the necessity of the following properties.

1. (Isolation) Consider a statement of the type $\mathcal{F} \sqsubseteq^{CP} \mathcal{R}$. Intuitively this holds only if $\mathcal{R}$ can isolate (corrupt) players as much as $\mathcal{F}$. We formalize this by showing that (roughly speaking) for any amount of communication that adversaries can obtain from $\mathcal{F}$, it must be that $\mathcal{R}$ does not allow any more communication. More specifically, we define an extremely weak ideal channel $\mathcal{C}$ and show that if $\mathcal{F}$ can be used to obtain $\mathcal{C}$ by collaborating adversaries controlling *all* interfaces to $\mathcal{F}$, then $\mathcal{R}$ cannot be used to obtain such a channel with more bandwidth. Given how weak $\mathcal{C}$ is and how much power adversaries have over $\mathcal{F}$ for obtaining it somehow, this result has some far reaching consequences with respect to standard communication models. Not only are they provably not CP-complete but in fact they can not be used to realize almost *any* interesting functionalities (other then themselves).
2. (Randomness) The second property states that any CP-complete resource must have it's own internal randomness upon which it's output depends. To prove this, we show that for any resource $\mathcal{R}$ which can be used for CP realizing ideal coin flipping as well as key agreement, $\mathcal{R}$ can not be deterministic.
3. (Programmability) Finally we define the notion of a programmable resource and show that any CP-complete resource must be programmable. Intuitively a programmable resource can be thought of as being instantiated with a special parameter upon which it's behavior depends in a non-trivial way. By non-trivial we mean that for at least one pair of possible values of the parameter the resulting pair of behaviors can not be CP-reduced to each other. Indeed the mediator resource in the next section is programmable and our protocol compiler not only outputs a protocol but also the parameters with which the mediator must be instantiated.

[9] In a stand-alone setting one might ask why this is even an interesting case (for example the stand-alone notion of [24] explicitly rules it out). But for a composable security notion (for example with the application of modular protocol design in mind) it is vital to consider such executions. Moreover in the context of game theory where there is no such notion of "honest" behavior all players behave as adversaries in a cryptographic sense.

4 GUC Fallback Security

Without any further requirements CP security, as defined in Definition 1, can be easily achieved from an appropriate resource. Indeed, because the resource is completely trusted it could trivially be the functionality we are trying to compute. However, such trust is a rare commodity and so one might ask for a better solution. To that end we add a second property which we call "fallback security". The goal is to capture what kind of security remains if the protocol is run not with the resource it was designed for but with an arbitrary (potentially malicious) resource instead. Note that the trivial solution provides essentially no fallback security at all. However, we will show, perhaps somewhat surprisingly, that in fact a very strong type of security can still be achieved; namely GUC-like realization.

Definition 2 (CP-realization with GUC fallback). *For setup $\bar{\mathcal{G}}$, functionalities $\mathcal{F}$ and $\mathcal{R}$, we say that a protocol π* CP-realizes a functionality $\mathcal{F}$ with GUC fallback in the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-hybrid model *if it has the following two properties:*

- CP SECURITY: π *CP-realizes $\mathcal{F}$ when using $\mathcal{R}$, i.e., $\{\bar{\mathcal{G}}, \mathcal{F}\} \sqsubseteq_{\pi}^{CP} \{\bar{\mathcal{G}}, \mathcal{R}\}$*
- GUC FALLBACK: *For any efficient resource $\mathcal{R}^*$ the protocol π still GUC-realizes $\mathcal{F}$, i.e., $\forall \mathcal{R}^* \ : \ \{\bar{\mathcal{G}}, \mathcal{R}_{\mathsf{ins}}, \mathcal{F}\} \sqsubseteq_{\pi}^{CP} \{\bar{\mathcal{G}}, \mathcal{R}^*\}$*

Recall that the (G)UC plain model implicitly assumes $[\mathcal{R}_{\mathsf{ins}}]$. Thus, by applying our CP-to-GUC translation from Section 2 (and omitting the redundant $[\mathcal{R}_{\mathsf{ins}}]$ term) we have that GUC fallback security directly implies: $\{[\bar{\mathcal{G}}], [\mathcal{F}]\} \sqsubseteq_{\pi}^{GUC} \{[\bar{\mathcal{G}}], [\mathcal{R}^*]\}$. Intuitively this means that π run with (even a single instance of) $\mathcal{R}^*$ and arbitrarily coordinated adversaries still GUC realizes $[\mathcal{F}]$.

We note that as an alternative, by restricting the class of resources $\mathcal{R}^*$ for which the fallback is desired one could, in turn, hope for weaker but still non-trivial fallback properties. This could reflect the real world settings where moderate guarantees about the behavior of the resource are given but it is still undesirable to completely trust the resource. In this sense the feasibility result in this work demonstrates that, at the very least, GUC fallback can be achieved even when no moderate guarantees of any type are made.

5 A General Feasibility Result

We are now ready to state and prove a general feasibility result. Roughly speaking we describe an (efficient) programmable resource $\{\mathsf{M}_{\mathcal{F}}\}_{\mathcal{F}\in\{0,1\}^*}$, called the *mediator*, parameterized by descriptions of functionalities, such that for any $\mathcal{F}$ in a large class of functionality, we can design a protocol π (using setup $\bar{\mathcal{G}}$) which CP-realizes $\mathcal{F}$ with GUC fallback in the $\{\bar{\mathcal{G}}, \mathsf{M}_{\mathcal{F}}\}$-hybrid model.

Output-synchronized functionalities. In contrast to previous frameworks, because we consider split simulators the environment $\mathcal{Z}$ has an additional means for distinguishing between executions. Briefly, $\mathcal{Z}$ can measure the amount of on-line synchronization that taking part in an execution provides to sets of adversaries. In contrast all actions taken by monolithic adversaries during an execution are already inherently perfectly synchronized so no such strategy exists in (G)UC frameworks. We address this by introducing the class of *output-synchronized* functionalities. For a arbitrary functionality $\mathcal{F}$ we write $\widehat{\mathcal{F}}$ to denote the "output synchronized" version of $\mathcal{F}$. That is $\widehat{\mathcal{F}}$ consists of a wrapper (the *synchronizing shell*) and inner functionality $\mathcal{F}$. The synchronizing shell works as follows: On it's first activation, $\widehat{\mathcal{F}}$ sends a request to one of the simulators (e.g. the one with the smallest ID) to acquire the index of the desired output round. Let R denote the response of this simulator (if no valid R is received then set $R := 1$). Subsequently, all inputs are forwarded to $\mathcal{F}$. Finally, outputs of $\mathcal{F}$ are not immediately delivered to their recipient. Rather they are recorded are given only *upon request* from the recipient and only after R subsequent *complete rounds* have been observed (each output-request which is issued before that is answered by a default message $\perp$). By a complete round we mean a sequence of at least one activation from every player (or the corresponding simulator) in an arbitrary order. Intuitively this modification provides minimal synchronization between adversaries. For a more detailed discussion of output synchronization we refer to [4].

To formalize our main feasibility theorem we introduce the following terminology (for a more accurate description we refer to [4]): A functionality is *well-formed* [11] if its behavior is independent of the identities of the corrupted parties, and it is *aborting* if it accepts a special input `ABORT` from corrupted parties, in which case it outputs `ABORT` to all parties (note that this is the only type of functionalities we can compute while tolerating a corrupted majority). A protocol is said to be *setup off-line* if it precedes all other computation and communication by it's only interaction with the setup $\bar{\mathcal{G}}$[10]. We call a setup (i.e. a CP shared functionality) $\bar{\mathcal{G}}$ *GUC–AuthComplete* if in the $[\bar{\mathcal{G}}]$-hybrid world:

1. There exists a setup off-line protocol which GUC realizes authenticated channels from insecure channels
2. Every well-formed functionality can be GUC securely realized (in the standard GUC model which assumes authenticated channels) by a setup off-line protocol.

We note that, assuming static adversaries,[11] one such setup is the Key-Registration with Knowledge (KRK) functionality of [10,14] when viewed as a CP shared functionality (we refer to the last paragraph of the current section for details). In particular the results of [14] imply Property (1) and the results of [10] imply Property (2). For the case of adaptive adversaries, one needs to use a setup which is stronger than KRK; as demonstrated in [14], a sufficient setup in this case is their so called Key-exchange functionality.

[10] All known protocols (in particular the protocols of [10,14]) are of this form.

[11] A static adversary chooses the parties to corrupt at the beginning of the protocol.

Ideally, we would like to state our feasibility result for *any* (albeit efficient) functionality $\mathcal{F}$. More realistically we require that $\mathcal{F}$ have the (standard) properties sketches above, i.e., be well-formend and aborting.

Theorem 2 (General Feasibility Theorem). *Let $\bar{\mathcal{G}}$ be a GUC-AuthComplete CP setup. Then there exists a programmable resource $\mathsf{M} = \{\mathsf{M}_{\mathcal{F}}\}$ such that for every well-formed aborting functionality $\mathcal{F}$ there exists a protocol π which CP-realizes $\widehat{\mathcal{F}}$ with GUC fallback in the $\mathsf{M}_{\mathcal{F}}$-hybrid model.*

We prove the theorem constructively, i.e., by describing an efficient compiler mapping a given aborting well-formed functionality $\mathcal{F}$ to a protocol $\mathtt{CP}(\pi)$ and parameters for resource M. Note that although we have assumed that $\widehat{\mathcal{F}}$ is output-synchronized, the GUC property holds, for the same protocol $\mathtt{CP}(\pi)$, even for the non-synchronized functionality, i.e., the one that results by removing from $\widehat{\mathcal{F}}$ the synchronizing shell. The reason is that in the GUC-fallback setting the simulators can synchronize output generation by using the insecure channels $\mathcal{R}^*_{\mathsf{ins}}$. The proof of the theorem proceeds in two steps: (1) In the "bootstrapping" step (Lemma 1) we show how to obtain from a GUC-AuthComplete setup $\bar{\mathcal{G}}$, a setup off-line protocol π which CP realizes $\mathcal{F}$ using insecure channels. (2) Then, in the "adding fallback" step (Lemma 2), we show how to compile π into protocol $\mathtt{CP}(\pi)$ and resource $\mathsf{M}_{\mathcal{F}}$ which CP realize $\widehat{\mathcal{F}}$ with GUC fallback.

Lemma 1. *Let $\bar{\mathcal{G}}$ be a GUC-AuthComplete CP setup. Then for every well-formed aborting CP functionality $\mathcal{F}$ there exist a setup off-line protocol π such that $\{\bar{\mathcal{G}}, \mathcal{R}_{\mathsf{ins}}, \mathcal{F}\} \sqsubseteq^{CP}_{\pi} \{\bar{\mathcal{G}}, \mathcal{R}_{\mathsf{ins}}\}$.*

The proof is simple and essentially verifies that the conditions for GUC-to-CP translation (Section 2) are met; for details we refer to [4]. The bulk of the work for proving Theorem 2 lies in proving the following lemma which states that if a protocol exists that CP realizes $\mathcal{F}$ from insecure channels then there exists a protocol and resource which additionally have GUC fallback. Together with the previous lemma the theorem follows directly.

Lemma 2. *Let $\bar{\mathcal{G}}$ be a GUC-AuthComplete CP setup, let $\mathcal{F}$ be a well-formed aborting functionality, and let π be a setup off-line protocol such that $\{\bar{\mathcal{G}}, \mathcal{R}_{\mathsf{ins}}, \mathcal{F}\} \sqsubseteq^{CP}_{\pi} \{\bar{\mathcal{G}}, \mathcal{R}_{\mathsf{ins}}\}$. Then there exists an efficient resource $\mathsf{M}_{\mathcal{F}}$ (the "mediator") and protocol $\mathtt{CP}(\pi)$ such that $\mathtt{CP}(\pi)$ CP realizes $\widehat{\mathcal{F}}$ with GUC fallback in the $\mathsf{M}_{\mathcal{F}}$-hybrid model.*

Proof idea. The proof is constructive, i.e., we show how to construct $\mathtt{CP}(\pi)$ and $\mathsf{M}_{\mathcal{F}}$. As a starting point we begin with the ideas of the protocol of [3]. We simplify it both in terms of exposition (crystalizing the underlying technique of assisted emulation) and by removing the need for emulation of broadcast. Then we adapt the resulting protocol such that the (rather minimal) synchronization wrapper suffices for full simulation (an issue which does not arise in the stand-alone setting of [3]). This latter step is done by minimizing the amount of synchronization the protocol affords players by adding explicit instructions for handling dummy steps. The result being that adversaries remain completely unaware of which round they are in even in an asynchronous environment where they may be activated many times (or just once) during any given round. On the highest level the idea is to have the mediator $\mathsf{M}_{\mathcal{F}}$ emulate an execution π "in it's head" such that the players are oblivious to everything but their input and output of π. Intuitively this guarantees the CP realization property of Definition 2.

To obtain the GUC fallback property: For each $i \in [n]$ the state of the emulated π_i is *shared* between P_i and $\mathsf{M}_{\mathcal{F}}$ such that $\mathsf{M}_{\mathcal{F}}$ can not alter it without the help of P_i yet both parties learn nothing about the actual value of the state. For this purpose we describe a pair of 2-party SFE's run between P_i and $\mathsf{M}_{\mathcal{F}}$ which allow for state of π_i to be updated as dictated by an honest execution of π. Intuitively it is the hiding of the states from $\mathsf{M}_{\mathcal{F}}$ and the security of the SFE's which ensure GUC fallback. While enjoying a significantly stronger security notion, our compiler is also conceptually simpler than the one in [3]. This stems from our assumption that the input protocol π operates over a *network of insecure channels* rather then the broadcast channel used in [3]. As a result, (1) our compiler does not need to worry about the parties authenticating their messages, as this is taken care of by π, and (2) we do not need specifically describe a "mediated" broadcast protocol as in [3]. We refer to the full version of this paper for a detailed description of the protocol construction and the proof of security.

5.1 A Concrete Instance

Thus far the results have been stated for an abstract GUC-AuthComplete setup. For a concrete instance we can use the Key Registration with Knowledge (KRK) setup of [10,14] (assuming a static adversary). Recall the KRK functionality (henceforth, GUC-KRK) allows the (monolithic) adversary to register and/or ask for the key of any corrupt player. We modify this to obtain the CP setup KRK such that the i^{th} adversary is allowed only to register and ask for the keys of the i^{th} player. One can verify that any protocol which is GUC secure in the original GUC-KRK hybrid world, is also GUC secure when using setup $[KRK]$.[12] Moreover all protocols of [10,14] using GUC-KRK are setup off-line. In [14] the GUC-KRK is used to register public keys which allow for non-interactive key agreement. Such public/secret key pairs can easily be constructed based on the Decisional Diffie-Hellman (DDH) assumption. Therefore we obtain the following corollary.

Corollary 1. *If the DDH assumption holds, there exist (efficient) setup $\bar{\mathcal{G}}$ and a programmable resource $\mathsf{M} = \{\mathsf{M}_x\}_{x\in\{0,1\}^*}$ such that for every well-formed aborting functionality $\mathcal{F}$, there exists a protocol π which CP-realizes $\widehat{\mathcal{F}}$ with GUC fallback in the $\{\bar{\mathcal{G}}, M_{\mathcal{F}}\}$-hybrid model.*

[12] The only difference is that when the (monolithic) adversary makes a registration or secret-key request for some party to $[KRK]$, it needs to append the ID of this party to the message. But this has no effect on the protocol.

6 Implications for Mechanism Design

In this section we translate our results into the language of game theory and interpret them in terms of reducing trust in mechanisms for games played in a computational setting.

Concurrent games and reducing trust in mechanisms. On the highest level we describe a setting where multiple (rational) players are involved in several concurrent games. As a guiding principle for this concurrent model of game-play we hold that *while actions are performed locally to a given game, the intentions and results should be interpreted globally.* This is done to reflect the real world consideration that "winning" in one game may be unimportant to a given player if a concurrent game is "lost" (or more generally lost by a fellow player).[13] Moreover strategies of concurrent games may depend on each other in a very real sense.[14]

A bit more formally, we generalizing the powerful mediated games model of [21]. Here game play involves message exchanges between the players and a trusted mediator i.e. the mechanism for that game. In line with our guiding principle we define actions as being local to a particular game. For example placing a bid in one auction is an independent event from say casting a vote in some concurrent election. However, strategies (i.e. the decision how much to bid and for whom to vote) as well as utility functions (i.e. say the cost of loosing the auction but winning the election) are defined over all games. This leads to a natural generalization of the equilibria notion(s) spanning concurrent games.

The goal of this section is to provide tools to simplify the task of mechanism design in a *concurrent computational world where trusted entities are a rare commodity.* To that end the tools we develop allow for the following:

- The mechanism may be described independently of the setting in which it will be used. In particular they are defined by a function mapping local actions to their local outputs.
- The mechanism may be designed and analyzed under the assumption it can be completely trusted.
- Yet the resulting (ideal) game can instead be played using a protocol and special network resource (in place of the mechanism) which can be implemented by computers such that:
  - The resource requires significantly less trust then the original mechanism. In particular it is only asked to enforce isolation. But it is not required to ensure that outcomes are computed correctly nor must it maintain any kind of privacy on behalf of the players. Instead these properties are obtained "for free".

[13] A telecom company that wins an auction for acquiring a patent for a new wireless technology may care less about this result if concurrently it lost the bid to use the wireless spectrum in which it was planning on implementing the technology.

[14] A company bidding concurrently for multiple add-spaces may be wiling to bid less in the first auction as it's bid in the second auction increases.

- Nevertheless, using the resource is equivalent to using the original mechanism in a very strong game-theoretic sense. In other words it does not matter to *this or any concurrent* game whether the ideal local mechanism is used as described by the mechanism designer or the protocol and resource are used. To justify this claim we show that (in particular) many types of interesting (computational) equilibria are maintained between the two settings. Moreover, finding an equivalent strategy can be done locally by each player with no interaction or external help.

Overview of results. In light of space constraints we refer the interested reader to the full version [4] for a formal treatise of these results. There we formalize our model of concurrent games and define a notion of isomorphic game sets. We note that the equivalence is both powerful and constructive. An isomorphism comes equipped with a pair of efficient and locally computable function for each player. These functions map from actions in Γ to actions in G (and vice versa) such that any payoff induced by the actions remains essentially unchanged in the new game. In particular we show that these mappings preserve computational Nash Equilibria [27] (cNE). In other words (the actions of) any cNE of Γ is mapped to (the actions of) a cNE in G inducing essentially the same payoffs. We go on to sketch how to extend this result to the computational versions of k-resilient equilibria [7], correlated equilibria [6], t-immune equilibria [2] and dominant strategies.

Next, we show a central theorem tying the notion of CP realization to concurrent games.

Adversary-oblivious functionalities. There is an important subtlety which must be addressed for such an application of CP-realization to go through. We view rational parties as taking the role of the adversaries (rather then the honest parties) in the CP setting. More technically to use CP-realization in a game theoretic setting we wish to preserve *adversarial* power, not just that of the honest players. In one direction (from the real to the ideal) we are given such power-preservation essentially for free by the CP-realization notion. However, in the other direction, in general, CP-realizing protocols and resources only guarantee preservation of the honest interfaces. This means that the strategy of an ideal rational player (making non-trivial use of their adversarial interface) may not have an equivalent strategy in the real world. For this reason we make the following definition restricting the type of resources about which we can use CP-realization to make game theoretic equivalence statements.

Definition 3 (informal). *An n-player CP functionality (with $2n$ interfaces) is called* adversary oblivious *if the only permitted input/output on any adversarial interface can be locally simulated given access only to the corresponding honest interface.*

Put simply an adversary oblivious functionality provides no additional (non-trivial) power to an adversary controlling both interfaces compared to one controlling only the honest player's interface.[15] Intuitively we require this restriction to guarantee that equilibria are preserved their stability when mapped to between game sets.

We can now (informally) state the central theorem tying CP-realization to mechanism design the proof of which can be found in the full version. In a nutshell it states that a resource which can be used to CP-realize another adversary oblivious resource can be used in it's place even in a concurrent game theoretic context.

Theorem 3 (Replacing Mechanisms). *Let $\mathcal{M}$ be an adversary oblivious CP-functionality, π be a protocol and $\mathcal{R}$ be a CP-functionality such that $\mathcal{M} \sqsubseteq^{CP} \mathcal{R}$. Let Γ be a game set using mechanism $\mathcal{M}$. Then there exists an isomorphic game set G using $\mathcal{R}$ in place of $\mathcal{M}$.*

Reducing trust. Finally, we apply the feasibility results for CP realization (Theorem 2) with GUC fallback to provide a concrete construction for reducing trust in concurrent used mechanisms. The GUC fallback satisfied by the construction in Theorem 2 is guaranteed regardless of the behavior of the resource it uses. In other words neither privacy of players actions, nor the correctness of the output computed by the mechanism rely on the honest behavior of the mechanism.

Since all players are rational our CP construction must tolerate full corruption. In this case the best we can hope to achieve is realization "with abort". In other words we can only apply our construction to mechanisms which permit players to abort.

Well motivated games. We model mechanisms as having a special input $\perp$ signaling the case when a player aborts the game or simply refuses to play. The construction of Theorem 2 permits simulating a mechanism which handles such actions by producing outcome $\perp$ for all players. However, we opt instead to place the reasonable assumption on the utility profile of the game ruling out such behavior as irrational. In particular we call a game set Γ *well motivated* if for all $i \in [n]$ the expected utility of any outcome obtained with an abort is less then the utility of all outcomes obtained without an abort.

Corollary 2 (Replacing Mechanisms). *Let $\bar{\mathcal{G}}$ be a functionality satisfying the conditions of Theorem 2 and let M be the corresponding programmable resource. Let $\mathcal{M}$ be an output synchronized mechanism*[16] *and c be polynomial in the security parameter. Then for any well motivated game set Γ using (amongst others) mechanism $\mathcal{M}$ there exists an isomorphic game set G using $\mathsf{M}_{\mathcal{M}}$ in place of $\mathcal{M}$ such that $\mathsf{M}_{\mathcal{M}}$ is not assumed to enforce privacy or compute output correctly.*

[15] Unlike an honest player, an adversary is not restricted to playing the honest (CP) protocol nor using the input (type) provided by the environment.

[16] i.e. an adversary oblivious CP-functionality with the synchronization wrapper applied to it.

Human strategies. We observe that the mappings between strategies arising from the construction in Theorem 2 actually simply compose the original strategy with a function (namely the protocol ITM) which translates actions "on-the-fly". In the context of humans playing games this has qualitative advantages over simply mapping from one strategy to the other. If actions are translated on the fly irrespective of the strategy used players need not even know their own strategy (let alone be aware of some efficient code implementing it) as long as they can always decide on a next action.

Acknowledgments. The authors thank Ran Canetti and Margarita Vald for their interesting discussions and helpful comments concerning the applications to mechanism design.

References

1. Abraham, I., Dolev, D., Gonen, R., Halpern, J.: Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In: 25th ACM PODC, pp. 53–62. ACM Press (2006)
2. Abraham, I., Dolev, D., Halpern, J.Y.: Lower Bounds on Implementing Robust and Resilient Mediators. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 302–319. Springer, Heidelberg (2008)
3. Alwen, J., Katz, J., Lindell, Y., Persiano, G., Shelat, A., Visconti, I.: Collusion-Free Multiparty Computation in the Mediated Model. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 524–540. Springer, Heidelberg (2009)
4. Alwen, J., Katz, J., Maurer, U., Zikas, V.: Collusion preserving computation. Cryptology ePrint Archive, Report 2011/443 (2011), `http://eprint.iacr.org/2011/433`
5. Alwen, J., Shelat, A., Visconti, I.: Collusion-Free Protocols in the Mediated Model. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 497–514. Springer, Heidelberg (2008)
6. Aumann, R.: Subjectivity and Correlation in Randomized Strategies. Journal of Math. Econ. 1, 67–96 (1974)
7. Aumann, R.J.: Acceptable points in general cooperative n-person games. In: Topics in Mathematical Economics and Game Theory Essays in Honor of Robert J Aumann, vol. 23, pp. 287–324 (1959)
8. Barany, I.: Fair distribution protocols, or how the players replace fortune. Mathematics of Operations Research 17, 327–340 (1992)
9. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE (2001), Full version at `http://eprint.iacr.org/2000/067/`
10. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally Composable Security with Global Setup. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 61–85. Springer, Heidelberg (2007)
11. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC, pp. 494–503 (2002)
12. Canetti, R., Vald, M.: Universally composable security with local adversaries. Cryptology ePrint Archive, Report 2012/117 (2012), `http://eprint.iacr.org/2012/117`

13. Crawford, V., Sobel, J.: Strategic information transmission. Econometrica 50, 1431–1451 (1982)
14. Dodis, Y., Katz, J., Smith, A., Walfish, S.: Composability and On-Line Deniability of Authentication. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 146–162. Springer, Heidelberg (2009)
15. Forges, F.: Universal mechanisms. Econometrica 58, 1342–1364 (1990)
16. Goldreich, O.: Foundations of Cryptography. Basic Applications, vol. 2. Cambridge University Press, Cambridge (2004)
17. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game, or a completeness theorem for protocols with honest majority. In: 19th ACM STOC, pp. 218–229. ACM Press (1987)
18. Izmalkov, S., Lepinski, M., Micali, S.: Rational Secure Computation and Ideal Mechanism Design. In: FOCS 2005: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, pp. 585–595. IEEE Computer Society, Washington, DC (2005)
19. Izmalkov, S., Lepinski, M., Micali, S.: Verifiably Secure Devices. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 273–301. Springer, Heidelberg (2008)
20. Izmalkov, S., Lepinski, M., Micali, S.: Perfect implementation. Games and Economic Behavior 71(1), 121–140 (2011), `http://hdl.handle.net/1721.1/50634`
21. Izmalkov, S., Micali, S., Lepinski, M.: Rational secure computation and ideal mechanism design. In: 46th FOCS, pp. 585–595. IEEE (2005), Full version available at `http://dspace.mit.edu/handle/1721.1/38208`
22. Lepinksi, M., Micali, S., Shelat, A.: Collusion-Free Protocols. In: STOC 2005: Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing, pp. 543–552. ACM, New York (2005)
23. Lepinski, M., Micali, S., Peikert, C., Shelat, A.: Completely fair SFE and coalition-safe cheap talk. In: 23rd ACM PODC, pp. 1–10. ACM Press (2004)
24. Lepinski, M., Micali, S., Shelat, A.: Collusion-free protocols. In: 37th ACM STOC, pp. 543–552. ACM Press (2005)
25. Lepinski, M., Micali, S., Shelat, A.: Fair-Zero Knowledge. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 245–263. Springer, Heidelberg (2005)
26. Maurer, U., Renner, R.: Abstract cryptography. In: Innovations in Computer Science. Tsinghua University Press (2011)
27. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory. Cambridge University Press, New York (2007)
28. Simmons, G.J.: The prisoners' problem and the subliminal channel. In: Crypto 1983, pp. 51–67. Plenum Press (1984)
29. Simmons, G.J.: Cryptanalysis and protocol failures. Communications of the ACM 37(11), 56–65 (1994)
30. Simmons, G.J.: The History of Subliminal Channels. In: Anderson, R. (ed.) IH 1996. LNCS, vol. 1174, pp. 237–256. Springer, Heidelberg (1996)