

Collusion Preserving Computation

by

Joël Alwen

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science
New York University
September 2011

Professor Yevgeniy Dodis

To my family and friends.

Acknowledgements

I would like to acknowledge the effort put into this thesis by myself. This work could not have been done without me. It was tough. But in the end I did it.

Good job me.

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 (Lepinski et al., STOC 2005) or in specific network topologies (Alwen et al., Crypto 2008).

This work provides a “clean-slate” definition of the stronger notion of collusion *preservation*. The goals in revisiting the definition are:

- To give a definition with respect to *arbitrary* communication resources (that includes 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 in the case when parties *can* communicate (a bounded amount of information) via other means. (This property is *not* implied by collusion-freeness.)
- To provide a definition supporting *composition*, so that 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 necessary properties of the underlying communication resource. Next we provide a general feasibility result

for collusion-preserving computation of arbitrary functionalities. We show that the resulting protocols enjoy an elegant (and surprisingly strong) fallback security even in the case when the underlying communication resource acts in a Byzantine manner. Finally, we investigate the implications of these results in the context of mechanism design.

Contents

- Dedication iii
- Acknowledgements iv
- Abstract v
- List of Figures ix
- Introduction 1

- 1 Collusion Preserving Computation 10**

 - 1.1 Collusion-Preserving Computation 10
 - 1.1.1 Preliminaries and Notation 10
 - 1.1.2 An Intuitive Description. 11
 - 1.1.3 Composition Theorem and Other Tools 17
 - 1.1.4 Relations to Existing Security Notions. 24
 - 1.2 Necessary Assumptions for CP 27
 - 1.2.1 Isolation 30
 - 1.2.2 Independent Randomness 32
 - 1.2.3 Programmability. 33

- 2 Fallback Security 35**

 - 2.1 GUC Fallback Security 35

2.2	A General Feasibility Result	39
2.2.1	Bootstrapping from GUC	43
2.2.2	Adding Fallback Security	43
2.2.3	A Concrete Instance	64
3	Implications for Mechanism Design	66
3.1	Viewing Protocols as Games	67
3.2	CP Realizing Mechanisms	68
3.3	Reducing Trust in Mechanisms	71
3.3.1	Equivalent Games	71
3.3.2	On Output Synchronization in a Stand-Alone Game	73
3.3.3	Implications of CP Compiler to Game Theory	76
3.4	Concurrent Games	80
A	Common Elements of (G)UC Models.	82
B	Relations to Abstract Cryptography.	86

List of Figures

1.1	UC corruptions compared to CP corruptions.(Setup functionalities are left implicit.)	12
1.2	\mathcal{R} -hybrid vs. \mathcal{F} -ideal CP executions where playerset $\mathcal{I} \subseteq [n]$ has been corrupted. (Setup functionalities are left implicit.)	14
1.3	The Insecure Channel functionality with split adversaries.	26
2.1	A detailed description of functionality $\mathcal{F}_{\text{inp}}(A, B)$	50
2.2	A detailed description of functionality $\mathcal{F}_{\text{comp}}(A, B)$	52
2.3	A detailed description of the mediator $M_{\mathcal{F}}$	54
2.4	A detailed description of protocol $\text{CP}_i(\pi)$	55

Introduction

Subliminal channels in protocols [39, 40, 41] allow parties to embed “disallowed” communication into the messages of the protocol itself, without being detected. (For example, a party might communicate a bit b by sending a valid protocol-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 parties to coordinate their actions (i.e., to collude) even if they may not have been aware of each other in advance. In other settings, parties may be disallowed or otherwise unable to communicate directly, and it would be undesirable if they could use the protocol itself to convey information.

These type of attacks are not exclusive to the realm of theory. Recently The Federal Communication Commission in the United States just auctioned off 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. A major concern for them was the aforementioned the problem of *collusion* between bidders. As documented in [15], 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.

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 [16, 20, 6, 33, 34, 35, 29, 1, 2, 27, 25]. Using protocols for secure multi-party computation (e.g., [22]) preserves 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 substantial effort toward constructing *collusion-free* protocols that do not allow any subliminal communication [33, 34, 35, 29, 27, 4, 3, 25]. Collusion-free protocols are impossible when parties are connected by pairwise communication channels, and so researchers have turned to other communication models. It is known that 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 [34, 29, 27, 25], or if parties are connected (via standard communication channels) to a semi-trusted entity in a “star network” topology [4, 3].

A New Definition: Collusion Preservation.

The works of Izmalkov et al. [34, 29, 27, 25] and Alwen et al. [4, 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*.¹ 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,

¹Collusion-free protocols ensure that if parties can communicate *nothing* without the protocol, then they can communicate nothing with the protocol. Collusion-preserving protocols provide a stronger guarantee: that *whatever* parties can communicate without the protocol, they can communicate no more with the protocol. See further discussion below.

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 definitional paradigm used by Alwen et al. [4, 3], and review their model here. 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 security are then defined, depending on whether or not the mediator is honest:

CONDITIONAL COLLUSION FREENESS: When the mediator is honest, collusion freeness is required. This is formalized by considering an ideal world where there are similarly adversaries who cannot communicate directly, but can only send inputs to (and receive outputs from) an ideal functionality. A protocol is collusion-free if for any PPT real-world adversaries interacting with the (honest) mediator, there are PPT ideal-world adversaries such that the real and ideal worlds are indistinguishable. Indistinguishability here is formulated in a “stand-alone” manner [21].

FALLBACK SECURITY: When the mediator is dishonest, we clearly cannot hope for collusion freeness any more. Nevertheless a strong and meaningful notion of security can be achieved. Namely, adversaries are now allowed to communicate arbitrarily (in both the real and ideal worlds), and the protocol is required to satisfy the standard notion of stand-alone security [21].

We strengthen and extend the definition of collusion freeness in several ways. First, rather than considering a specific “star network” topology, or the specific physical assumptions of [34, 29, 27, 25], 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 the inherent advantages of a more general definition, formulating a generic definition allows us to characterize the minimal properties that resources need in order to achieve collusion-preserving computation.

An additional difference is that we formulate our definitions in a universally composable (UC) [8] fashion, where there is an environment controlling the entire execution. (Actually, we use the *generalized* UC framework [10] as our starting point.) 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 obtain composition “for free” due to the power of the UC framework. In this way we improve upon the results of [4, 3], which do not claim nor realize any form of composition, as well as the results of [34, 29, 27, 25] which obtain only a limited sort of composition; see below.

Collusion preservation is stronger than collusion freeness. We give a simple counter-example showing that collusion preservation is stronger than collusion freeness. Consider a protocol π that is collusion-free in the communication model of [4, 3] where, in short, there is a central mediator to whom all parties are connected (and no other channels are available). 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) .
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!

One can interpret this counter-example 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

²In 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.

implication is the same: a collusion-free protocol may potentially allow additional communication once parties have the ability to communicate at all.

Protocol composition. A critical feature of the protocols of Izmalkov et al. [34, 29, 27, 25] is that they are only collusion-free when *at least one party running the protocol is honest*. The underlying reason is that in their communication model parties have *the ability* to communicate arbitrary information; the guarantee provided by their protocols is that any such communication will be detected. Collusion freeness thus requires an honest party to perform the detection.

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 P_1, P_2, 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 . (Izmalkov et al. implicitly avoid this issue by having all parties take part in any sub-protocols that are run. While this solves the issue, it is not an efficient approach.)

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.

Implications for computational game theory. The CP framework is defined in terms of computationally bounded parties. As such the implications for game theory concern games with similar limitations as in the field of algorithmic game theory [38]. We consider these (extensive form) computational games to be equivalent in some sense if they contain the same set of computational Nash Equilibria (cNE) which also implies that stronger kinds equilibria are also preserved.

Translating the new security notion and our general feasibility result to the setting of computational mediated games we show how to replace a mechanism with a protocol running over a “less trusted” mechanism such that the resulting game is equivalent to the original. By less trusted we mean that the mechanism need no longer be trusted to enforce privacy, nor to compute the game round functions correctly. (However it is still trusted to enforce both fairness and the isolation of players.)

In comparison to results of [26, 28, 25] 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 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 [26], they do not provide the full robustness to aborts of [28, 25]. 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 [32], better than [26] (where even executions of a protocol produce randomness pollution) but weaker than [28, 25] which do not produce any randomness pollution at all.

Relations to Abstract Cryptography. In concurrent work Maurer and Renner introduced the Abstract Cryptography (AC) framework [36]. In the final section we describe in some detail how the CP framework can be viewed as a concrete instance of the AC framework. Besides elucidating parallels between these two frameworks, to the best of our knowledge the following discussion represents the first concrete instantiation of AC.

Outline of this thesis. In Chapter 1 we define a Universally Composable version of Collusion Freeness, called Collusion Preservation (CP). In Section 1.1 we prove its composability and show how it relates to existing universally composable security notions. In Section 1.2 we characterize three minimal assumptions any resource must have if it is to be used for realizing some important classes of functionalities; our characterization rules out the use of standard communications models to implement almost any interesting functionality. Next, in Chapter 2 we demonstrate a general feasibility result. First, in Section 2.1 we define an additional security property called GUC Fallback. Then in Section 2.2 we present a protocol compiler and matching resource with which a large class of functionalities can be CP implemented while simultaneously enjoying GUC Fallback security. In Chapter 3

we investigate the implications of these results to the area of Mechanism Design. Somewhat more precisely: we show how to use the protocol compiler to significantly reduce the trust in a mechanism for games in a computationally bounded setting. Finally in Appendix A we briefly review common elements of universally composable frameworks of which we make use in defining the CP framework and in Appendix B we show how the CP framework can be defined in the terms of the Abstract Cryptography of [36].

Chapter 1

Collusion Preserving Computation

We begin by introducing the CP framework, and investigate some of its properties.

1.1 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 [11] with the model of split simulators along the lines of [3].

1.1.1 Preliminaries and Notation

We denote by $[n]$ the set $\{1, \dots, n\}$ (by convention $[0] = \emptyset$) and for a set $\mathcal{I} \subseteq [n]$ we write $\bar{\mathcal{I}}$ to denote the set $[n] \setminus \mathcal{I}$. Similarly, for element $i \in [n]$ we write \bar{i} to

denote the set $[n] \setminus \{i\}$. Using this notation we denote by $\mathbf{A}_{\mathcal{I}}$ a set of ITMs $\{\mathbf{A}_i\}_{i \in \mathcal{I}}$. For input tuple $x_{\mathcal{I}} = \{x_i\}_{i \in \mathcal{I}}$ we write $\mathbf{A}_{\mathcal{I}}(x_{\mathcal{I}})$ to denote that for all $i \in \mathcal{I}$ the ITM \mathbf{A}_i is run with input x_i (and a fresh uniform independent random tape).

Additionally we will make heavy use of elements from the GUC framework such GUC protocols and GUC setup functionalities. For reasons of space we assume passing familiarity with the model used their in. For a more detailed description of the main features of GUC used to describe our framework we refer to Appendix A.

1.1.2 An Intuitive Description.

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 $\mathbf{A}_{[n]} = \{\mathbf{A}_i : i \in [n]\}$, where \mathbf{A}_i correspond to the adversary associated with the player i (and can corrupt at most this party). Moreover, we ask that for each $\mathbf{A}_i \in \mathbf{A}_{[n]}$ 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 $\mathbf{A}_{[n]}$ and $\mathbf{B}_{[n]}$ as long as $\mathbf{A}_i = \mathbf{B}_i$.

Modeling Split Adversaries. To incorporate the notion of a split adversary in the GUC model, we make the following modifications to the model of execution: Let $\mathcal{P} = [n]$ be the player set; instead of a single adversary, we introduce to the model n independent adversaries $\mathbf{A}_1, \dots, \mathbf{A}_n$, such that for each $P_i \in \mathcal{P}$, \mathbf{A}_i might

corrupt at most party p_i .¹ For this purpose, each

adversary is associated with a unique adversary-ID, which includes the party-ID of the corresponding party. Each of the adversaries A_i has a dedicated interface to communicate with Z .² Note that, unlike the standard (G)UC models, the adversaries do not serve as an underlying insecure network, as they do not share communication tapes

with the honest parties or with each other. In order to make statements about computation of non-trivially computable, i.e., non-locally computable, functionalities one needs to consider hybrid worlds, where the hybrid serves as the communication resource for the parties and/or as the “co-ordination” mechanism for the adversaries. In the following we give a generic specification of the mode of operation of functionalities in our split adversaries setting and sketch the model of execution of protocol in the corresponding hybrid-model. The difference between the multi-adversary hybrid world execution and the standard (G)UC execution is graphically represented in Figure 1.1.

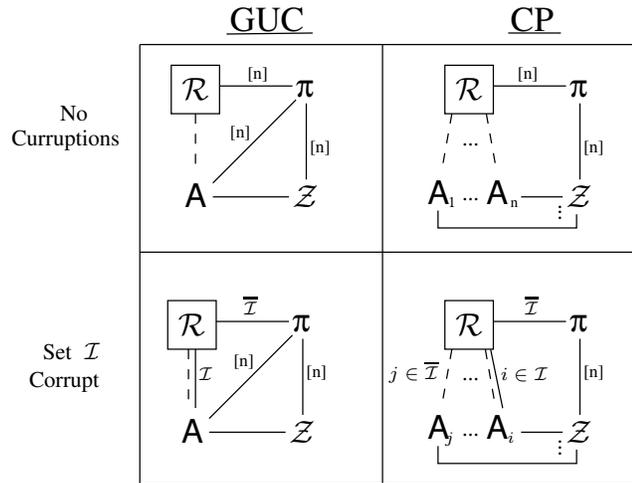


Figure 1.1: UC corruptions compared to CP corruptions.(Setup functionalities are left implicit.)

¹The corruption mechanism is similar to the (G)UC setting, i.e., the environment Z requests A_i corrupt P_i .

²Technically, as in the UC setting, each such interface corresponds to A_i and Z sharing a communication tape.

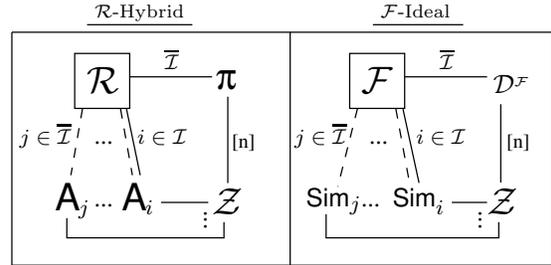
Resources, Shared Functionalities, and Exclusive Protocols. The main *difference* between a CP functionality \mathcal{F} and GUC one is that besides the n interfaces to the (honest) parties it also has interfaces to *each* of the n adversaries A_1, \dots, A_n . In other words rather than n interfaces a CP functionality has $2n$ interfaces.

Moreover, similar to the GUC framework (but in contrast to plain UC) we distinguish between two types of functionalities: *resources* which we denote with capital calligraphic font as in “ \mathcal{R} ” and *shared functionalities* which we denote with an additional over-line as in “ $\bar{\mathcal{G}}$ ”. Formally a resource \mathcal{R} maintains state only with respect to a single instance of a protocol, while a *shared* functionality $\bar{\mathcal{G}}$ can maintain state across protocol instances.³ For example concurrent executions can maintain shared state via say a global CRS (like the ACRS of the GUC framework) or via a global PKI (such as the KRK setup for GUC) as long as these are modeled as shared functionalities. However, although concurrent instances of a protocol π may use the same resource \mathcal{R} , the behavior of \mathcal{R} in one execution of π must be independent of all other executions of π (and more generally of all other concurrent protocols instantiated by the environment). For clarity, in the remainder of this work we will usually refer to shared functionalities simply as *setup* and protocols which only share state across executions through some setup $\bar{\mathcal{G}}$ as *$\bar{\mathcal{G}}$ -subroutine respecting*. As observed already in previous works, e.g., see [11], in practice most protocols are in fact subroutine respecting with respect to the shared functionality which they use as setup.⁴

³Technically this is modeled by restricting an instance of a resource to only accept inputs from ITI with a fixed session ID value while a shared functionality is can accept inputs from an ITI with *any* session ID.

⁴This notion is completely analogous to that of “subroutine respecting” as defined for the (G)UC frameworks.

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. In particular when the environment \mathcal{Z} requests corruption of a player P_i the adversary A_i is given control of P_i 's interface to \mathcal{R} (c.f. Figure 1.2). On a technical level the execution of an \mathcal{R} -hybrid protocol is almost identical to an execution in the GUC framework.



In particular, environments can instead invoke an arbitrary number of ITI of any kind; even those running other protocols and sharing state via some setup. We denote the out-

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

Figure 1.2: \mathcal{R} -hybrid vs. \mathcal{F} -ideal CP executions where playerset $\mathcal{I} \subseteq [n]$ has been corrupted. (Setup functionalities are left implicit.)

On 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 the CP case parties can communicate with at most a *single* instance of \mathcal{R} . This is in contrast to all other UC like models where say an OT-hybrid world is understood to mean that parties can make as many calls as they wish to the OT functionality instantiating a new copy for each new OT transaction they wish to perform.

At first glance this may seem like a rather minor modeling issue since anyway environments are not always aware of the presence of ideal functionalities (otherwise distinguishing hybrid world would be trivial). However we argue that, in contrast to a setting with a monolithic adversary where no such restriction is made, for a composable notion with split adversaries fixing the number of instances of functionalities (i.e. 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 in the mediators used in mechanism design by providing protocols which can be used to replace the interaction with the mediator. However 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.

From a cryptographic point of view, suppose we prove that some protocol π “realizes” a functionality \mathcal{F} *without* fixing the number of instances of \mathcal{F} simulators can interact with. Intuitively, this would mean π allows as much collusion as can be obtained by an *unlimited* number of calls to \mathcal{F} . If \mathcal{F} is a one-bit bi-directional channel for example the meaning of the statement changes completely if simulators use only a single instance of \mathcal{F} to simulate versus when they can use unlimited calls to \mathcal{F} .

We contrast this with the case of a monolithic adversary where no such issue arises (and indeed, the definition of (G)UC seem to allow for such simulations). The underlying reason is that even in the ideal world there is only a single simulator. Thus any correlation obtained by the simulators in the previous example for say player i and player j from the first execution of π is now irrelevant as the same correlation can be simulated trivially internally by the monolithic simulator

controlling both parties. Thus no intuition is lost by not fixing how many calls to the ideal functionality are made for simulation purposes.

Definition 1.1.1 (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 $\text{Sim} = \text{Sim}_{[n]}$ mapping ITMs to ITMs such that for every set of adversaries $\mathbf{A} = \mathbf{A}_{[n]}$, and every PPT environment \mathcal{Z} the following holds:*

$$\text{CP-EXEC}_{\pi, \mathbf{A}, \mathcal{Z}}^{\bar{\mathcal{G}}, \mathcal{R}} \approx \text{CP-EXEC}_{\phi, \text{Sim}(\mathbf{A}), \mathcal{Z}}^{\bar{\mathcal{G}}, \mathcal{F}}.$$

Realization, Reductions and the “ \sqsubseteq ” Notation. As in the (G)UC frameworks, we distinguish between the more general notion of “emulation” and the special case of “realization”. For an (implicit) functionality \mathcal{F} we denote by $\mathcal{D}_i^{\mathcal{F}}$ the i^{th} dummy \mathcal{F} -hybrid protocol which simply acts as a transparent conduit between the i^{th} honest and adversarial interfaces of \mathcal{F} and \mathcal{Z} . In particular $\mathcal{D}_i^{\mathcal{F}}$ forwards all messages it receives from \mathcal{Z} to the functionality \mathcal{F} (where the choice of adversarial or honest interface is specified by \mathcal{Z}) and vice-versa. If for functionality \mathcal{F} , an \mathcal{R} -hybrid protocol π CP-emulates $\mathcal{D}^{\mathcal{F}}$ then we say that π realizes \mathcal{F} (in the \mathcal{R} -hybrid world). In symbols we denote this by $\mathcal{F} \sqsubseteq_{\pi}^{\text{CP}} \mathcal{R}$, which can intuitively be read as “ \mathcal{F} CP-reduces to \mathcal{R} via protocol π ”.⁵ 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.

By convention for any functionality \mathcal{R} we consider a pair of protocols equivalent

⁵Alternatively it can also be read as “Protocol π CP-realizes \mathcal{F} in the \mathcal{R} hybrid model.”

$\pi = \phi$ in the \mathcal{R} -hybrid world if for all functionalities \mathcal{F} we have:

$$\mathcal{F} \sqsubseteq_{\pi}^{CP} \mathcal{R} \iff \mathcal{F} \sqsubseteq_{\phi}^{CP} \mathcal{R}$$

In particular let $\mathcal{D}^{\mathcal{R}}$ be the dummy protocol for \mathcal{R} . Then by convention, for any protocol π we have $\pi^{\mathcal{D}^{\mathcal{R}}} = \pi$ in the \mathcal{R} -hybrid world where $\pi^{\mathcal{D}^{\mathcal{R}}}$ is the protocol behaving as π but with calls to \mathcal{R} being forwarded through $\mathcal{D}^{\mathcal{R}}$ as if they came from \mathcal{Z} .

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 $\mathcal{D}^{\mathcal{F}}$. For example we might write $\text{CP-EXEC}_{\mathcal{A}_{[n]}, \mathcal{Z}}^{\mathcal{F}}$ when the honest players are running $\mathcal{D}_{[n]}^{\mathcal{F}}$. For clarity, we include a graphical representation of the multi-adversary hybrid-world/ideal-world definitions in Figure 1.2.

1.1.3 Composition Theorem and Other Tools

We formalize a strong (universally) composable property of CP security and, along the way, provide a useful tool for proving CP security of protocols.

1.1.3.1 Simplified CP

Following the approach of [11] we make two simplifications to the definition of CP security obtaining what we call *Simplified CP (SCP)*.⁶ SCP has two desirable properties:

1. It significantly easier to prove a protocol SCP secure then CP secure.

⁶This can be thought of as being analogous to Externalized-UC combined with the Dummy Lemma.

2. Yet SCP security is equivalent to CP security under certain reasonable conditions.

Combining these two property also results in a simpler proof of the main UC-type composition theorem then if we try and prove it directly for CP security.

To simplify the task of proving protocols SCP secure we restrict the class of environments being considered (much like "Externalized UC" of [11]). Let $\bar{\mathcal{G}}$ be a setup, \mathcal{R} be a resource and π be an $(\bar{\mathcal{G}}, \mathcal{R})$ -exclusive protocol.⁷ Then for SCP security we quantify only over *restricted environments* which are PPT environments which do not invoke any other ITMs besides a single instance of protocol π as well as one instance of the dummy protocol $\mathcal{D}^{\bar{\mathcal{G}}}$. The second modification we make to CP is that we quantify only over dummy adversaries. That is for all $i \in [n]$ we consider only the adversary A_i which acts as a conduit between \mathcal{Z} and the resource \mathcal{R} .

Proving SCP security is significantly easier than proving CP security, because only a single kind of simulator need be considered (namely for the dummy adversaries). Moreover when verifying the correctness of these simulators it is easier (tractable) to reason about the entire view of an environment if it's view consists of only a single execution of the protocol and the interaction with the shared functionality.

Definition 1.1.2 (Simplified 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 protocols and ϕ be a $\{\bar{\mathcal{G}}, \mathcal{F}\}$ -exclusive protocols (both n -party protocols). Let $\mathbf{B} = \mathbf{B}_{[n]}^{\pi}$ be the set of n dummy adversaries interacting with π . Then we say that π Simplified CP (SCP) emulates ϕ in the $\{\bar{\mathcal{G}}, \mathcal{R}\}$ -hybrid world, if there exists a collection of efficiently*

⁷Note that this implies that π is also $\bar{\mathcal{G}}$ -subroutine respecting protocol.

computable transformations $\text{Sim} = \text{Sim}_{[n]}$ mapping ITMs to ITMs such that for every restricted environment \mathcal{Z} it holds that:

$$\text{CP-EXEC}_{\pi, \mathcal{B}, \mathcal{Z}}^{\bar{\mathcal{G}}, \mathcal{R}} \approx \text{CP-EXEC}_{\phi, \text{Sim}(\mathcal{B}), \mathcal{Z}}^{\bar{\mathcal{G}}, \mathcal{F}}$$

Equivalence Theorem. We first show that although SCP security may at first glance seem weaker than CP security, in fact they are equivalent. As in the GUC framework, results relating SCP to CP are conditioned on protocols being subroutine respecting. Intuitively this is because only if a protocol is $\bar{\mathcal{G}}$ -subroutine respecting can it be guaranteed that giving the environment access to $\bar{\mathcal{G}}$ is sufficient to faithfully internally emulate multiple concurrent executions of the protocol.

Theorem 1.1.3 (Equivalence). *Let $\bar{\mathcal{G}}$ be a setup, \mathcal{R} and \mathcal{F} be resources, π be a $(\bar{\mathcal{G}}, \mathcal{R})$ -exclusive protocol and ϕ be a $(\bar{\mathcal{G}}, \mathcal{F})$ -exclusive protocol. Then π CP emulates ϕ if and only if π SCP emulates ϕ .*

Proof. As the set of environments and adversaries for SCP are a subset of those for CP so the forwards direction (“ \implies ”) is trivially true.

The backwards direction (“ \impliedby ”) requires more work. It combines in a straight forward manner the ideas of the proof of the so called “dummy lemma” of [9] with those from the proof of Theorem 2.1 [11]⁸. Indeed the original proofs translate almost one-to-one and all that is needed is to verify that splitting the adversaries does not cause a break down in the logical arguments.

For completeness we sketch the entire proof highlighting the changes required to accommodate split adversaries. We first argue that quantifying over fewer environments does not change the quality of the security notion. Then we argue

⁸Theorem 2.1 in [11] states that GUC and EUC are equivalent.

that considering only dummy adversaries also has no effect on the quality of the security notion which concludes the proof.

We show that for any \mathcal{Z} in the CP framework there exists a restricted environment $\tilde{\mathcal{Z}}$ with at most polynomial loss in advantage at distinguishing executions of ϕ from π . This breaks down into two arguments:

1. Instantiating polynomially many arbitrary concurrent protocols communicating with $\bar{\mathcal{G}}$ does not help \mathcal{Z} as $\tilde{\mathcal{Z}}$ has a direct link to $\bar{\mathcal{G}}$ (via the dummy protocol for $\bar{\mathcal{G}}$) and so can perfectly emulate all concurrent protocols internally.
2. Interacting with polynomially many instances of π does not help \mathcal{Z} significantly via a standard hybrid argument. The core of the hybrid argument is that there must be an l^{th} concurrent execution of π that helps \mathcal{Z} distinguish significantly and so $\tilde{\mathcal{Z}}$ can simulate the previous $l - 1$ execution and use the l^{th} as the real one. Therefore there is at most a polynomial loss in the advantage of $\tilde{\mathcal{Z}}$ when compared to the advantage of \mathcal{Z} .

Roughly speaking, we have shown that a protocol looks the same to all efficient environments only if it looks the same to all restricted environments. It remains to show that for any distinguishing restricted environment interacting with *arbitrary* adversaries there exists a distinguishing restricted environment interacting only with dummy adversaries. The proof remains essentially unchanged from that of the dummy lemma in [9] so we only briefly mention how to extend it to the SCP setting. In particular it relies on the idea that the simulator for dummy adversaries can be used as “on-line translators” turning views of ϕ into views of π .

The original lemma (Claim 10 in [9]) concerns a single dummy adversary \mathbf{B}^π attacking π (potentially controlling multiple players $\mathcal{I} \in [n]$) rather than a set of

individual dummies each controlling only a single player). By assumption there exists a simulator $\tilde{\text{Sim}}(\mathbf{B}^\pi)$ attacking execution(s) of ϕ that is able to create an indistinguishable view of execution(s) of π for all players it controls which it feeds to the environment (just as \mathbf{B}^π would when attack π). The proof uses this fact to construct, for any adversary \mathbf{A} attacking π , a simulator $\text{Sim}(\mathbf{A})$ attacking ϕ with similar capabilities as $\tilde{\text{Sim}}(\mathbf{B}^\pi)$. More precisely, for an arbitrary adversary \mathbf{A} the proof describes a simulator $\text{Sim}(\mathbf{A})$ which internally emulates \mathbf{A} and let's it interact with an internal emulation of $\tilde{\text{Sim}}$ which is used to translate the view of ϕ into a matching view of π for \mathbf{A} to attack. To see that $\text{Sim}(\mathbf{A})$ is a good simulator it is observed that an environment \mathcal{Z} which can distinguish between $\text{EXEC}_{\pi_{\mathcal{I}}, \mathbf{A}_{\mathcal{I}}, \mathcal{Z}}^{\tilde{\mathcal{G}}}$ and $\text{EXEC}_{\phi_{\mathcal{I}}, \tilde{\text{Sim}}_{\mathcal{I}}(\mathbf{A}), \mathcal{Z}}^{\tilde{\mathcal{G}}}$ can be used to break the correctness of $\tilde{\text{Sim}}(\mathbf{B}^\pi)$. All that is needed is for an environment \mathcal{Z}' to run \mathcal{Z} and \mathbf{A} internally and let them interact directly with $\tilde{\text{Sim}}(\mathbf{B}^\pi)$ or \mathbf{B}^π .

The same argument carries through to our setting. In the SCP framework, given a set of n simulators which work for any subset of dummy adversaries, the same construction of simulators for sets of arbitrarily corrupt players will work. Let $\mathbf{A}_{[n]}$ be a set of arbitrary adversaries attacking π . The goal is to describe an (efficient) transformation Sim_i mapping \mathbf{A}_i to a corresponding simulator $\text{Sim}_i(\mathbf{A}_i)$ for all $i \in [n]$. By assumption, for dummy adversaries $\mathbf{B}_{[n]}^\pi$ interacting with protocol π SCP security already provides for a set of transformations $\tilde{\text{Sim}}_{[n]}$ such that the simulators $\tilde{\text{Sim}}_{[n]}(\mathbf{B}_{[n]}^\pi)$ interact with an execution of ϕ and can simulate all dummy adversaries' views of π faithfully to any (restricted) environment. For each $i \in [n]$ we construct $\text{Sim}_i(\mathbf{B}_i^\pi)$ from $\tilde{\text{Sim}}_i(\mathbf{B}_i^\pi)$ just as in the original proof.

It remains to show that these are good constructions. Formally we must show that no environment \mathcal{Z} has significantly different output for $\text{CP-EXEC}_{\pi, \mathbf{A}_{[n]}, \mathcal{Z}}^{\tilde{\mathcal{G}}, \mathcal{R}}$

and for $\text{CP-EXEC}_{\phi, \text{Sim}_{[n]}(\mathbf{B}_{[n]}^\pi), \mathcal{Z}}^{\bar{\mathcal{G}}, \mathcal{F}}$. Suppose for sake of contradiction that \mathcal{Z} can distinguish between these two executions. In this case we can build restricted environment \mathcal{Z}' which breaks the SCP security of π by absorbing $\mathbf{A}_{[n]}$ and \mathcal{Z} just as in the original proof. Each \mathbf{A}_i interacts directly with the i^{th} adversary (be it \mathbf{B}_i^π or $\tilde{\text{Sim}}_i(\mathbf{B}_i^\pi)$). Then by outputting the same bit as \mathcal{Z} environment \mathcal{Z}' has an identical advantage at breaking the SCP security of π as \mathcal{Z} has at breaking the simulation of $\text{Sim}_{[n]}(\mathbf{B}_{[n]}^\pi)$. The fact that there are n different absorbed \mathbf{A}_i each interacting with a single external adversary (rather than one \mathbf{A} interacting with all corrupt players) has no effect on the strategy of \mathcal{Z}' as it can directly communicate with all external adversaries and so does not rely on their implicit coordination via a monolithic modeling. \square

1.1.3.2 A 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 capture this in the following central theorem.

Theorem 1.1.4 (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. If π SCP-emulates ϕ and ρ uses ϕ as a subroutine then $\rho^{\pi/\phi}$ CP-emulates ρ in $\{\bar{\mathcal{G}}, \mathcal{R}\}$ -hybrid world.*

Proof. The ideas behind the proof of GUC composition carry over directly to the setting with split simulators but for completeness we give a full proof. For further details we refer to the proof of Theorem 2.1 in [11] which can be applied almost directly to this setting.

Let $\mathbf{B} := \mathbf{B}_{[n]}$ be the set of dummy adversaries for the $\{\bar{\mathcal{G}}, \mathcal{R}\}$ -hybrid world. Following the same logic as in the proof of Theorem 1.1.3 it suffices to prove that for any efficient environment \mathcal{Z} :

$$\text{CP-EXEC}_{\rho^{\pi/\phi}, \mathbf{B}, \mathcal{Z}}^{\bar{\mathcal{G}}, \mathcal{R}} \approx \text{CP-EXEC}_{\rho, \mathbf{A}, \mathcal{Z}}^{\bar{\mathcal{G}}, \mathcal{R}}. \quad (1.1)$$

Since π and ϕ are subroutine respecting it follows by Theorem 1.1.3 that π *CP*-emulates ϕ . Thus there exist simulators $\text{Sim} = \text{Sim}_{[n]}(\mathbf{B})$ such that

$$\text{CP-EXEC}_{\pi, \mathbf{B}, \mathcal{Z}_\pi}^{\bar{\mathcal{G}}, \mathcal{R}} \approx \text{CP-EXEC}_{\phi, \text{Sim}, \mathcal{Z}_\pi}^{\bar{\mathcal{G}}, \mathcal{R}} \quad (1.2)$$

for any environment \mathcal{Z}_π . We use Sim to construct \mathbf{A} satisfying Equation 1.1. Adversary \mathbf{A}_i internally runs $\text{Sim}_i(\mathbf{B}_i)$ forwarding messages from \mathcal{Z} intended for instances of π_i to $\text{Sim}_i(\mathbf{B}_i)$ instead. Moreover, any messages from $\text{Sim}_i(\mathbf{B}_i)$ destined for either \mathcal{Z} or ϕ_i are forwarded faithfully.

We note that the transformation from \mathbf{B}_i to \mathbf{A}_i is efficiently computable and moreover it depends only on the code of \mathbf{B}_i . Thus it remains to show that Equation 1.1 is satisfied. We do this by constructing an environment \mathcal{Z}_π for which Equation 1.2 holds if and only if Equation 1.1 holds. The fact that π SCP-emulates ϕ then concludes the proof.

Intuitively \mathcal{Z}_π absorbs \mathcal{Z} , ρ and all of \mathbf{A} except $\text{Sim}_i(\mathbf{B}_i)$ internally. However for all messages \mathbf{A}_i would forward to its internal copy of $\text{Sim}_i(\mathbf{B}_i)$ are instead routed by \mathcal{Z}_π to the i^{th} (external) adversary. This will either be \mathbf{B}_i or $\text{Sim}_i(\mathbf{B}_i)$. All responses are similarly forwarded back to \mathbf{A}_i . Finally when \mathcal{Z} produces output \mathcal{Z}_π outputs the same value and terminates.

Observe that $\text{CP-EXEC}_{\pi, \mathbf{B}, \mathcal{Z}_\pi}^{\bar{\mathcal{G}}, \mathcal{R}} = \text{CP-EXEC}_{\rho^{\pi/\phi}, \mathbf{B}, \mathcal{Z}}^{\bar{\mathcal{G}}, \mathcal{R}}$. Indeed the only dif-

ference in the view of \mathcal{Z} for both executions is where any given message is sampled (but not from which distribution). Similarly $\text{CP-EXEC}_{\phi, \text{Sim}, \mathcal{Z}_\pi}^{\bar{\mathcal{G}}, \mathcal{R}} = \text{CP-EXEC}_{\rho, \mathcal{A}, \mathcal{Z}}^{\bar{\mathcal{G}}, \mathcal{R}}$. Thus Equation 1.1 holds for \mathcal{Z} if and only if Equation 1.2 hold for \mathcal{Z}_π . \square

1.1.4 Relations to Existing Security Notions.

We prove a pair of lemmas relating CP results to matching GUC results. In Lemma 1.1.5 we show to translate a statement about CP realization into a related statement about GUC realization. Then in Lemma 1.1.6 we show how to convert statements in the opposite direction. Together these results capture the intuitive claim that the CP model is at least as expressive as the GUC (and so also UC) models.

1.1.4.1 CP Implies GUC.

We first formalize the intuitive claim that CP security is at least as strong as GUC security with the lemma which states that CP realization essentially implies the GUC realization. We describe a natural mapping of CP functionalities to analogous GUC functionalities $\mathcal{F} \mapsto [\mathcal{F}]$. 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 $[\mathcal{F}]$.

Recall that the only difference between (possibly shared) GUC and CP functionalities is that the former has a single adversarial interface while the latter has n such interfaces. The mapping $[\cdot]$ is obtained as follows. For a CP functionality \mathcal{F} , let $[\mathcal{F}]$ denote the GUC functionality which behaves as \mathcal{F} except that the adversaries interface has the following modification:

- Whenever \mathcal{F} would output a message \mathbf{Msg} to the i^{th} adversarial interface, $[\mathcal{F}]$ instead outputs (i, \mathbf{Msg}) on its (single) adversarial interface.
- Whenever $[\mathcal{F}]$ receives a message of the form (i, \mathbf{Msg}) on its adversarial interface, it simulates the behavior of \mathcal{F} upon receiving message \mathbf{Msg} on the i^{th} adversarial interface.

Lemma 1.1.5. *Let $\bar{\mathcal{G}}$ be a setup and \mathcal{F} and \mathcal{R} be functionalities all in the CP model. If a protocol π CP-realizes \mathcal{F} in the $\{\bar{\mathcal{G}}, \mathcal{R}\}$ -hybrid model, then π GUC-securely realizes $[\mathcal{F}]$ in the $\{[\bar{\mathcal{G}}], [\mathcal{R}]\}$ -hybrid model. In symbols:*

$$\{\bar{\mathcal{G}}, \mathcal{F}\} \sqsubseteq_{\pi}^{CP} \{\bar{\mathcal{G}}, \mathcal{R}\} \implies \{[\bar{\mathcal{G}}], [\mathcal{F}]\} \sqsubseteq_{\pi}^{GUC} \{[\bar{\mathcal{G}}], [\mathcal{R}]\}$$

1.1.4.2 Translating GUC Statements to CP Statements

As a primary building block for feasibility results we will use GUC protocols. To show how to translate GUC results into analogous statements in the CP framework we first need to formalize the CP analogue of the insecure channels functionality.

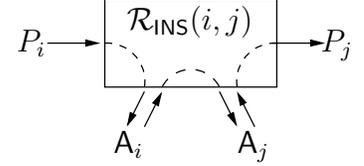
The \mathcal{R}_{ins} Functionality. Intuitively, the main differences between the two models are that in the CP model:

1. In the plain CP model is not equipped with insecure channels (as is the case for GUC).
2. CP adversaries are split and so cannot, a priori, coordinate their attacks arbitrarily.
3. The GUC model explicitly assumes authenticated communication.

To address the first difference we *explicitly* equip the CP world with insecure channels. When modeled appropriately, a nice side effect is that the CP adversaries

can also use the insecure channels to coordinate their attacks arbitrarily taking care of the second difference.

Formally we model the insecure channel resource (depicted in Figure 1.3), denoted by $\mathcal{R}_{\text{ins}}(i, j)$, as being parameterized by integers $i, j \in [n]$ which denote the indices of two parties called the *sender* P_i and the *receiver* P_j . Upon receiving a message m from party:



- P_i : Resource $\mathcal{R}_{\text{ins}}(i, j)$ forwards m to adversary A_i .
- A_i : Resource $\mathcal{R}_{\text{ins}}(i, j)$ forwards m to adversary A_j .
- A_j : Resource $\mathcal{R}_{\text{ins}}(i, j)$ forwards m to receiver P_j .

Figure 1.3: The Insecure Channel functionality with split adversaries.

We denote the complete network of insecure channels by

$$\mathcal{R}_{\text{ins}} = \{\mathcal{R}_{\text{ins}}(i, j) : (i, j) \in [n]^2, i \neq j\}.$$

Resource \mathcal{R}_{ins} is used just as the implicit insecure channels are used in (G)UC.

Lemma 1.1.6. *Let $\bar{\mathcal{G}}$ be a (CP) setup such that there exists some $[\bar{\mathcal{G}}]$ -subroutine respecting protocol that GUC realizes authenticated channels (over insecure channels).⁹ Further let \mathcal{F} and \mathcal{R} be CP functionalities.*

Then if a protocol π GUC realizes $[\mathcal{F}]$ in the $[\bar{\mathcal{G}}]$ -hybrid world, then it also CP realizes $\{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}, \mathcal{F}\}$ in the $\{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}\}$ -hybrid model with shared functionality $\bar{\mathcal{G}}$. In symbols:

$$\{[\bar{\mathcal{G}}], [\mathcal{F}]\} \sqsubseteq_{\pi}^{\text{GUC}} [\bar{\mathcal{G}}] \implies \{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}, \mathcal{F}\} \sqsubseteq_{\pi}^{\text{CP}} \{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}\}$$

⁹For example let $[\bar{\mathcal{G}}]$ be the KRK setup of [11] for which [17] shows a protocol GUC realizing authenticated channels in the static corruptions setting.

Remark 1. Formally, in the above lemma the syntax of the protocol π also needs to be adapted to the CP setting on the right hand side of the implication. For an arbitrary GUC protocol π , denote by $\bar{\pi}$ the CP protocol which works just like π , but while π_i sends a message \mathbf{m} to π_j via the adversary (i.e., by writing (\mathbf{m}, j) to its communication tape shared with the adversary), $\bar{\pi}_i$ instead uses $\mathcal{R}_{\text{ins}}(i, j)$ to send \mathbf{m} to $\bar{\pi}_j$. However, for the sake of clarity we do not distinguish between π and $\bar{\pi}$ as it is clear from the context which protocol is meant.¹⁰

1.2 Necessary Assumptions for CP

Having defined the CP framework and verified its 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

¹⁰In particular for the general feasibility result in Section 2.2 the input to the protocol compiler is $\bar{\pi}$ rather than π . This is because the fallback security (Definition 2.1.1) is defined in the CP setting rather than the GUC setting and so the security provided by the compiled protocol is analyzed in the presence of \mathcal{R}_{ins} rather than in the GUC model.

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} .

More precisely, in this section we prove statements of the form "if $\mathcal{F} \sqsubseteq^{CP} \mathcal{R}$ and \mathcal{F} has property P1 then \mathcal{R} must have property P2". We provide some conditions under which \mathcal{R} must be "isolating" (Lemma 1.2.2), "probabilistic" (Lemma 1.2.4) and finally "programmable" (Lemma 1.2.5). Taken together we see that a CP-complete resource (such as our mediator) must simultaneously have all three properties.

The proofs extract these properties by reason about the necessary properties the behavior of \mathcal{R} must have in the setting where all players are corrupt.¹¹

To this end we use the following reduction notion between functionalities.

Definition 1.2.1. *Let \mathcal{R} and \mathcal{F} be n -party CP resources and let $\mathcal{D}'[n]$ be the n -tuple of \mathcal{F} -hybrid adversaries that first corrupt their respective players and then act as the dummy adversary. We say that \mathcal{F} is contained in \mathcal{R} (written $\mathcal{F} \sqsubseteq^c \mathcal{R}$) if there exists n adversaries $\mathbf{A}_{[n]}$ (corrupting all players) such that for any efficient environment \mathcal{Z} (and protocol π):*

$$\text{CP-EXEC}_{\mathcal{D}'[n], \mathcal{Z}}^{\mathcal{F}} \approx \text{CP-EXEC}_{\pi, \mathbf{A}, \mathcal{Z}}^{\mathcal{R}}.$$

A benefit of defining the \sqsubseteq^c reduction to be so weak is that statements of the form $\mathcal{F} \sqsubseteq^c \mathcal{R}$ are often easy to verify for many interesting resources \mathcal{R} .¹² This

¹¹In a stand alone setting one might ask why complete corruptions are even an interesting case (for example the stand-alone collusion-free notion of [34] explicitly rules this case out). But for a composable security notion (for example with the application of modular protocol design in mind) it is vital to consider these executions since sub-protocols may be run but a set of entirely corrupt parties.

¹²See the following discussion on bounded collaborative channels.

makes it a good candidate for properties of functionalities in the following results. Next we define a simple and very weak channel which we will use in place of \mathcal{F} in the properties.

Bounded Collaborative Channels. Let resource $\mathcal{C}_{i,j}^c$ to be the ideal n -party functionality allowing up to $c \in \mathbb{N}$ bits to be sent by P_i to P_j (and from adversary A_i to A_j). Moreover $\mathcal{C}_{i,j}^c$ requires the cooperation of all other $n - 2$ players *and* n *adversaries* in order for the message to be delivered. More precisely the message from P_i is given to both A_i and A_j who can modify it (or even halt delivery) at will. The message is only delivered to P_j once all other $n - 2$ players and adversaries have submitted a special message `ok` (otherwise, on any other input $\mathcal{C}_{i,j}^c$ simply halts producing no output). We call such a channel a *bounded collaborative* channel.

We observe that statements of the type $\mathcal{C}_{i,j}^d \sqsubseteq^c \mathcal{F}$ are often easy to verify for many interesting examples such as when \mathcal{F} is a VCG auction mechanism [42, 14, 23] (i.e. a second price auction). Even if the winners fee is kept secret from all other bidders the statement $\mathcal{C}_{i,j}^d \sqsubseteq^c \mathcal{F}$ still holds for any value i, j and d .¹³ On the other hand for a voting mechanism with say t candidates, private votes, public outcome and the possibility to abstain the statement holds for any value of i, j and $d \leq \log t$. Therefore when trying to realize these kinds relevant resources the following lemma provides an easy property to verify for candidate real world communication resources that are to be used.

¹³For example a set of adversaries to such that the VCG mechanism contains $\mathcal{C}_{i,j}^d$ act as follows. Each adversary corrupts it's player (ignoring their valuations). Then A_j bids 2^d on behalf of receiver P_j , A_j bids $m \in [0, 2^n - 1]$ on behalf of sender P_i and all other adversaries bid 0 for their players. The winner of the auction (i.e. P_j) then outputs it's fee as the message received.

1.2.1 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} . Somewhat more formally suppose we wish to CP-realize functionality \mathcal{F} which allows for at most d bits of communication between players P_i and P_j . The following lemma rules out using any resource \mathcal{R} (with *arbitrary* protocol) which can be used (formally: contains) a c -bit bounded collaborative channel between these players whenever $c > d$. In fact, the requirements for obtaining such a channel are so weak that the lemma rules out using almost all standard communication channels as CP-complete resources as they all allow for *unbounded* amounts insecure communication between at least one pair of players.

Lemma 1.2.2. *Let \mathcal{R} and \mathcal{F} be n -player resources with $\mathcal{C}_{i,j}^c \sqsubseteq^C \mathcal{R}$ and $\mathcal{C}_{i,j}^d \sqsubseteq^{CP} \mathcal{F}$ then:*

$$\mathcal{F} \sqsubseteq^{CP} \mathcal{R} \implies c \leq d.$$

Proof. We use the transitivity of the CP relation to obtain $\mathcal{C}_{i,j}^d \sqsubseteq^{CP} \mathcal{R}$. Let $A_{[n]}$ be the adversaries such that $\mathcal{C}_{i,j}^c \sqsubseteq^C \mathcal{R}$ and suppose for the sake of contradiction that $c > d$. We briefly describe an environment \mathcal{Z} and \mathcal{R} -hybrid adversaries $B_{[n]}$ for which no simulators exist fooling \mathcal{Z} in the $\mathcal{C}_{i,j}^d$ -hybrid world. The environment samples a uniform random string $r \leftarrow \{0, 1\}^c$ and gives it to B_i . Then it waits for the output of r' of B_j and outputs 1 if and only if $r' = r$. Meanwhile, for each $s \in [n]$ adversary B_s acts as A_s intuitively turning \mathcal{R} into $\mathcal{C}_{i,j}^c$. For $s \notin \{i, j\}$ the adversary B_s uses ok as input to $\mathcal{C}_{i,j}^c$ and terminates. Adversary B_i reads it's input r from \mathcal{Z} and uses it as input for $\mathcal{C}_{i,j}^c$ (on behalf of players P_i) to be sent to P_j . Finally adversary B_j waits till it receives r' from $\mathcal{C}_{i,j}^c$ (intended for delivery to P_j)

and outputs r' to \mathcal{Z} before terminating. It is easy to verify that \mathcal{Z} will always output 1 during such an execution.

Clearly there are no good simulators for $\mathbb{B}_{[n]}$ in the $\mathcal{C}_{i,j}^d$ -hybrid world as by definition of a bounded collaborative channel there isn't enough bandwidth between the i^{th} and j^{th} honest (and adversarial) interfaces in $\mathcal{C}_{i,j}^d$. Thus with probability at least $1/2$ the output of the j^{th} simulator will not equal the input to the i^{th} simulator in which case \mathcal{Z} will output 0. \square

As an immediate application of Lemma 1.2.2 we obtain the following (informal) result ruling out most, if not all, the usual communication channels from being CP-complete. In particular, we view the following corollary provides significant justification for seaming strength of the assumptions made later on about the mediator resource. Also, the following implications for broadcast channels can be seen as extending the related impossibility result for broadcast channels from [33].

Corollary 1.2.3. *Let resource $\mathcal{R} \sqsubseteq^{CP} \mathcal{H}$ then if \mathcal{H} is a broadcast channel then \mathcal{R} is not CP-complete. Further if \mathcal{H} is a fully connected network of insecure, authenticated or secure channels then \mathcal{R} is not CP-complete.*

Remark On the Models of [34, 29]. When all players are corrupt their communication model contains $\mathcal{C}_{i,j}^c$ for any value of $c > 0$. Thus their notion of composition requires all parties in a protocol π to take part in all sub-protocols of a given protocol π (via observation of publicly verifiable events) in order to guarantee security of π . On the one hand this approach requires no honest party to trust another since each can verify the computation but on the other hand this type of composition scales badly in the presence of many users. Especially in a real world setting.

1.2.2 Independent Randomness

Besides the capability of enforcing bounded isolation another requirement for realizing several interesting functionalities is that the resource being used must be a *probabilistic* ITM. This stands in contrast with necessary assumptions on (even complete) resources in a setting with monolithic adversaries. On the other hand, in [29] for example, protocols make use of a ballot-box which contains inherent fresh randomness and the verifiable devices of [27] also have similar capabilities.

For positive integer c we denote by $\mathcal{F}_{\text{coin}}^c$ the 2-party coin flipping functionality which takes no input and outputs c uniform random coins to P_1 and P_2 .¹⁴

Lemma 1.2.4 (Randomness is Necessary). *For any integer $c > 0$ and resource \mathcal{R} if $\mathcal{F}_{\text{coin}}^c \sqsubseteq^{CP} \mathcal{R}$ then \mathcal{R} is a probabilistic ITM.*

The proof relies on the intuition that if \mathcal{R} is deterministic and all players are corrupt, the adversaries can completely determine the behavior of \mathcal{R} which they can use to transmit at least a bit between each other.

Proof. Suppose \mathcal{R} is a deterministic resource. We construct an environment \mathcal{Z} and pair of adversaries $A_{[2]}$ for which there are no simulators in the $\mathcal{F}_{\text{coin}}^c$ -hybrid world.

Suppose protocol π is such that $\mathcal{F}_{\text{coin}}^c \sqsubseteq_{\pi}^{CP} \mathcal{R}$ and for an execution of π we denote by Y the random variable describing its (common) output. Fix a random tape r_1 for π_1 . Then it must be that Y still has noticeable entropy even conditioned on the choice of r_1 . If this were not the case then an adversary corrupting only P_1 and running π_1 with tape r_1 on his behalf could not be simulated since the output

¹⁴We opt for this unusually general formulation as the result does not require $c = 1$ and so we can also capture ideal key exchange; another interesting functionality.

of the honest P_2 would always be fixed unlike an execution with \mathcal{F}_{coin}^c in the ideal world. More generally the number of random tapes r_2 for π_2 which cause Y to take on a fixed value is at most negligibly less than a $1/2^c$ fraction of all possible values. Thus for any r_1 it is easy to compute two tapes r_2^0 and r_2^1 which cause Y to take on two distinct values y^0 and y^1 respectively.

The adversaries each corrupt their respective player and run π on their behalf. Adversary A_1 uses random tape r_1 and if π_1 outputs y^0 it outputs 0 otherwise it outputs 1. The environment provides A_2 with input a uniform random bit b and A_2 uses the random tape r_2^b for its copy of π_2 . Thus by comparing the output of A_1 with the value of b over many executions the environment can distinguish the \mathcal{R} -hybrid world from any execution in a \mathcal{F}_{coin}^c -hybrid world. This is because the \mathcal{F}_{coin}^c functionality allows now communication between players and so a simulator for A_2 has no way to signal b to A_1 . \square

1.2.3 Programmability.

We stated the goal of finding a CP-complete resource. Unfortunately, (unlike in models with monolithic adversaries) in the CP setting no single resource can be CP-complete. Instead we consider *parameterized sets* of resources. More formally we say that a set $\mathcal{R} = \{\mathcal{R}_k\}_{k \in \mathcal{K}}$ of resources is *CP-complete* if for any functionality \mathcal{F} , there exists a resource $\mathcal{R}_k \in \mathcal{R}$ such that $\mathcal{F} \sqsubseteq^{CP} \mathcal{R}_k$. Indeed later on, the mediator which we define is really a (CP-complete) *class* of resources. In our main theorem we only guarantee full security for the protocols we construct when the correct element of the mediator class is used. We argue that any completeness in

the CP framework will require such a format.¹⁵

For index set \mathcal{K} let $\mathcal{R}_{\mathcal{K}} := \{\mathcal{R}_k \mid k \in \mathcal{K}\}$ denote a set of resources parameterized by elements of \mathcal{K} . We call $\mathcal{R}_{\mathcal{K}}$ a *programmable resource* if there exist $k_1, k_2 \in \mathcal{K}$ such that $\mathcal{R}_{k_1} \not\sqsubseteq^{CP} \mathcal{R}_{k_2}$.

Lemma 1.2.5. *A class of resources is CP-complete only if it is programmable.*

Proof. Let $\mathcal{C}_{i,j}^1$ and $\mathcal{C}_{i,j}^2$ denote the 1-bit and 2-bit (respectively) bounded collaborative channels between P_i and P_j (see the definition in Section 1.2.1). Let the class of resources $\mathcal{R}_{\mathcal{K}}$ be CP-complete. Then there exists $a, b \in \mathcal{K}$ and protocol π such that $\mathcal{C}_{i,j}^2 \sqsubseteq_{\pi}^{CP} \mathcal{R}_b$ and $\mathcal{C}_{i,j}^1 \sqsubseteq^{CP} \mathcal{R}_a$.

Assume for a moment that $\mathcal{R}_{\mathcal{K}}$ is not programmable. This implies that for some protocol ϕ we have $\mathcal{R}_b \sqsubseteq_{\phi}^{CP} \mathcal{R}_a$. Consider the n -tuple of adversaries $A_{[n]}$ living in the \mathcal{R}_a -hybrid world which each corrupt their respective players and then run protocol (stack) $\phi \circ \pi$ on the players' behalf using input supplied by the environment. In other words, the adversaries corrupt all players and then turn \mathcal{R}_a into \mathcal{R}_b using protocol ϕ and then turn \mathcal{R}_b into $\mathcal{C}_{i,j}^2$ using protocol π . The message transmitted from A_i and A_j (over $\mathcal{C}_{i,j}^2$) is supplied to A_i by the environment and is the output of A_j .

We construct environment \mathcal{Z} for which no simulators exist in the $\mathcal{C}_{i,j}^1$ -hybrid world (much as in the proof of Lemma 1.2.2). It selects a pair of random bits $r \in \{0, 1\}^2$ and gives them to A_i as input. Then it waits for A_j to produce output r' . Finally \mathcal{Z} outputs 1 if and only if $r = r'$. Clearly in the $\mathcal{C}_{i,j}^1$ -hybrid world the i^{th} and j^{th} adversary do not have enough bandwidth between them to simulate for this strategy which contradicts $\mathcal{C}_{i,j}^1 \sqsubseteq^{CP} \mathcal{R}_a$. \square

¹⁵To mitigate this we show that even if an arbitrary different resources is used (not necessarily from the mediator class) the protocol still attains a certain (GUC-like) security.

Chapter 2

Fallback Security

In this chapter we define GUC Fallback security and give a general feasibility result.

2.1 GUC Fallback Security

Without any further requirements CP security as defined in Definition 1.1.1 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 2.1.1 (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 properties:*

CP SECURITY: $\{\bar{\mathcal{G}}, \mathcal{F}\} \sqsubseteq_{\pi}^{CP} \{\bar{\mathcal{G}}, \mathcal{R}\}$

GUC FALLBACK: *For any efficient resource \mathcal{R}^* : $\{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}, \mathcal{F}\} \sqsubseteq_{\pi}^{CP} \{\bar{\mathcal{G}}, \mathcal{R}^*\}$.*

Recall that by default the GUC plain model is equipped with $[\mathcal{R}_{\text{ins}}]$. Thus applying Lemma 1.1.5 and omitting the redundant $[\mathcal{R}_{\text{ins}}]$ 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 in the presence 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 stronger 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.

Next we provide a useful tool for proving GUC fallback.

A “Dummy”-Resource Lemma. We show that instead working with arbitrary resource \mathcal{R}^* , it suffices to prove that GUC fallback holds for a special “dummy” resource \mathcal{M}^* , which behaves as a forwarder between all parties and a dedicated

adversary, e.g., the one with the smallest index.

More precisely the *dummy resource*, denoted by M^* , is the n -party functionality behaving as follows:

- When M^* receives a message by some party or adversary, M^* forwards it to a default adversary, e.g., the adversary of the party with the smallest index (wlog assume that this is A_1)
- When M^* receives a message of the form (send, m, ID) from A_1 , where ID is the identifier of some party or adversary, M^* sends m to the corresponding party or adversary, respectively.

As shown in the following lemma, M^* capture the behavior of the worst possible resource \mathcal{R}^* . In particular, the following lemma, allows us to reduce the GUC-fallback property (of Definition 2.1.1) to CP security in the M^* -hybrid model. In other words any statement about a monolithic adversary/simulator is translated to a statement in the M^* -hybrid model, where the simulator for Sim_1 plays the role of a coordinator among the simulators to allow the to mimic the behavior a monolithic simulator.

Lemma 2.1.2. *Let M^* denote the “dummy” resource defined above. Moreover let π be a protocol, $\bar{\mathcal{G}}$ be a shared functionality and \mathcal{F} be a CP functionality. Then for any efficiently computable resource \mathcal{R}^* it holds that:*

$$\{\bar{\mathcal{G}}, M^*, \mathcal{F}\} \sqsubseteq_{\pi}^{CP} \{\bar{\mathcal{G}}, M^*\} \implies \{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}, \mathcal{F}\} \sqsubseteq_{\pi}^{CP} \{\bar{\mathcal{G}}, \mathcal{R}^*\}.$$

Proof. Let \mathcal{R}^* be some efficiently computable resource, $\mathcal{D}^{\mathcal{R}^*}$ be the \mathcal{R}^* -hybrid dummy protocol and \mathcal{D}^{M^*} be the dummy M^* -hybrid protocol. One can easily verify the following two properties hold simultaneously:

1. $\{\bar{\mathcal{G}}, \mathbf{M}^*\} \sqsubseteq_{\mathcal{D}^{\mathcal{R}^*}}^{CP} \{\bar{\mathcal{G}}, \mathcal{R}^*\}$
2. $\{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}, \mathcal{F}\} \sqsubseteq_{\mathcal{D}^{\mathbf{M}^*}}^{CP} \{\bar{\mathcal{G}}, \mathbf{M}^*, \mathcal{F}\}$.

Indeed, for property (1) the simulator S_1 for A_1 need simply simulate the behavior of \mathcal{R}^* (the assumption of efficient computability of \mathcal{R}^* ensures that S_1 can do so). As \mathbf{M}^* is a forwarder between S_1 and the parties, this setting is indistinguishable from the \mathcal{R}^* -hybrid setting. For property (2) all simulators use \mathcal{R}_{ins} to redirect any messages originally destined for \mathbf{M}^* to S_1 . Moreover for any message of the form (send, m, ID) to be sent by A_1 to \mathbf{M}^* the simulator S_1 instead uses \mathcal{R}_{ins} to forward m directly to the party with the matching ID.

The above two properties applied in order together with Theorem 1.1.4 (the composition theorem) imply the lemma.¹

$$\{\bar{\mathcal{G}}, \mathbf{M}^*, \mathcal{F}\} \sqsubseteq_{\pi}^{CP} \{\bar{\mathcal{G}}, \mathbf{M}^*\} \implies \{\bar{\mathcal{G}}, \mathbf{M}^*, \mathcal{F}\} \sqsubseteq_{\pi}^{CP} \{\bar{\mathcal{G}}, \mathcal{R}^*\} \quad (2.1)$$

$$\implies \{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}, \mathcal{F}\} \sqsubseteq_{(\mathcal{D}^{\mathbf{M}^*})^{\pi}}^{CP} \{\bar{\mathcal{G}}, \mathcal{R}^*\} \quad (2.2)$$

$$\implies \{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}, \mathcal{F}\} \sqsubseteq_{\pi}^{CP} \{\bar{\mathcal{G}}, \mathcal{R}^*\} \quad (2.3)$$

The third implication follows from the composition theorem applied to the protocol $(\mathcal{D}^{\mathbf{M}^*})^{\pi}$ which forwards between \mathcal{Z} and π running in the $\{\bar{\mathcal{G}}, \mathcal{R}^*\}$ -hybrid world (instead of between \mathcal{Z} and \mathcal{F} as would protocol $\mathcal{D}^{\mathbf{M}^*}$). In other words, the composition implies $(\mathcal{D}^{\mathbf{M}^*})^{\pi} = \pi$ in the $\{\bar{\mathcal{G}}, \mathcal{R}^*\}$ -hybrid world. \square

¹Formally, we also make use of the convention of omitting dummy protocols from protocol call hierarchy (see Section 1.1). That is if $\mathcal{D}^{\mathcal{R}^*}$ is the dummy protocol for \mathcal{R}^* , by convention $\pi^{\mathcal{D}^{\mathcal{R}^*}} = \pi$.

2.2 A General Feasibility Result

We are now ready to prove a general feasibility result. Roughly speaking we describe an (efficient) programmable resource $\{\mathbf{M}_{\mathcal{F}}\}_{\mathcal{F} \in \{0,1\}^*}$, called the *mediator*, parametrized 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 $\{\bar{\mathcal{G}}, \mathbf{M}_{\mathcal{F}}\}$ -hybrid model.

Output-Synchronized Functionality. 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.

A bit more formally suppose the n -party functionality \mathcal{F} takes one input from each player P_i and then produces some (global) output y . Moreover we want to realize \mathcal{F} with a protocol π which requires strictly more than one such round.² Then \mathcal{Z} can use the following fact to distinguish: it knows that in an execution of π the adversary A_j can not compute y in less than two rounds. However, in general the simulators Sim which only have access to \mathcal{F} may not be able to communicate to each other how many times they have been activated. In particular Sim_i cannot keep track of how often say Sim_j has been activated. So Sim_i has no way of knowing when to deliver the output y to \mathcal{Z} .

We address this by introducing the class of *output-synchronized* functionalities.

²Observe that most general MPC protocols need more than one round for many interesting non-reactive functionalities.

For an 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 its 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} .
- Outputs of \mathcal{F} are not immediately delivered to their recipient. Rather they are recorded and 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 per player in an arbitrary order.

We note that intuitively this modification provides minimal synchronization between adversaries. In particular only the output delivery is synchronized but surprisingly, they can not for example, tell at any given moment during their execution which round another adversary believes they are in. Nevertheless it turns out that output synchronization is a sufficient condition enabling CP secure realization. For the remainder of this section we shall assume that \mathcal{F} is an *output-synchronized* functionality, i.e. is of the form $\widehat{\mathcal{F}'}$ for some functionality \mathcal{F}' .

To formalize our main feasibility theorem:

WELL-FORMED FUNCTIONALITY: As in [12], due to technical limitations inherited from UC-type frameworks, we restrict ourselves to the class of *well-*

formed functionalities[12], which intuitively includes all functionalities whose behavior does not depend on the identities of corrupted parties.

ABORTING FUNCTIONALITY: Consistent with existing literature on general secure computation tolerating a dishonest majority, we get security *with abort*. We capture this with the notion of an *aborting* functionality. These are such that they might accept a special input **ABRT**. Moreover, if any corrupted player provides input **ABRT** then the output of all players is also **ABRT**.

SETUP OFF-LINE PROTOCOL: A protocol which precedes all other computation and communication by it's only interaction with the setup $\bar{\mathcal{G}}$ is called *setup off-line*.³ Roughly speaking our construction has the mediator run a GUC protocol obliviously but verifiably on behalf of the players. If the π were not setup off-line then it is unclear by whom the interaction with $\bar{\mathcal{G}}$ could be performed. On the one hand players should remain oblivious to the messages of π (to obtain CP security) and the other hand the mediator should not be trusted to perform the interaction (so as to obtain GUC fallback).

GUC-AUTHCOMPLETE SETUP: We call a setup $\bar{\mathcal{G}}$ (i.e. a CP shared functionality) *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.

³We note that all known protocols satisfy this definition. In particular the protocols of [11, 17] are of this form.

We note that one such setup is the Key-Registration with Knowledge (KRK) functionality of [11, 17] when viewed as a CP shared functionality (we refer to Section 2.2.3 for details). In particular the results of [17] imply Property (1) and the results of [11] imply Property (2).

Ideally, we would like to state our feasibility result for *any* (albeit efficient) functionality \mathcal{F} . More realistically we require \mathcal{F} some additional (standard) properties.

Theorem 2.2.1 (General Feasibility Theorem). *Let $\bar{\mathcal{G}}$ be a GUC-AuthComplete CP setup there exists a programmable resource $\mathbf{M} = \{\mathbf{M}_{\mathcal{F}}\}$ such that for every well-formed aborting functionality \mathcal{F} there exists a protocol π which CP-realizes \mathcal{F} with GUC fallback in the $\mathbf{M}_{\mathcal{F}}$ -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 $\text{CP}(\pi)$ and parameters for resource \mathbf{M} . We point out that although we have assumed that \mathcal{F} is output-synchronized, the GUC property holds, for the same protocol $\text{CP}(\pi)$, even for the non-synchronized functionality, i.e., the one that results by removing from \mathcal{F} the synchronizing shell. The reason is that in the GUC-fallback setting the simulators can synchronize when to produce output by using the insecure channels

$\mathcal{R}_{\text{ins}}^*$

The proof of the theorem proceeds in two steps:

1. We show how to obtain from a GUC-AuthComplete setup $\bar{\mathcal{G}}$, e.g., the KRK [11], a setup off-line protocol π which CP realizes \mathcal{F} using insecure channels (in subsection 2.2.1).

2. Then we show how to compile π into protocol $\text{CP}(\pi)$ and resource $\mathbf{M}_{\mathcal{F}}$ which CP realize \mathcal{F} with GUC fallback (in subsection 2.2.2).

2.2.1 Bootstrapping from GUC

We prove the following lemma.

Lemma 2.2.2. *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 $\{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}, \mathcal{F}\} \sqsubseteq_{\pi}^{\text{CP}} \{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}\}$.*

Proof. Let $[\mathcal{F}]$ be the GUC analogue⁴ of \mathcal{F} . By assumption $\bar{\mathcal{G}}$ is GUC-AuthComplete so there exists a GUC protocol π such that $\{[\bar{\mathcal{G}}], [\mathcal{F}]\} \sqsubseteq_{\pi}^{\text{GUC}} [\bar{\mathcal{G}}]$. As $\bar{\mathcal{G}}$ is GUC-AuthComplete we can apply Lemma 1.1.6 to obtain the result. \square

2.2.2 Adding Fallback Security

We prove 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. The proof of Theorem 2.2.1 follows directly from this and Lemma 2.2.2.

Lemma 2.2.3. *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 $\{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}, \mathcal{F}\} \sqsubseteq_{\pi}^{\text{CP}} \{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}\}$.*

Then there exists an efficient resource $\mathbf{M}_{\mathcal{F}}$ (called the “mediator”) and protocol $\text{CP}(\pi)$ such that $\text{CP}(\pi)$ CP realizes \mathcal{F} with GUC fallback in the $\mathbf{M}_{\mathcal{F}}$ -hybrid model.

⁴Recall that in Section 1.2, for a CP functionality \mathcal{F} we defined $[\mathcal{F}]$ to be the monolithic extension of a CP functionality.

Proof Idea. The proof is constructive, i.e., we show how to construct $\text{CP}(\pi)$ and $\text{M}_{\mathcal{F}}$, and somewhat involved; it is inspired by the protocol of [3]. On the highest level the idea is to have the mediator $\text{M}_{\mathcal{F}}$ emulate an execution π “in his 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 2.1.1.

To obtain the GUC fallback property we need to make the following modifications to this approach. For each $i \in [n]$ the state of the emulated π_i is *shared* between P_i and $\text{M}_{\mathcal{F}}$ such that $\text{M}_{\mathcal{F}}$ 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 $\text{M}_{\mathcal{F}}$ 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 $\text{M}_{\mathcal{F}}$ 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 than 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].

Assumptions and Constraints. For simplicity we make the following assumptions.

NON-REACTIVE \mathcal{F} : We restrict ourselves to the case of non-reactive functionalities \mathcal{F} , aka Secure Function Evaluation (SFE); however at the cost of more involved notation, the same methods can be extended to the case of reactive functionalities.

Without loss of generality we also assume the following.

FIXED ROUNDS: We assume that π consists of a sequence of rounds. In each round, every player sends a fixed length message (from a public domain \mathcal{M}) to every other player. The rounds are *non-overlapping*, in the sense that messages which a party sends in some round (even when he is corrupted) depend on the party's input, randomness, and the messages that were sent to this party *up to that round*.⁵ Such a non-overlapping rounds structure can be obtained by known protocols, even in the asynchronous setting, by using synchronization techniques from [31, 30]. Furthermore, we assume that the number of rounds is fixed ⁶ and public, and that every party receives its π output in the same round. We note that most known general MPC protocols either already satisfy this and the next property, or can be trivially modified to do so. Specifically all GUC protocols of [11, 17] (which are presumably the protocols that will be used to instantiate our construction as in for example Corollary 2.2.6) satisfy this and the next property.

BLACK-BOX SIMULATION: We assume that π CP realizes \mathcal{F} with black-box simulation. We point out that this is already implied by Definition 1.1.1.

COMMITMENT PROTOCOL: We use a GUC-secure commitment scheme (Com, Dec) with the same message space \mathcal{M} as π . The scheme's existence follows from the facts that $\bar{\mathcal{G}}$ is GUC-AuthComplete and that the commitment functionality is well- formed and aborting. In particular [11] provides

⁵The adversary is rushing, hence, messages of corrupted parties in any round might depend on the messages which honest parties send in that round, *but not on messages that honest parties prepare for future rounds*.

⁶More generally, at the cost of elegance, we could have assumed only that the distribution describing the number of rounds in an honest execution of π be efficiently sampleable and independent of all inputs used by the players.

a concrete construction.

We sketch the remainder of the proof which is detailed in Sections 2.2.2.1 and 2.2.2.2:

IN SECTION 2.2.2.1: First, we describe how to share the state of π_i between a player and the mediator. Next we describe two useful programmable functionalities $\mathcal{F}_{\text{inp}} = \{\mathcal{F}_{\text{inp}}^{(\pi)}(A, B)\}_{\pi \in \{0,1\}^*}$ and $\mathcal{F}_{\text{comp}} = \{\mathcal{F}_{\text{comp}}^{(\pi)}(A, B)\}_{\pi \in \{0,1\}^*}$.⁷ These specify how from (a sharing of) some valid π -state for a party P_i , the mediator and P_i can compute a sharing of P_i 's next message, and thus an updated π -state from P_i . To avoid over-complicated notation, wherever it is clear from the context or redundant we might omit the protocol from the notation of the above functionalities, i.e., we write $\mathcal{F}_{\text{comp}}$ (resp. \mathcal{F}_{inp}) instead of $\mathcal{F}_{\text{comp}}^{(\pi)}$ (resp. $\mathcal{F}_{\text{inp}}^{(\pi)}$). In order to achieve the GUC fallback, the mediator should not be allowed to see the parties' shares of the shared protocol states. Therefore, instead of invoking the functionality \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$, the mediator and P_i execute a two-party protocol implementing the corresponding functionality. In fact, as we shall show, the code of P_i in the compiled protocol $\text{CP}(\pi)$ consists of playing as Bob in the evaluations of these two-party SFE's.

IN SECTION 2.2.2.2: As soon as we specify the code of the mediator and of the parties, the proof proceeds along the lines of the proof of [3]: We analyze the security of the protocol in the $\{\mathcal{F}_{\text{inp}}, \mathcal{F}_{\text{comp}}\}$ -hybrid world, and use composition to argue that replacing the hybrids by the corresponding protocols does not compro-

⁷We use Alice (A) and Bob (B), and not indices $i, j \in [n]$, for the parties executing \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$ to avoid confusion, as we intend to have the mediator play the role of Alice. Furthermore, for notational simplicity we will often omit the superscript (π) from this notation.

mise security. The main technical obstacle in this approach is that, similarly to UC and GUC, our framework does not specify how different functionalities could communicate with each-other. Hence, it is not straight-forward how to make the argument in the $\{\mathcal{F}_{\text{inp}}, \mathcal{F}_{\text{comp}}\}$ -hybrid world, as this would require the mediator to communicate with the hybrids. To overcome this difficulty, we consider the model where the mediator is replaced by an extra party $P_M \notin [n]$; P_M executes the code of the mediator and shares an ideal bidirectional secure channel with every P_i (i.e., we have a star network of ideal secure channels where P_M is in the middle of the star). Because the mediator is a functionality and cannot be corrupted, we will only consider in the security statements for the case when P_M is honest. However, in order to proof CP realization with GUC fallback, we will consider two versions/programs of P_M : the first one, denoted as $P_{M_{\mathcal{F}}}$, will behave as the mediator $M_{\mathcal{F}}$ and we will show that it allows for CP-realizing the functionality \mathcal{F} . The second one, denoted as P_{M^*} , will behave as the dummy mediator M^* , will allows us to achieve the security required for the GUC fallback.

2.2.2.1 A Detailed Description of $M_{\mathcal{F}}$ and $\text{CP}(\pi)$.

Intuitively the mediator $M_{\mathcal{F}}$ and the compiled protocol $\text{CP}(\pi)$ use the protocols π_{inp} and π_{comp} that securely implement the functionalities \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$, respectively, as follows.

INIT: Initially, mediator $M_{\mathcal{F}}$ sets the round index to $\rho := 0$.

INPUTS: The first interaction between a player P_i and $M_{\mathcal{F}}$ involves an invocation of π_{inp} . The purpose is to fix P_i 's input x_i and a random tape r_i used in the emulation of π_i . The output of π_{inp} for the mediator is commitments to

x_i and r_i while P_i receives the matching decommitments. As soon as every party has committed to his input and randomness, $M_{\mathcal{F}}$ augments the round index to $\rho := 1$.

COMPUTATION & OUTPUTS: All subsequent interactions between P_i and $M_{\mathcal{F}}$ consist of invocations of π_{comp} which takes input the current (shared) state of π_i together with an $(n - 1)$ -tuple of messages received by π_i during the current round. Initially $\mathcal{F}_{\text{comp}}$ checks that all inputs match and are of a valid form. Next it computes the next state of π_i together with an $(n - 1)$ -tuple of messages out produced by π_i for use in the next round $\rho + 1$. $\mathcal{F}_{\text{comp}}$ outputs the updated shared state and, to $M_{\mathcal{F}}$, the messages out. Finally, if ρ is the last round of π then P_i also receives its output of the computation (and otherwise it receive a default message \perp).

For any value of ρ which is not the last round-index, as soon as the mediator has completed one computation of π_{comp} with each player, $M_{\mathcal{F}}$ increments $\rho := \rho + 1$.

In the following, we describe in detail the functionalities \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$ (which implicitly depend on n and the code of π). As already mentioned, these are functionalities between two parties, Alice and Bob, where the role of Alice will be played by the mediator, and the role of Bob will be played by each of the parties in $[n]$. Recall that we denote by \mathcal{M} the message-space of protocol π . We assume that \mathcal{M} is equipped with a special abort message **ABRT**.

We begin the detailed description with some language and notation for working with the state of party P_i in protocol π_i (for simplicity, we denote P_i 's code in π as π_i).

Shared Protocol State. In a nutshell we model a shared state of a protocol as a pair of matching vectors of commitments/decommitments to the view of an execution. More precisely, recall that π has a fixed number of rounds $\mathbf{rmax} \in \mathbb{N}$, during each of which each player sends $(n - 1)$ messages from message space \mathcal{M} , one to each of the other player. We call an $(n - 1)$ -tuple of messages \vec{m} (containing one component either per sender or per receiver) a *message burst*. I.e. in each round each player receives a message burst and sends a message burst.⁸ At any point of the execution of protocol π , the received message bursts, together with the random tape r_i and input x_i completely determine the state of π_i . Thus we model the *state of π_i* as a quadruple of the form $\Pi_i = (x_i, r_i, M_i, \rho)$ where ρ is the index of the current round, and M_i is an $\mathbf{rmax} \times (n - 1)$ matrix with rows \vec{m}_a such that for all $a \in [\mathbf{rmax}]$ and $b \in [n - 1]$ the entry $m_{a,b} \in \mathcal{M} \cup \{\perp\}$. The goal of \mathcal{F}_{inp} will be to initialize Π_i and the goal of $\mathcal{F}_{\text{comp}}$ will be to iteratively and obliviously fill in the entries of M_i .

We call a state Π_i *sound for round \mathbf{r}* (where $0 \leq \mathbf{r} \leq \mathbf{rmax}$) if:

1. $\rho = \mathbf{r}$.
2. For all $j \in [\rho]$ each message burst $\vec{m}_j \in \mathcal{M}^{n-1}$.
3. For all $j \in \{\rho + 1, \dots, \mathbf{rmax}\}$ each message burst $\vec{m}_j = \perp^{n-1}$.

A *shared state* is a pair of quadruples (\vec{C}_i, \vec{D}_i) where \vec{C}_i maintained by P_i and \vec{D}_i is maintained by $M_{\mathcal{F}}$. Mediator $M_{\mathcal{F}}$ holds commitments to x_i and r_i , while commitments to M_i and ρ are held by P_i (and vice versa for the corresponding decommitments).⁹ In symbols P_i maintains the quadruple $\vec{C}_i = [\text{Dec}(x_i), \text{Dec}(r_i), \text{Com}(\rho), \text{Com}(M_i),]$ while $M_{\mathcal{F}}$ maintains $\vec{D}_i =$

⁸For the first round the received message bursts can take on arbitrary fixed default value.

⁹The commitment to matrix M_i is assumed to work component wise with the components arranged in a canonical order. For the sake of readability we use matrix notation.

$[\text{Com}(x_i), \text{Com}(r_i), \text{Dec}(\rho), \text{Dec}(M_i)]$. We call a shared state (\vec{C}_i, \vec{D}_i) *valid for round* ρ if:

1. The matching components of \vec{C}_i and \vec{D}_i are valid commitment/decommitments pairs.
2. Let Π_i be the vector of decommitted values. Then Π_i is sound for round ρ .

Functionality $\mathcal{F}_{\text{inp}}(A, B)$. The goal of this functionality is to prepare a shared state of π_i between P_i (playing as Bob) and $M_{\mathcal{F}}$ (plying as Alice). This includes fixing an input and random tape, initializing all incoming message bursts to a default value and then committing to the state using a GUC commitment protocol (Com, Dec) .

For sake of simplicity, we describe the functionality \mathcal{F}_{inp} (see Figure 2.1) in a form of a function. A CP functionality can be trivially obtained from our description, by considering the corresponding SFE functionality in the GUC framework [8, 11] and modifying it by applying the transformation $[\cdot]$ along the lines of Section 1.1.4.1.

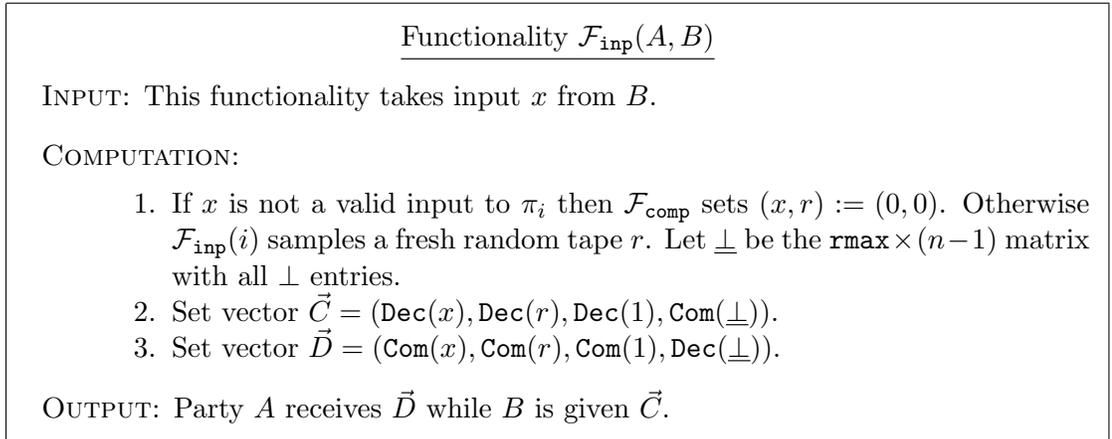


Figure 2.1: A detailed description of functionality $\mathcal{F}_{\text{inp}}(A, B)$.

Functionality $\mathcal{F}_{\text{comp}}(A, B)$. This functionality is accessed by P_i (playing as Bob) and $M_{\mathcal{F}}$ (playing as Alice). Its goal is to update the state of π_i and compute an outgoing message burst for the next round.

In more detail it takes input a shared state for π_i and, from $M_{\mathcal{F}}$, an (incoming) messages burst \vec{m} received by π_i during that round. The functionality checks that the shared state is valid for some $\rho \in [\text{rmax}]$. If not or if the state or \vec{m} contain an ABRT message then $\mathcal{F}_{\text{comp}}$ set's the outgoing message burst to $\{\text{ABRT}\}^{n-1}$. Otherwise $\mathcal{F}_{\text{comp}}$ updates the state with the new message tuples. If $\rho = \text{rmax}$ then $\mathcal{F}_{\text{comp}}$ computes and sets y equal to the output of π_i . Otherwise it sets y to a default value. Finally, $\mathcal{F}_{\text{comp}}$ outputs an updated sharing of the state to the parties, as well as y for P_i and the outgoing message burst for round $\rho + 1$ to $M_{\mathcal{F}}$.

For a detailed description of $\mathcal{F}_{\text{comp}}(A, B)$ see Figure 2.2. For simplicity we describe \mathcal{F}_{inp} in form of a function; as in the case of \mathcal{F}_{inp} , the actual functionality can be derived by taking the corresponding GUC functionality and modifying it along the lines of Section 1.1.4.1.

Remark 2 (Minimal Signaling by Abort). The specification of $\mathcal{F}_{\text{comp}}$ ensure that by making the protocol abort, an adversary might signal no more than one bit to other adversaries. In particular, no adversary (other than the one that caused the abort) is informed about when or by whom the message that caused the abort was sent. This is consistent with other recent results in collusion-free computation [27, 3].

The protocols $\pi_{\text{inp}}(A, B)$ and $\pi_{\text{comp}}(A, B)$: As already mentioned, the mediator $M_{\mathcal{F}}$ and the parties in the compiled protocol $\text{CP}(\pi)$ do not have access to the two-party functionalities \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$. Rather the behavior of these functionalities is emulated by corresponding two-party protocols π_{inp} and π_{comp} ,

Functionality $\mathcal{F}_{\text{comp}}(A, B)$

INPUT: This functionality takes input integer ρ , incoming message burst $\vec{\text{in}}$ and vector \vec{D} from A as well as vector \vec{C} from B .

COMPUTATION:

1. Initialize flag **abort** := 0.
2. If (\vec{C}, \vec{D}) is not a valid shared state for round ρ set **abort** := 1.
3. Let $\Pi_i = (x_i, r_i, M_i)$ be the state of π_i shared in (\vec{C}, \vec{D}) . If M_i has an entry **ABRT** then set **abort** := 1.
4. If for some $j \in [n - 1]$ burst $\vec{\text{in}}$ has component $\text{in}_j = \text{ABRT}$ then set **abort** := 1.
5. If **abort** = 1 then set outgoing message burst $\vec{\text{out}} = \{\text{ABRT}\}^{n-1}$. Otherwise:
 1. Set the received message burst for round ρ in Π_i to $\vec{\text{in}}$. In symbols: $\vec{m}_\rho = \vec{\text{in}}$.
 2. Run π_i with state Π_i obtaining the $\rho + 1$ outgoing message burst $\vec{\text{out}}$ according to the honest protocol.
3. Build a fresh sharing (\vec{C}', \vec{D}') of Π_i with fresh commitments.
4. If $\rho < \text{rmax}$ set $y := \perp$. Otherwise if **abort** = 1 set $y = \text{ABRT}$. Otherwise run π_i with Π_i and store its output as y .

OUTPUT: Party A receives $(\vec{\text{out}}, \vec{D}')$ while B is given (y, \vec{C}') .

Figure 2.2: A detailed description of functionality $\mathcal{F}_{\text{comp}}(A, B)$.

which exchange messages with the mediator. In fact, one can verify that for CP-realizing \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$ when ideally secure communication channels among the parties are assumed,¹⁰ we can use protocols that GUC realize the corresponding GUC functionalities $[\mathcal{F}_{\text{inp}}]$ and $[\mathcal{F}_{\text{comp}}]$, as long as these protocols are setup-off-line (Lemma Lemma 1.1.6). Such protocols are described in [11]. As syntactic sugar, we denote by π_{inp}^A and π_{inp}^B the code of Alice and Bob in π_{inp} , respectively; analogously we define π_{comp}^A and π_{comp}^B .

¹⁰By ideally secure channels we mean channels that resemble the communication means of the interface between the parties and the mediator. We refer to the following section for a detailed description.

The Mediator $M_{\mathcal{F}}$. We now turn to describing the mediator $M_{\mathcal{F}}$. In a nutshell, it's role is to emulate an execution of π . This is done by maintaining an n -tuple of shares of states for π (one per player), a counter ρ to keep track of the current round in the emulation, and two n -tuple of message bursts \vec{m}_i which contain the messages exchanged during the current round.

Initially $\rho := 0$. The mediator can perform two types of operations. Upon request by a player P_i the mediator engages with P_i into an execution of $\pi_{\text{inp}}(M_{\mathcal{F}}, P_i)$ for evaluating $\mathcal{F}_{\text{inp}}(M_{\mathcal{F}}, P_i)$ to initialize P_i 's state. This is done exactly once for each $i \in [n]$. Once all states have been initialized $M_{\mathcal{F}}$ sets $\rho := 1$. The second operation (also upon request by some P_i) consists of invocation of protocol $\pi_{\text{comp}}(M_{\mathcal{F}}, P_i)$ for evaluating $\mathcal{F}_{\text{comp}}(M_{\mathcal{F}}, P_i)$ on the current (sharing of) the state. These calls are performed upon any request by any player unless the state of that player has not been initialized yet. In particular, multiple calls to $\pi_{\text{comp}}(M_{\mathcal{F}}, P_i)$ for the same P_i may be made *sequentially* during any given round –though only one is needed per round (for details on why accepting multiple computation-requests per round is useful see Remark 3). More precisely, during each round of the emulation the mediator makes at least one call to $\pi_{\text{comp}}(M_{\mathcal{F}}, P_i)$ for every value of $i \in [n]$. Once this is done $M_{\mathcal{F}}$ increments $\rho := \rho + 1$.

For a formal description of the mediator $M_{\mathcal{F}}$ see Figure 2.3.

Remark 3 (Round Obliviousness). As mentioned in Section 2.1, the functionality \mathcal{F} which we compute is output-synchronized. This implies that it allows the simulators to ensure that they produce their output to \mathcal{Z} only when the last (simulated) round kicks in. However, the simulators are not aware of the following events: (1) every party has been activated for giving input, and (2) the same simulator is activated multiple times in the same round. For this reason, we have designed the

The Mediator $M_{\mathcal{F}}$

ON START: Set $\rho := 0$ and for each $i \in [n]$ initialize the pair of message bursts $\vec{\text{in}}_i = \vec{\text{out}}_i = \{0\}^{n-1}$.

INITIALIZE STATE: Upon receipt of the first message of the form $(\text{input_session}, SID)$ from P_i invoke protocol $\pi_{\text{inp}}(M_{\mathcal{F}}, P_i)$ and, once it produces output store it as \vec{D}_i . If all other players have already sent this message to $M'_{\mathcal{F}}$ then set $\rho := 1$. (Any future messages of this type from P_i are ignored.)

NEXT STATE: Upon receipt of a message of the form $(\text{compute_session}, SID)$ from P_i :

1. If no `input_session` message has been received from P_i or an instance of $\pi_{\text{inp}}(M_{\mathcal{F}}, P_i)$ or $\pi_{\text{comp}}(M_{\mathcal{F}}, P_i)$ is currently executed, then do nothing.
2. Otherwise:
 1. If all players (including P_i) have already sent a message `compute_session` during round ρ then increment $\rho := \rho + 1$.
 2. Invoke $\pi_{\text{comp}}(M_{\mathcal{F}}, P_i)$ with input $(\vec{\text{in}}_i, \vec{D}_i)$; once $\pi_{\text{comp}}(M_{\mathcal{F}}, P_i)$ generates output, store it as $(\vec{\text{out}}_i, \vec{D})$.

Figure 2.3: A detailed description of the mediator $M_{\mathcal{F}}$.

mediator to hide this information from the adversaries as well. In particular, whenever the mediator receives a `compute_session` request from some P_i , he invokes the protocol $\pi_{\text{comp}}(M_{\mathcal{F}}, P_i)$ for updating the state, irrespective of whether P_i 's state for this round has already been updated (by another invocation of $\pi_{\text{comp}}(M_{\mathcal{F}}, P_i)$). By definition of $\mathcal{F}_{\text{comp}}$, if P_i 's state for the current round has already been computed, it will not be changed, but a new sharing of this state will be computed to hide from the adversaries this information.

The Protocol $\text{CP}(\pi)$. Having described the code of $M_{\mathcal{F}}$, the protocol $\text{CP}(\pi)$ can be described in a straight-forward manner. In a nutshell, the code $\text{CP}_i(\pi)$ of each party P_i is described as follows: whenever $M_{\mathcal{F}}$ invokes any of the protocols $\pi_{\text{inp}}(M_{\mathcal{F}}, P_i)$ or $\pi_{\text{comp}}(M_{\mathcal{F}}, P_i)$, the party P_i executes his part (i.e., Bob's role) of

that protocol. More precisely, P_i receives input x_i from the environment \mathcal{Z} and maintains a share \vec{C} of the state of π_i . The vector \vec{C} is initialized with an invocation of $\pi_{\text{inp}}(\mathbf{M}_{\mathcal{F}}, P_i)$ and updated via multiple (sequential) invocations of $\pi_{\text{comp}}(\mathbf{M}_{\mathcal{F}}, P_i)$. Eventually π_{comp} returns the output of π_i which is passed back to \mathcal{Z} . For a formal description see Figure 2.4.

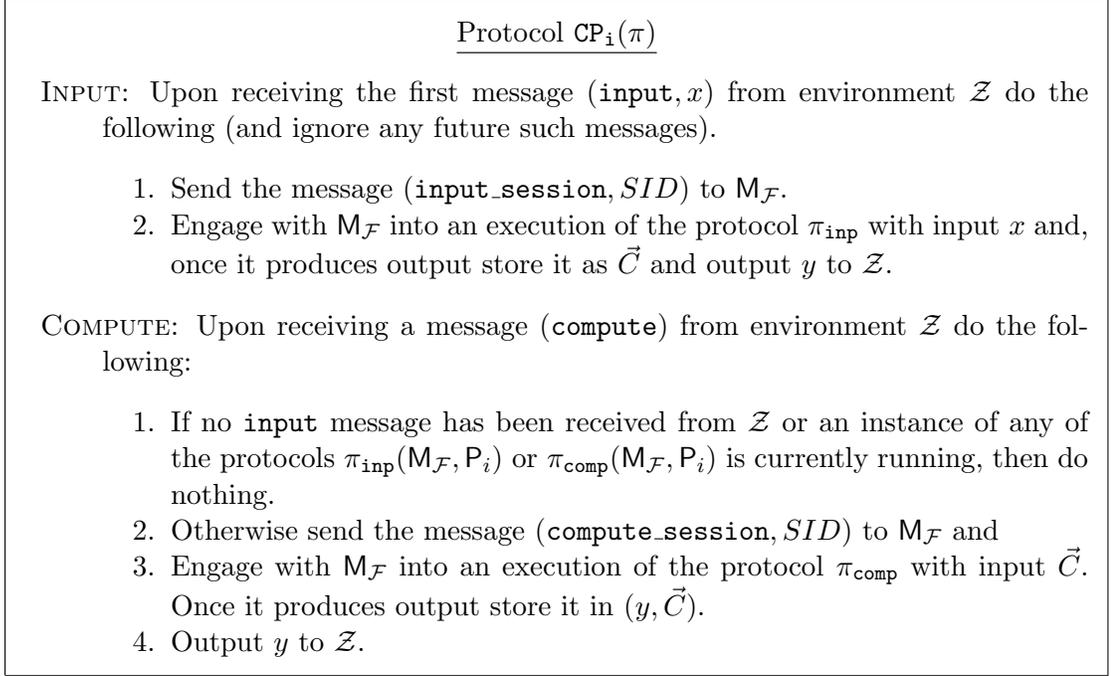


Figure 2.4: A detailed description of protocol $\text{CP}_i(\pi)$.

2.2.2.2 Completing the Proof of Lemma 2.2.3

To complete the proof, we show for the above described mediator $\mathbf{M}_{\mathcal{F}}$ and protocol $\text{CP}(\pi)$, the statement of Lemma 2.2.3 holds. To help the reader we give an outline of the rest of proof: We consider a world where the mediator is replaced by an extra party $P_M \notin [n]$. P_M might be programmed in one of two different manners (depending on the program of P_M we get CP or GUC-fallback security):

in one version, denoted as $P_{M_{\mathcal{F}}}$, he executes exactly the code of the mediator $M_{\mathcal{F}}$, whereas in the other version, denoted as P_{M^*} , the code of the dummy mediator M^* (see Section 2.1). P_M communicates with the parties over bi-directional ideal communication channels $\mathcal{F}_{\text{ideal}}$. These are functionalities that resemble the interaction of the mediator with the party, e.g., in $\mathcal{F}_{\text{ideal}}(P_M, P_i)$, P_M might send a message which is immediately delivered to P_i . Furthermore, $\mathcal{F}_{\text{ideal}}$ allows P_M to exchange messages with the adversaries in $A_{[n]}$ (note that $P_{M_{\mathcal{F}}}$ will never use this additional functionality, but P_{M^*} will). We denote the star-network of such ideal channels where P_M sits in the middle as $\mathcal{F}_{\text{ideal}}^* = \{\mathcal{F}_{\text{ideal}}(P_M, P_i), \mathcal{F}_{\text{ideal}}(P_i, P_M) \mid i \in [n]\}$. Apart from this star-network there is no other communication resource among the parties.

Denote by π^* the $(n + 1)$ -party protocol, i.e., among the parties in $[n] \cup \{P_M\}$, in which P_M executes his program (i.e., either $P_{M_{\mathcal{F}}}$ or P_{M^*}) and for each $i \in [n]$ party P_i executes his code from $\text{CP}_i(\pi)$, where the interaction with the resource $M_{\mathcal{F}}$ or M^* is replaced by interaction with P_M . We compare the execution of π^* with the one of $\text{CP}(\pi)$ *conditioned on P_M being honest*, and show that: ¹¹

1. When $P_M = P_{M_{\mathcal{F}}}$: if the functionality \mathcal{F} (among the parties in $[n]$) is CP realized from π^* then it is also CP-realized from $\text{CP}(\pi)$ (in the $M_{\mathcal{F}}$ -hybrid model)
2. When $P_M = P_{M^*}$: if the functionality $\{\mathcal{F}, \mathcal{R}_{\text{ins}}\}$ is CP-realized from π^* then it is also CP realized from $\text{CP}(\pi)$ (in the M^* -hybrid model)

Next, we show that

3. When $P_M = P_{M_{\mathcal{F}}}$: the protocol π^* CP-realizes the functionality \mathcal{F} ; this

¹¹In slight abuse of notation we use $\text{CP}(\pi)$ for denoting both the M^* -hybrid and the $M_{\mathcal{F}}$ -hybrid protocol.

together with Statement 1 imply that the protocol $\text{CP}(\pi)$ CP-realizes \mathcal{F} in the $\mathcal{M}_{\mathcal{F}}$ -hybrid model.

4. When $\mathsf{P}_{\mathcal{M}} = \mathsf{P}_{\mathcal{M}^*}$: the protocol π^* CP securely realizes the functionality $\{\mathcal{F}, \mathcal{R}_{\text{ins}}\}$; this together with Statement 2 imply that the protocol $\text{CP}(\pi)$ CP-realizes $\{\mathcal{F}, \mathcal{R}_{\text{ins}}\}$ in the \mathcal{M}^* -hybrid model (GUC fallback).

Steps 3 and 4 are first proved for the setting where π^* is executed in the $\{\mathcal{F}_{\text{inp}}, \mathcal{F}_{\text{comp}}\}$ -hybrid model (i.e., invocations of π_{inp} and π_{comp} are replaced by calls to \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$, respectively). The actual statements are then obtained by applying the composition theorem.

In the remaining of this section we prove the above steps.

Steps 1 and 2: The first two steps are relatively easy. Indeed, when $\mathsf{P}_{\mathcal{M}} = \mathsf{P}_{\mathcal{M}_{\mathcal{F}}}$ then the environment witnesses essentially the same interaction as in the case where $\mathcal{M}_{\mathcal{F}}$ is playing instead of $\mathsf{P}_{\mathcal{M}}$ (recall that we assume $\mathsf{P}_{\mathcal{M}}$ to be honest). The same holds for $\mathsf{P}_{\mathcal{M}} = \mathsf{P}_{\mathcal{M}^*}$ and \mathcal{M}^* . We only need to take care of the following (technical) issue: In the execution of π^* we have one more party and adversary than in the execution of $\text{CP}(\pi)$. We argue that this is not a problem: indeed, one can consider the extra party $\mathsf{P}_{\mathcal{M}}$ also in the $\mathcal{M}_{\mathcal{F}}/\mathcal{M}^*$ -hybrid world, but have him do nothing, i.e., ignore all inputs from \mathcal{Z} . As, by assumption, $\mathsf{P}_{\mathcal{M}}$ is honest and $\mathcal{F}_{\text{ideal}}^*$ does not interact with $\mathsf{P}_{\mathcal{M}}$'s adversary, such an extension cannot change \mathcal{Z} 's distinguishing advantage.

Step 3 We next show that when $\mathsf{P}_{\mathcal{M}} = \mathsf{P}_{\mathcal{M}_{\mathcal{F}}}$ then the protocol π^* CP-securely realizes the functionality \mathcal{F} . We show this statement in two sub-steps:

Sub-step 3.1. We consider the protocol $\pi_{\mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{ideal}}^*}^*$ which works as follows:

$\pi_{\mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{ideal}}^*}^*$ is a version of π^* where calls to protocols π_{inp} and π_{comp} are replaced by calls to the functionalities \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$. As syntactic sugar we denote the star-networks of such functionalities, where P_M sits in the middle, by $\mathcal{F}_{\text{inp}}^* = \{\mathcal{F}_{\text{inp}}(P_M, P_i) \mid i \in [n]\}$ and $\mathcal{F}_{\text{comp}}^* = \{\mathcal{F}_{\text{comp}}(P_M, P_i) \mid i \in [n]\}$

Claim 2.2.4. *Assuming $P_M = P_{M_{\mathcal{F}}}$ is honest, the protocol $\pi_{\mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{ideal}}^*}^*$ CP-securely realizes the functionality \mathcal{F} .*

Proof. We show the claim for the notion of SCP security, i.e., where the environment invokes only one instance of $\pi_{\mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{ideal}}^*}^*$ and the assumed adversaries are “dummy”. The idea for the simulation is the following: Before the output is generated from \mathcal{F} , the only thing that any adversary A_i of some corrupted P_i sees is the commitments to the state that are output from \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$ and the de-commitment information on the input x . Because these commitments are generated with independent randomness, they can be easily simulated. In the following we sketch how the simulator Sim_i for A_i works. The program of all simulators is the same, except from Sim_1 who, in addition to what the other simulators do, starts off by sending the maximum round rmax to the (synchronizing shell of) the functionality \mathcal{F} . The simulator Sim_i works as follows (we only consider the case where P_i is corrupted, as in the case where P_i is honest A_i does not see anything from the interaction):

1. When Sim_i receives from \mathcal{Z} a request to invoke \mathcal{F}_{inp} with input x' , if a request of this type has already been received he ignores it. Otherwise: he sends x' to the functionality \mathcal{F} , who acknowledges the reception. Subsequently, Sim_i choose some random value r'_i , and computes decommitment information $\text{Dec}(x'_i)$ and $\text{Dec}(r'_i)$; furthermore, Sim_i computes com-

mitments $\text{Com}(1)$ and $\text{Com}(\perp)$. Finally, Sim_i hands (y', \vec{C}) to \mathcal{Z} , where $\vec{C} = (\text{Dec}(x'_i), \text{Dec}(r'_i), \text{Com}(1), \text{Com}(\perp))$.

2. When Sim_i receives from \mathcal{Z} a request to invoke $\mathcal{F}_{\text{comp}}$ with input x' : he notifies \mathcal{F} , which acknowledges the activation (to ensure output synchronization) and answers back to Sim_i with either y or \perp . Sim_i sets y' to this answer. Sim_i computes (fresh) decommitment information $\text{Dec}(x'_i)$ and $\text{Dec}(r)$; furthermore, Sim_i computes (fresh) commitments $\text{Com}(1)$ and $\text{Com}(\perp)$. Finally, Sim_i hands (y', \vec{C}) to \mathcal{Z} , where $\vec{C} = (\text{Dec}(x'_i), \text{Dec}(r'_i), \text{Com}(1), \text{Com}(\perp))$.

We argue that the simulators described above produce a view, for \mathcal{Z} , which is indistinguishable from the view of the execution of $\pi_{\mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{ideal}}^*}$. One of the difficulties is the fact that at round \mathbf{rmax} the output y' is set to the actual output of the computation. However, this behavior is simulated with the help of the synchronizing-shell, which ensures that also the simulators will be informed (and given the output) as soon as the simulated protocol reaches round \mathbf{rmax} . The indistinguishability of the real and the simulated view follows then from the GUC security of the used commitment scheme. Indeed, the only difference in the real/simulated view is that in the real view \mathcal{Z} sees commitments on the actual round ρ and state M_i of the protocol (or to ρ and ABRT 's in case it forces an early abort), whereas in the simulated execution those are commitments to the default values $\rho = 1$ and $M_i = \perp$. The GUC security of the commitment scheme ensures that the environment cannot distinguish between these two sequences of commitments. \square

Sub-step 3.2. Second, we observe that, by definition, π_{inp} and π_{comp} are CP secure realizations of functionalities \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$, respectively. The composition theo-

rem applied to Claim 2.2.4 implies that π^* CP-securely realizes the functionality \mathcal{F} .

Step 4. We next show the GUC fallback, i.e., that when $P_M = P_{M^*}$ then the protocol π^* CP-securely realizes the functionality $\{\mathcal{F}, \mathcal{R}_{\text{ins}}\}$, where \mathcal{R}_{ins} denotes a complete network of insecure channels among the parties in $[n]$. This is also split in two sub-steps, along the lines of Step 3:

Sub-step 4.1. As in Step 3.1, we start with the $\{\mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{ideal}}^*\}$ -hybrid protocol $\pi_{\mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{ideal}}^*}^*$ (note however that now P_M executes the code defined by P_{M^*}).

Claim 2.2.5. *Assuming $P_M = P_{M^*}$ is honest, the protocol $\pi_{\mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{ideal}}^*}^*$ CP-securely realizes the functionality $\{\mathcal{F}, \mathcal{R}_{\text{ins}}\}$.*

Proof. By assumption of Lemma 2.2.3 we know that $\{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}, \mathcal{F}\} \sqsubseteq_{\pi}^{CP} \{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}\}$. In order to show the statement of the claim, it suffices to show that $\pi_{\mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{ideal}}^*}^*$ CP emulates the \mathcal{R}_{ins} -hybrid protocol π .¹² Indeed, this would imply that

$$\{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}, \mathcal{F}\} \sqsubseteq_{\pi_{\mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{ideal}}^*}^*}^{CP} \{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}, \mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{ideal}}^*\} \quad (2.4)$$

hence, in order to obtain the statement of the claim from Reduction 2.4 we need to remove \mathcal{R}_{ins} from the left-hand side. To this direction, we observe that

$$\{\bar{\mathcal{G}}, \mathcal{R}_{\text{ins}}\} \sqsubseteq_{\mathcal{D}_{\mathcal{F}_{\text{ideal}}^*}^{CP}} \{\bar{\mathcal{G}}, \mathcal{F}_{\text{ideal}}^*\}, \quad (2.5)$$

where $\mathcal{D}_{\mathcal{F}_{\text{ideal}}^*}^{CP}$ denotes the dummy $\mathcal{F}_{\text{ideal}}^*$ -hybrid protocol. Indeed, similar to the dummy-mediator lemma, in order to simulate the behavior of \mathcal{R}_{ins} in the $\mathcal{F}_{\text{ideal}}^*$ -hybrid world, the simulators use the party P_{M^*} that allows them to communicate

¹²Recall that π is an n -party protocol which includes no instruction from P_M .

arbitrarily. Using Reduction 2.5 and the composition theorem, we can remove \mathcal{R}_{ins} from the right side of Reduction 2.4.

In the remaining of the proof we show that $\pi_{\mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{ideal}}^*}$ CP emulates protocol π . To this direction, we considered a modified version of the protocol $\pi_{\mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{ideal}}^*}$ as follows: the hybrid functionalities $\mathcal{F}_{\text{comp}}$ and \mathcal{F}_{inp} are removed and each honest P_i executes the code of these functionalities internally and sends to P_M his outputs over the channel $\mathcal{F}_{\text{ideal}}(P_i, P_M)$. More concretely, any call to $\mathcal{F}_{\text{inp}}(P_M, P_i)$ or $\mathcal{F}_{\text{comp}}(P_M, P_i)$ (in protocol $\pi_{\mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{ideal}}^*}$) is replaced by P_M handing his input (for \mathcal{F}_{inp} or $\mathcal{F}_{\text{comp}}$) to P_i , who executes the code of the corresponding functionality and sends P_M his output (i.e., his share of the state and the next round messages) and to the environment the output that he (i.e., P_i) would receive from the functionality. We denote this modified protocol as $\pi^{*'}$. We show that:

- i. the protocol $\pi_{\mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{ideal}}^*}$ CP-emulates the protocol $\pi^{*'}$
- ii. the protocol $\pi^{*'}$ CP-emulates the protocol π

and then use the composition theorem to conclude that $\pi_{\mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{ideal}}^*}$ CP-emulates the protocol π .

We start with Step i: We argue that protocol $\pi_{\mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{ideal}}^*}$ SCP-emulates the protocol $\pi^{*'}$ (from this we can directly derive that it also CP emulates it by means of Theorem 1.1.3): We describe a simulator-profile $\text{Sim}_{[n]}$ for the dummy adversary-profile $\mathbf{A}_{[n]}$ (denote by $\mathcal{I} \subseteq [n]$ the set of corrupted players):

1. For $i \in ([n] \cup \{P_M\}) \setminus (\mathcal{I} \cup \{1\})$ Sim_i simply follows \mathcal{Z} 's instructions as the dummy adversary would do. This is a sound simulation because both the adversary and the simulator of such an honest party is completely cut-off by the assumed hybrids).

2. Let us consider the adversary A_i of some corrupted $i \in \mathcal{I}$: other than performing local computation, \mathcal{Z} might give A_i the instruction to send a message to P_{M^*} over the ideal channel. When the simulator Sim_i receives such an instruction, he simply sends this message to P_{M^*} . Any message which Sim_i receives from P_{M^*} , he forwards it to \mathcal{Z} . Additionally, Sim_1 might receive from \mathcal{Z} messages of the form (send, m, ID) to be sent to P_{M^*} ; when Sim_1 receives such a message he simply forwards it to P_{M^*} .

It is straight-forward to verify that with the above simulator protocol $\pi_{\mathcal{F}_{\text{comp}}^*, \mathcal{F}_{\text{inp}}^*, \mathcal{F}_{\text{ideal}}^*}$ SCP-emulates the protocol $\pi^{*'}$.

We go on with Step ii: The idea of the simulation is the following: the distribution of π -messages exchanged by parties executing $\pi^{*'}$ is identical to the input/output distribution of parties executing π . Hence the simulators needs only to simulated the commitments which P_{M^*} receives as part of the shared state between P_{M^*} and honest parties P_i .

Because the simulators can use the complete network of insecure channel to coordinate their actions, we assume that the only simulator who has a non-trivial program is Sim_1 . Every Sim_i with $i \neq 1$ acts as “slave” of Sim_1 , i.e. forwards everything he receives to Sim_1 and follows Sim_1 ’s instructions. The simulator Sim_1 works as follows:

1. He makes sure, by using \mathcal{R}_{ins} to co-ordinate with the slave-simulators, that whenever \mathcal{Z} expect to see some message from some adversary A_i , it is Sim_i who outputs the corresponding simulated message to \mathcal{Z} .
2. He keeps track of the current round index ρ . Sim_1 can do that as he has the overview of the execution of π , i.e., he gets informed for every activation

coming from \mathcal{Z} .

3. For each round ρ (initially $\rho = 0$) and each honest party P_i , Sim_1 stores in a variable M'_i the message-bursts $\vec{m} \in (\mathcal{M} \cup \{\perp\})^{n-1}$ that P_i receives in the execution of protocol π (M'_i is updated similar to how \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$ update M_i). As in the original protocol, M'_i is an $\text{rmax} \times (n - 1)$ matrix initially filled with \perp 's.
4. Whenever in the simulated $\pi^{*'} P_{M^*}$ would interact with some honest party P_i to have him run the code of π_{inp} , Sim_1 simulates P_{M^*} 's output, i.e., P_{M^*} 's share $\vec{D} = [\text{Com}(x_i), \text{Com}(r_i), \text{Dec}(0), \text{Dec}(M_i)]$, by $\vec{D}' = [\text{Com}(0), \text{Com}(r'_i), \text{Dec}(0), \text{Dec}(M'_i)]$ for some uniformly random value r'_i . Finally, Sim_1 sets P_{M^*} 's output to (\vec{out}', \vec{D}') .
5. Whenever in the simulated $\pi^{*'} P_{M^*}$ would interact with some honest party P_i to have him run the code of π_{comp} , Sim_1 simulates P_{M^*} 's output, i.e., P_{M^*} 's share $\vec{D} = [\text{Com}(x_i), \text{Com}(r_i), \text{Dec}(\rho), \text{Dec}(\perp)]$ and the message-burst \vec{out} for the current round as follows: Sim_1 hands P_i his round ρ messages (which are copied from M'_i) and receives back the messages for round $r + 1$. Sim_1 sets \vec{out}' to be the messages he received from P_i and uses these messages to update the corresponding entries in M'_i . Subsequently, he simulates the vector $\vec{D} = [\text{Com}(x_i), \text{Com}(r_i), \text{Dec}(\rho), \text{Dec}(M_i)]$ by $\vec{D}' = [\text{Com}(0), \text{Com}(r'_i), \text{Dec}(\rho), \text{Dec}(M'_i)]$. Finally, Sim_1 sets P_{M^*} 's output to (\vec{out}', \vec{D}') .

It is straight forward to verify that for every honest P_i the simulated M'_i is distributed identically to M_i that P_i would store in the execution of $\pi^{*'}$. Indeed, the functionalities \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$ correspond to the parties' state-transition functions,

and are defined to generate the same protocol-messages as the party would generate if they would be executing the protocol π . Hence the only difference between the real and the simulated transcript is that for each honest P_i , the commitments $\text{Com}(x_i)$ and $\text{Com}(r_i)$ to P_i 's input x_i and randomness r_i are replaced by commitments $\text{Com}(0)$ and $\text{Com}(r'_i)$. The GUC-security of our commitment scheme ensures that \mathcal{Z} cannot distinguish between the two views. \square

Sub-step 4.2. As in Sub-step 3.2, we observe that, by definition, π_{inp} and π_{comp} are CP secure realizations of functionalities \mathcal{F}_{inp} and $\mathcal{F}_{\text{comp}}$, respectively. The composition theorem applied to Claim 2.2.4 implies that π^* CP-securely realizes the functionality \mathcal{F} .

This completes the proof of Lemma 2.2.3.

2.2.3 A Concrete Instance

Thus far the results have been stated for an abstract GUC-AuthComplete setup. For a concrete instance we can use the KRK setup of [11, 17]. Recall that in those works the Key Registration with Knowledge (KRK) functionality (hence forth the 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 easily verify that any protocol which is GUC secure in the original GUC-KRK hybrid world, is also GUC secure when using setup $[KRK]$.¹³ Moreover all protocols of [11, 17] using GUC-KRK are setup off-line. In [17]

¹³The 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.

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. Therefor we obtain the following corollary.

Corollary 2.2.6. *If the DDH assumption holds, there exist (efficient) setup $\bar{\mathcal{G}}$ and a programmable resource $\mathbf{M} = \{\mathbf{M}_x\}_{x \in \{0,1\}^*}$ such that for every well-formed aborting functionality \mathcal{F} , there exists a protocol π which CP realizes \mathcal{F} with GUC fallback in the $\{\bar{\mathcal{G}}, M_{\mathcal{F}}\}$ -hybrid model.*

Chapter 3

Implications for Mechanism

Design

In this chapter we translate our results into the language of game theory and interpret them in terms of reducing trust in mechanisms. We first translate the cryptographic language of the CP framework into that of computational game theory and define the model of games we work with. For this we adapt the language of [37] and we refer to it for a more detailed discussion. Next we formalize some relations between strategies and give the game theoretic interpretation of Theorem 2.2.1. Finally we describe our equivalence notion between games and use our main construction from the proof of the theorem to show how to reduce trust in the mechanism for a large class of games.

3.1 Viewing Protocols as Games

We view CP protocols as *mediated (extensive form) games of incomplete information* [29]; a very general model of games. We call an n -party (adversary oblivious¹) functionality \mathcal{M} a *mechanism*. The environment's input x_i for player P_i is its *type*, the protocol π_i run by P_i is its *strategy* and the messages sent by π_i are called *actions*. We call P_i 's view at the end of an execution its *output*. The view includes x_i , the random tape, a description of π_i and all messages received during the execution.

We call a vector of these objects with one dimension per user *profiles*. We use standard notation to construct strategy profiles as $(\pi_{-i}, \phi_i) = (\phi_1, \dots, \phi_{i-1}, \pi_i, \phi_{i+1}, \dots, \phi_n)$.

A game is also equipped with an efficiently computable *interpretation* function Ω mapping output profiles to *outcome* profiles in an arbitrary outcome space.² Moreover, for each player P_i we assume the existence of a real valued functions μ_i called the *utility function* defined over the outcome space. For a outcome y the value of $\mu_i(y) \in \mathbb{R}$ is denoted as the *payoff* of P_i .

In a mediated game Γ players are perfectly isolated from each other beyond their interfaces with the mechanism \mathcal{M} . First they are given their types, then they run their strategy with \mathcal{M} eventually obtaining an output for the execution. The game is played as a sequence of rounds r_1, r_2, \dots . In each round r_ℓ all players take an action. Once all players are done \mathcal{M} computes a vector of messages m_ℓ (which may depend on all previous actions taken in the game) and sends player P_i

¹The output of an adversary oblivious functionality is defined as not depending on the adversarial interfaces.

²Keeping in line with standard game theoretic models we restrict the output of Ω to not depend on the choice of strategy used (i.e. the value of the code for the ITM in the views).

component $m_{\ell,i}$. The notion of *rationality* predicts that each player chooses their strategies so as to maximize the expected value of their payoff.

Definition 3.1.1 (Mediated Game). *A computational extensive form mediated game of incomplete information (or just game for short) is a 4-tuple $\Gamma = (\mathcal{M}, \Omega, \mu, T)$ with mechanism \mathcal{M} , interpretation function Ω , utility profile μ , and public distribution over type profiles T . Additionally \mathcal{M}, Ω and μ are efficiently computable and T is efficiently samplable.*

3.2 CP Realizing Mechanisms

We begin by interpreting the definition of CP realization in the game theoretic setting. For this we develop a formal means for comparing and relating games to each other.

Using “small o-notation” we denote by $\text{NEGL} = \{v(\cdot) \mid \forall c \in \mathbb{N}, v \in o(\lambda^{-c})\}$ the set of negligible functions in security parameter λ . For game $\Gamma = (\mathcal{M}, \Omega, \mu, T)$ let $S(\Gamma)$ be the set strategy profiles. For $\pi \in S(\Gamma)$ let $\mu_i(\pi)$ be a random variable over \mathbb{R} describing P_i 's payoff induced by playing Γ with π using type profile $t \leftarrow T$ and fresh randomness for π and \mathcal{M} and let $\bar{\mu}_i(\pi) = \mathbb{E}(\mu_i(\pi))$ be it's the expected value. A pair of strategy profiles are equivalent if they induce the same payoff profiles. That is for games Γ and G with utility profiles μ and ν we call strategy profile $\pi \in S(\Gamma)$ and $\phi \in S(G)$ *equivalent* (write $\mu(\pi) \approx \nu(\phi)$) if:

$$\forall D \in \text{PPT} \quad \Pr[p_0 \leftarrow \mu(\pi), p_1 \leftarrow \nu(\phi), b \leftarrow \{0, 1\} : D(p_b) = 1] \in \text{NEGL}.$$

In this case we abuse notation and write $\pi \approx \phi$.³

³We observe that this is stronger then only comparing the expected values of the two distri-

We call a function over n dimensional vectors an *efficiently and locally computable* function if it can be described as the product of n independent efficiently computable functions where the i^{th} function maps the i^{th} input coordinate to the i^{th} output coordinate. We would like to say that a game Γ “contains” a game G if for every strategy profile in G there is an equivalent strategy profile in Γ . As we work in a computational setting though we make the additional requirement that the mapping be efficiently and locally computable.

Definition 3.2.1 (Containing a Game). *Let $\Gamma = (\mathcal{M}, \Omega, \mu, T)$ and $G = (\mathcal{R}, \Pi, \mu, T)$ be a pair of n -party mediated games. We say that Γ contains G if there exists an efficiently and locally computable function $f : S(G) \rightarrow S(\Gamma)$ such that:*

$$\forall \phi \in S(G) \quad \pi \approx f(\pi).$$

In this case we write $G \subseteq \Gamma$.

We are now ready to prove the primary tool which relates the CP framework to that of computational games. Intuitively, the following theorem states that if a mechanism \mathcal{R} can be used to CP realize a mechanism \mathcal{M} , then for any game Γ using \mathcal{M} there exists a game G using \mathcal{R} (with a different interpretation function but) with the same types and utilities with $G \subseteq \Gamma$.

Theorem 3.2.2. *Let \mathcal{R} and \mathcal{M} be a pair of mechanisms such that $\mathcal{M} \sqsubseteq^{CP} \mathcal{R}$.*

Then for any computational mediated game $\Gamma = (\mathcal{M}, \Omega, \mu, T)$ there exists a game $G = (\mathcal{R}, \Pi, \mu, T)$ for which it holds that $G \subseteq \Gamma$.

butions; the only property of payoffs considered in an a computational Nash equilibria. In fact any poly-time testable property of the payoff distributions is preserved in the above equivalence notion.

Proof. We construct a game G and prove the theorem. As \mathcal{R} , μ and T are fixed it only remains to define interpretation function Π and utility profile ν .

Let Υ be a function which maps \mathcal{R} -hybrid views to computationally indistinguishable \mathcal{M} -hybrid views. More precisely Υ takes as input an n -tuple of views (one per player) generated by an execution in the \mathcal{R} -hybrid world using arbitrary (efficient) n -party protocol $\phi_{[n]}$ in the \mathcal{F} -hybrid world.

1. It extracts the description of $\phi_{[n]}$, their inputs $x_{[n]}$ and all random tapes used to run $\phi_{[n]}$.
2. It computes a description of the protocol $\sigma_{[n]} = \text{Sim}_{[n]}(\phi_{[n]})$.
3. It selects fresh randomness for \mathcal{M} and runs σ on input $x_{[n]}$ with the matching extracted random tapes to simulating an execution in the \mathcal{M} -hybrid world.
4. Finally it outputs the n -tuple of views of $\phi_{[n]}$ generated by the simulation.

Then let $\Pi = \Upsilon \circ \Omega$ be the function that takes as input an n -tuple of views of a \mathcal{R} -hybrid execution and maps them to a real number by first applying Υ and then Ω .

It remains to prove that G is contained in Γ . Assume for the sake of contradiction that $G \not\subseteq \Gamma$. That is $\exists \phi \in S(G)$ such that for any efficiently and locally computable function $f : S(G) \rightarrow S(\Gamma)$ there exists an efficient distinguisher D for $\nu(\phi)$ and $\mu(f(\phi))$. We use ϕ and D to construct adversaries and an environment for which there are no simulators contradicting $\mathcal{M} \sqsubseteq^{CP} \mathcal{R}$.

Let $\mathbf{A}_{[n]}$ be the set of \mathcal{R} -hybrid world adversaries which corrupt their respective players and run ϕ on them with input (type) supplied by the environment \mathcal{Z} and at the end of the execution return their entire view to \mathcal{Z} . By definition a mechanism is an adversary oblivious functionality and so it ignores adversarial interfaces. In

other words the actions of any adversaries $\sigma_{[n]}$ in the \mathcal{M} -hybrid world depend only on their input from \mathcal{Z} , random tapes and the messages they receive on the corrupt players interface to \mathcal{M} . In particular they can be viewed as a strategy profile $\pi \in S(\Gamma)$ where the i^{th} strategy π_i behaves precisely as σ_i simply dropping any output σ_i might produce on it's adversarial interface to \mathcal{M} and letting player P_i act on behalf of the environment (in order to supply a type argument).

Consider the environment \mathcal{Z}_σ which samples $t \leftarrow T$, gives it to $A_{[n]}$ and then runs D on the n -tuple of views it receives as output. Then by assumption of D distinguishes between the outputs of $\sigma_{[n]}$ and $A_{[n]}$. But this implies that for every set of \mathcal{M} -hybrid simulators $\sigma_{[n]}$ for $A_{[n]}$ there exists a distinguishing \mathcal{Z}_σ which contradicts $\mathcal{M} \sqsubseteq^{CP} \mathcal{R}$. \square

3.3 Reducing Trust in Mechanisms

In this subsection we prove Theorem 3.3.5. For this we first formalize what it means for a pair of games to be equivalent. Then we prove a lemma stating that if two games contain each other then they are equivalent. Next we prove a lemma that, for any game with mechanism \mathcal{M} constructs an equivalent game but with the synchronization wrapper $\widehat{\mathcal{M}}$ applied to the mechanism. Finally we prove the central theorem of this section showing how to reduce the trust in a mechanism while building an equivalent game.

3.3.1 Equivalent Games

The following definition of a computational Nash Equilibrium is taken from [38]. We note that these are one of the most general types of (computational) equilibria

and so if we can preserve these between games then in particular we also preserve other more resilient (and desirable) equilibria such as k -resilient equilibria, dominant strategies and also correlated equilibria.

Definition 3.3.1 (Computational NE). *For game $\Gamma = (\mathcal{M}, \Omega, \mu, T)$ strategy profile $\pi \in S(\Gamma)$ is called a computational Nash Equilibrium (cNE) if:*

$$\forall \phi \in S(\Gamma) \quad \exists \epsilon \in \text{NEGL} \quad \forall i \in [n] : \bar{\mu}_i(\pi) \geq \bar{\mu}_i(\phi_i, \pi_{-i}) - \epsilon.$$

We denote the set of cNE of Γ with $E(\Gamma) = \{\pi \in S(\Gamma) \mid \pi \text{ is a cNE}\}$.

As before, besides equating the sets of cNE we make the additional requirement that there be an efficiently and locally computable function mapping cNE in one game to cNE in the other.

Definition 3.3.2 (Equivalent Computational Games). *We call a pair of n -party computational mediated games $\Gamma = (\mathcal{M}, \Omega, \mu, T)$ and $G = (\mathcal{R}, \Pi, \mu, T)$ equivalent if there exist a pair of efficiently and locally computable functions $f : E(\Gamma) \rightarrow E(G)$ and $g : E(G) \rightarrow E(\Gamma)$ such that:*

- $\forall \pi \in E(\Gamma) \quad \pi \approx f(\pi)$
- $\forall \phi \in E(G) \quad \phi \approx g(\phi)$.

In this case we abuse notation and write $\Gamma \approx G$.

It is easy to verify that \approx is transitive. Next we show that if a pair of games contain each other then they are equivalent.

Lemma 3.3.3. *For n -party mediated games Γ and G :*

$$\Gamma \subseteq G \quad \wedge \quad G \subseteq \Gamma \quad \implies \quad G \approx \Gamma.$$

Proof. We need to show that every cNE in Γ can be efficiently and locally mapped to cNE in G and vice versa. We show the first part and the second follows with an identical argument switching the roles of G and Γ .

Let $f : S(\Gamma) \rightarrow S(G)$ be the mapping given by $\Gamma \subseteq G$. Let equilibria $\pi \in E(\Gamma)$ with $g(\pi) = \phi \in S(G)$. Suppose for the sake of contradiction that $\phi \notin E(G)$. This implies that for some $i \in [n]$ there is a strategy ϕ'_i with an expected payoff significantly⁴ larger than that of ϕ . In symbols $\bar{v}(\phi) \leq \bar{v}(\phi'_i, \phi_{-i}) + \delta$ for some function δ growing faster than any function in NEGL. By Definition 3.2.1 f is locally computable and preserves payoff profiles it follows that there exists a strategy π'_i for P_i in Γ such that:

$$\bar{\mu}(\pi) \approx \bar{\mu}(f(\phi)) \approx \bar{\mu}(\phi) \leq \bar{\mu}(\phi'_i, \phi_{-i}) + \delta \approx \bar{\mu}(f(\phi'_i, \phi_{-i})) + \delta \approx \bar{\mu}(\pi'_i, \pi_{-i}) + \delta$$

which contradicts $\pi \in E(\Gamma)$. □

3.3.2 On Output Synchronization in a Stand-Alone Game

To formally apply the results of Theorem 2.2.1 we first need to address the issue of output synchronization. As the CP security notion is composable it is equipped with an online distinguisher (namely environment \mathcal{Z}). Thus a protocol which takes say 2 rounds of communication to produce output can at most realize a functionality which waits till all players have been activated 2 times (in an appropriate order) before it gives them output. Otherwise \mathcal{Z} can trivially distinguish by activating all player once and then checking if they already have output before it continues the execution. (See Section 2.1 for a more detailed discussion.)

⁴By “significant” we mean a function in λ larger than any $\epsilon \in \text{NEGL}$.

While we briefly address the more general composable mediated games setting later on, for now we take the common view in game theory and consider a game as being an isolated event (or at least that no players are involved in concurrent interactions). In this setting output synchronization has no effect on the game.

To see why this is we first describe the effects of the output synchronization wrapper in game theoretic terms. Let \mathcal{M} be an efficient mechanism and $\widehat{\mathcal{M}}$ be it's output synchronized version such that we are given a \mathcal{R} -hybrid protocol π which CP realizes it. Moreover let u be an upper-bound on the number of rounds of π . As we need $\widehat{\mathcal{M}}$ to be adversary oblivious we fix the value of $R = u$ in it's wrapper instead of having it query a simulator for the value. This guarantees that on the one hand simulators in the $\widehat{\mathcal{M}}$ -hybrid world obtain at least as much output synchronization as obtained from an execution of π , but on the other hand $\widehat{\mathcal{M}}$ is adversary oblivious making it a valid mechanism.

Now when viewed as mechanisms $\widehat{\mathcal{M}}$ behaves exactly as \mathcal{M} except that for any round r_ℓ the mechanism only produces the output m_ℓ once all players have made (at least) R attempts at taking action for r_ℓ . Although the first attempted action is the one that is actually used to compute m_ℓ , mechanism $\widehat{\mathcal{M}}$ does not deliver m_ℓ until all players have made R attempts.

Translating this observation to the (stand-alone) game theoretic setting we obtain the following lemma.

Lemma 3.3.4. *For any mechanism \mathcal{M} and $\widehat{\mathcal{M}}$ and computational mediated game $\Gamma = (\mathcal{M}, \Omega, \mu, T)$ there exists an interpretation function Π such that the computational mediated game $G = (\widehat{\mathcal{M}}, \Pi, \mu, T)$ is equivalent to Γ .*

Proof. We show the lemma in two parts. First we show that $G \subseteq \Gamma$ and then we show that $\Gamma \subseteq G$ which implies $\Gamma \approx G$ by Lemma 3.3.3.

Fix any positive integer R (of size polynomial in the security parameter). To show $G \subseteq \Gamma$ we first define an efficient and locally computation function $g : S(G) \rightarrow S(\Gamma)$. It consists of n efficient functions g_i that take input a strategy ϕ_i and output strategy ϕ'_i . The ITM ϕ'_i runs ϕ_i internally such that for all rounds r_ℓ once ϕ_i has made its first attempt taking action, any resulting messages $m_{\ell,i}$ from \mathcal{M} to ϕ_i are only made visible to ϕ_i once it has made (at least) $R - 1$ additional attempts taking action. The remaining $R - 1$ attempts are simply dropped. Other than that ϕ'_i behaves exactly as directed by its internal copy of ϕ_i .

We are now ready to construct interpretation function Π from Ω . Function Π takes in an n -tuple of outcomes of a run of G . Much like the function Υ described in the proof of Theorem 3.2.2 Π extracts the strategy profile ϕ , the types and the random tapes used in G to runs $g(\phi)$ with the same inputs in a new game with mechanism \mathcal{M} . Thus by construction, for every strategy profile $\phi \in S(G)$ there exists strategy profile $g(\phi) \in \Gamma$ inducing identical payoff distribution. In particular $G \subseteq \Gamma$.

Next we show that $\Gamma \subseteq G$. We define an efficient and locally computable function $f : S(\Gamma) \rightarrow S(G)$. It too consists of n efficient functions f_i that take in a strategy π_i for Γ and output a strategy π' for G . The only difference between π_i and π'_i is that for every action taken by the former the later repeats the action an additional $R - 1$ times. Then by construction of f (i.e. Π) and the fact that Ω does not depend on the particular choice of strategy used to compute the output profile we get:

$$\Omega(\pi) = \Omega(g(f(\pi))) = \Pi(f(\pi))$$

which implies that $\Gamma \subseteq G$. □

3.3.3 Implications of CP Compiler to Game Theory

We apply the game theoretic interpretation of CP security (Theorem 3.2.2) to our main feasibility result for CP security with GUC fallback (Theorem 2.2.1). This results in a transformation mapping any mediated game $\Gamma = (\mathcal{M}, \Omega, \mu, T)$ to an equivalent mediated games $G = (\mathcal{R}, \Pi, \mu, T)$ which places significantly less trust in its mechanisms. Borrowing some cryptographic concepts the change in trust intuitively be expressed as follows: while Γ relies on \mathcal{M} to act as an ideal functionality computing the output correct, the new game G only trusts \mathcal{R} to act as a network of insecure channels over which a GUC secure protocol is run.⁵ In particular players in G no longer have to trust the mechanism to maintain the privacy of their actions nor do they have to trust that \mathcal{R} computes the correct output. On the other hand they do still have to trust the mechanism to enforce isolation of colluding players as well as fairness.

Well Motivated Games. We view any game as having a special input \perp modeling the case when a player aborts the game or simply refuses to play. The construction of Theorem 2.2.1 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 Γ *well motivated* if for all $i \in [n]$ the expected utility of any outcome obtained with an abort is less than the utility of all outcomes obtained without an abort.

Next we state and prove the central theorem of this section.

Theorem 3.3.5 (Replacing Mechanisms). *Let $\bar{\mathcal{G}}$ be a functionality satisfying the*

⁵A more formal interpretation is obtained from the fallback security achieved in Theorem 2.2.1.

conditions of Theorem 2.2.1 and let \mathbf{M} be the corresponding programmable resource. Then for any well motivated computational mediated game $\Gamma = (\{\mathcal{M}, \bar{\mathcal{G}}\}, \Omega, \mu, T)$ there exists a computational mediated game $G = (\{\mathbf{M}_{\mathcal{M}}, \bar{\mathcal{G}}\}, \Pi, \mu, T)$ such that $\Gamma \approx G$.

Proof. Using protocol compiler in the proof of Theorem 2.2.1 we obtain a protocol π with number of rounds upper bounded by some fixed u such that $\{\widehat{\mathcal{M}}, \bar{\mathcal{G}}\} \sqsubseteq_{\pi}^{CP} \{\mathbf{M}_{\mathcal{M}}, \bar{\mathcal{G}}\}$.⁶ Suppose we can construct a game G equivalent to Γ but using mechanism $\mathcal{M}' = \{\widehat{\mathcal{M}}, \bar{\mathcal{G}}\}$ then the theorem follows directly by Lemma 3.3.4.

Let G be a computational mediated game with mechanism \mathcal{M}' , utility profile μ and type profile T . We construct G 's interpretation function Π from Ω as in the proof of Theorem 3.2.2 thus $G \subseteq \Gamma$. So if we can also show that $\Gamma \subseteq G$ then the theorem follows directly from Lemma 3.3.3.

To show that $\Gamma \subseteq G$ we define a function $g : S(\Gamma) \rightarrow S(G)$ and show that it satisfies Definition 3.2.1. By assumption there exists a protocol (strategy profile) ϕ such $\mathcal{M} \sqsubseteq_{\phi}^{CP} \mathcal{R}$. Let $\pi \in S(\Gamma)$. Then define $g(\pi) := \pi \circ \phi$ which is the protocol that uses ϕ in the \mathcal{R} -hybrid world to simulate an instance of \mathcal{M} which it uses in turn to run π . In particular g is efficiently and locally computable as the i^{th} output component $\pi_i \circ \phi_i$ depends only on the value of π_i .

Moreover for any $\pi \in S(\Gamma)$ we claim that $\pi \approx g(\pi)$. Suppose for a moment that this were not so. That implies that there exists of an efficient distinguisher D for $\mu(\pi)$ and $\mu(g(\pi))$. But if this were the case then an environment \mathcal{Z}_{π} can use π and D to attack ϕ . More precisely \mathcal{Z}_{π} samples $t \leftarrow T$ and runs $\pi(t)$ forwarding all messages from $\pi_i(t_i)$ for \mathcal{M} to P_i and feeding the responses back to $\pi_i(t_i)$. Let V

⁶As discussed above to make $\widehat{\mathcal{M}}$ adversary oblivious (i.e. a valid mechanism) we fix the value of $R = u$ in the synchronization wrapper (rather than have it query the adversary).

be the n -tuple of views of the emulated π 's in the execution. Then \mathcal{Z}_π produces the output bit computed as $D(\mu(\Omega(V)))$. That is it interprets the views as outputs for Γ , computes the payoff profile and gives the result to the distinguisher returning it's output. As D is a good distinguisher so must \mathcal{Z}_π be. \square

At this point a traditional game theorist will no doubt be concerned with the fact that the compiled CP protocol of Section 2.2 requires the use of a setup functionality. We address two issues the presence a setup may introduce.

On Games with Setup. As observed already in Chapter 2 an undesirable side effect of using our protocol compiler is the presence of a shared setup functionality.

However, we note that the setup is only used at the beginning of the computation and the interaction is independent of players type. Thus, to apply the compiler to *any* efficient mechanism (rather than ones with setup) one could also adopt a model similar to that of [32] where games are preceded by a pre-computation round which is run before types are distributed. Moreover, in contrast to [32] players need only be isolated just before they receive their types (but after they have finished the pre-computation phase). In this setting any signaling (or more generally correlation) that may occur due to the setup is not a problem as types are not known yet.

Another mitigating factor is that the CP compiler can use all known GUC setup functionalities. The KRK model of [11] calls for players to place their public keys into a publicly viewable file which they could potentially use to signal by skewing the distribution of their keys. However the Augmented CRS (ACRS) setup in the same paper only contains the master public key of an IBE. In particular the shared view of players contains a random sample from an *honestly sampled* distribution.

Thus the most they can gain in this setting is added correlation (aka randomness pollution). However actually communication is no longer an option. One issue here is that [11] already assumes authenticated channels so the CP compiler requires a setup in the form of PKI to realize these (as described in [18]). But using identity based signatures [24] this too can be achieved with out having players choose values in the public view.

Randomness Pollution and Setup. In order to apply our CP protocol to reduce trust in mechanisms players need to interact with a setup functionality. Formally this means a game with mechanism \mathcal{M} also needs to give players access to a setup $\bar{\mathcal{G}}$ which we denote with a mechanism of the form $\{\mathcal{M}, \bar{\mathcal{G}}\}$ and indeed, this is the language we use in the theorem bellow.

However this causes *randomness pollution* i.e. the presence of unwanted randomness in the public view of players. We point out three mitigating factors. First the interaction with $\bar{\mathcal{G}}$ can take place before types are distributed (as in [32]) after which $\bar{\mathcal{G}}$ can disappear. Moreover if the ACRS of [11] is used in combination with identity based signatures [24] then in particular, players no longer have any influence over the public view (unlike a normal PKI such as the KRK setup where users choose their public keys). Finally the setup can be reused indefinitely across many runs of many (possibly different) games. So taking an an asymptotic view of game play (such as for repeated games) the randomness pollution tends towards 0.

3.4 Concurrent Games

In this section we have so far opted for the standard model of a single game being played in a stand-alone setting. However the cryptographic results obtained above would actually allow for more general statements about games running *concurrently*.

By concurrently we mean the intuitive model where each rational player may be taking part in multiple mediated games concurrently, each with different player sets and mechanisms. While mechanisms still operate only on actions of a single game, the notions of strategies, interpretations, utilities, types distributions and payoffs are generalized to cover all games a particular player is involved in simultaneously. Thus an equilibrium spans all games being run in parallel and is described as a strategy profile with one strategy for every player taking part in at least one game.

Even in this setting a CP protocol can still be used to replace a particular mechanism. However an interesting subtlety concerns the inherent synchronization a protocol might contain. Indeed, in Theorem 2.2.1 this is addressed by adding the synchronization shell to \mathcal{F} . We note that this issue also arises with the protocols of [29, 27, 25] as the many publicly verifiable actions also provide new means for synchronization between players. In a concurrent setting sets of players could well leverage their common view of these events in one game to coordinate actions “illegally” in other games running concurrently or even to communicate illegally.

We envisage two solutions for this issue. The first is to assume that protocols are executed atomically. Thus players can not use the more fine grained synchronization obtained from a protocol execution effectively in a different game. Indeed, on an implicit level this is the approach of [27]. However it could be argued that such a solution does not really capture the intuition behind “universally compos-

able” mechanism design. Instead it’s closer to sequential composability, especially for normal form games.

In order to salvage the intuition behind the nature of strong composition we propose a second variant more along the lines of Theorem 2.2.1. In particular extra the means for synchronization must be modeled explicitly in the original mechanism \mathcal{M} . For example suppose we can CP realize \mathcal{M} in a \mathcal{R} -hybrid world with some protocol π which uses c rounds. Further, suppose \mathcal{M} is modified by applying the synchronization wrapper as described above in this appendix to obtain mechanism $\widehat{\mathcal{M}}$. In particular at the beginning of each round $\widehat{\mathcal{M}}$ asks for c actions from each player where only the first action per player is actually used to compute the next output message of the $\widehat{\mathcal{M}}$. However the output is only produced and delivered to the players once all players have made handed in their inputs c . Somewhat more precisely $\widehat{\mathcal{M}}$ also has to ensure that the activations happen in an appropriately interleaved manner reflecting the constraints of how a protocol replacing $\widehat{\mathcal{M}}$ would be executed. For example if P_i hasn’t made his attempt yet then any other is allowed at most one action counting towards it’s c' .

Appendix A

Common Elements of (G)UC Models.

In order to describe the CP framework formally first clarify the paradigms inherited from the GUC framework [11] and fix some common notation for comparing protocol executions with each other.

Ensembles and Distance. We briefly review the notions of distribution ensembles and (computational) indistinguishability. A pair of ensembles X and Y indexed by a set S is an infinite set of random variables each associated with a unique element in S .

Definition A.0.1 (@Vassilis: get it from Canetti). *We say that a pair of distribution ensembles X and Y indexed by S are (computationally) indistinguishable if for all non-uniform PPT turing machines D the following function $\Delta_{X,Y}(s)$ with domain S is negligible in s :*

$$\Delta_{X,Y}(s) := |\Pr[D(X_s) = 1] - \Pr[D(Y_s) = 1]|.$$

In this case we write $X \approx Y$ to denote the relation.

Modeling Functionalities. As in the UC framework we model functionalities as PPT ITMs. A functionality might consist of a set of functionalities $\mathcal{F} := \{\mathcal{F}_1, \dots, \mathcal{F}_m\}$ available to the parties in parallel, where it is assumed that there exists some addressing mechanism for the players to interact with the component functionalities $\mathcal{F}_1, \dots, \mathcal{F}_m$.

A primary feature of the CP model are the split adversaries. Thus the main *difference* between the CP and UC functionalities lies in the interface(s) made available to the adversaries. In addition to its interfaces to the (honest) parties an n -party resource \mathcal{F} has a dedicated interfaces to *each* of the n adversaries A_1, \dots, A_n . For any message which is sent/received on an adversarial interface, the specification of \mathcal{F} defines the adversary-ID (in particular the matching interface) of the adversary that should send/receive the message (just as is already the case for messages sent/received by honest parties).

Simulation-based Security. Simulation-based security definitions follow the real-world/ideal-world paradigm: In the real world, the players execute the protocol and can communicate over channels as defined by the model. In the ideal world, the players securely access an ideal functionality \mathcal{F} that obtains inputs from the players, runs the program that specifies the task to be achieved by the protocol, and returns the resulting outputs to the players. Intuitively, a protocol *securely realizes* the functionality \mathcal{F} if, for any real-world adversary A attacking the protocol execution, there is an ideal-world adversary Sim , also called the *simulator*, that emulates A 's attack in the ideal evaluation of \mathcal{F} . The “quality” of the simulation is specified by considering a distinguisher \mathcal{Z} , called the *environment*, which interacts,

in a well defined manner, with the parties and the adversary or the simulator and tries to distinguish between the two worlds.

The advantage of simulation-based security is that it satisfies strong composability properties: let π_1 be a protocol that securely realizes a functionality \mathcal{F}_1 . If a protocol π_2 , using the functionality \mathcal{F}_1 as a subroutine, securely realizes a functionality \mathcal{F}_2 , then the protocol $\pi_2^{\pi_1/\mathcal{F}_1}$, which results when replacing the calls to \mathcal{F}_1 by invocations of π_1 , securely realizes \mathcal{F}_2 (without calls to \mathcal{R}_1). therefore, it is sufficient to analyze the security of the simpler protocol π_2 in the \mathcal{F}_1 -*hybrid* model, where the players run π_2 with access to the ideal functionality \mathcal{F}_1 . For more details on composability of protocols and a formal handling of composition, the reader is referred to, e.g., [7, 19, 9, 13, 5, 36].

UC with Global Setup (GUC). In defining the model of execution we use as a starting point the UC with global setup (GUC) framework of Canetti et al. [11]. The main difference between UC and GUC is that in the GUC model any global setup is present in *both* the real and the ideal worlds, and is accessible by the environment. Thus, in the GUC framework, a simulator must work with the setup it is provided; this is in contrast to the UC framework where the simulator gets to *simulate* the setup (which provides the simulator with extra power). For the purpose of collusion preservation, this latter approach will not work since we will, in general, have many simulators running in parallel, with no mechanism for coordinating the setup among themselves.¹ It is for this reason that we use GUC

¹For concreteness consider a 2-party protocol which realizes \mathcal{F} : a secure unidirectional transfer of a single bit using a CRS as setup. Suppose both parties are corrupt with the strategy that they obtain the CRS, output it and terminate. Of course in the real world their outputs will be randomly distributed and identical. However, in an ideal world with no CRS functionality but only \mathcal{F} made available to the two simulators they have no way to feed random but identical values for the CRS to their emulations.

as our starting point.

Execution Notation. In this work we often compare the outcome of executions of different arrangements of ITMs with each other. To this end we extend the notation of [9, 11] which we now briefly review as it applies to the case of the GUC framework. For more details we refer the reader to the original works. We denote by $\bar{\mathcal{G}}$ the ACRS setup functionality of [11]. For an \mathcal{R} -hybrid protocol π , setup $\bar{\mathcal{G}}$, adversary \mathbf{A} , and environment \mathcal{Z} , we denote the output of \mathcal{Z} after witnessing an execution of π in the GUC \mathcal{R} -hybrid model in the presence of \mathbf{A} as the ensemble $\text{EXEC}_{\pi, \mathbf{A}, \mathcal{Z}}^{\bar{\mathcal{G}}, \mathcal{R}}$. To be precise, by this we denote the ensemble of random variables, parameterized by the security parameter k and the auxiliary input z of the environment (for details the reader is referred to [11]). For notational simplicity, similarly to [9, 11], if for a particular player P_i , no ITM is given in the subscript then it is implicitly meant that P_i acts as the dummy player simply forwarding all messages faithfully between \mathcal{Z} and \mathcal{R} .

Definition A.0.2 (UC Computation with Global Setup). *Let $\bar{\mathcal{G}}$ be a global setup, \mathcal{R} be a resource. For an n -party efficient protocol π and functionality \mathcal{F} we say that π GUC-realizes \mathcal{F} , if $\forall \mathbf{A} \exists \text{Sim} \forall \mathcal{Z}$:*

$$\text{EXEC}_{\pi, \mathbf{A}, \mathcal{Z}}^{\bar{\mathcal{G}}, \mathcal{R}} \approx \text{EXEC}_{\text{Sim}, \mathcal{Z}}^{\bar{\mathcal{G}}, \mathcal{F}}$$

Appendix B

Relations to Abstract Cryptography.

We briefly show how to define the CP framework of this thesis using the language of the Abstract Cryptography (AC) framework of [36]. In particular we provide concrete instantiations of many of the abstract mathematical concepts introduced in AC such that by plugging in these definitions we obtain the CP framework as defined above. For this subsection only, we assume the reader is familiar with the definitions and results of [36].

We begin at the highest level of abstraction and briefly provide a concrete instance of many of the most important notions defined by the AC framework. We view CP as living at Level 3, the most concrete level of AC. In the following discussion, reflecting the absence of adversaries in AC, we use the term *party* to denote the union of the set of players and adversaries in CP.

A Concrete Isomorphism. CP uses the *pseudo-metric* of computational indistinguishability defined over sets of random variables which is, in particular,

an equivalence relation. Moreover CP implicitly equips random variables with a *parallel composition* operator: composed variables are simply viewed as a single variable. In the words of [36] this composition operation (denoted by “||”) is *non-expanding*. Taking this view in turn induces the CP notion of an *isomorphism*. Intuitively, in CP we consider two executions (captured by the collection of random variables they induce as the output of the environment \mathcal{Z}) to be equivalent (i.e. isomorphic) if they are computationally indistinguishable. More precisely, the CP framework implicitly realizes the definition of isomorphisms using functions f and g that are defined via 2 sets of ITMs (including all ITMs of parties and available ideal functionalities) participating in an execution. The functions map sets of $2n$ -inputs (one per party) to random variables describing the output of \mathcal{Z} . The relation ρ is essentially the identity relation on the possible inputs of players and the pseudo-metric ‘ \simeq ’ is computational indistinguishability. One caveat is that CP implicitly only requires isomorphism to hold for input tuples which can actually be sampled by \mathcal{Z} where as AC requires isomorphism to hold for all possible inputs.

Components and Constructors. AC introduces the notion of a *component set* which in the case of CP consists of the set of ideal functionalities (i.e. PPT ITMs with one interface per party). As required by AC, they can be composed in parallel by making several such functionalities available to players in a hybrid world setting. As a concrete *constructor set* CP uses ITMs run by parties. Viewed this way, serial composition consists of running a “protocol stack” (as is done in several proofs in this work) while parallel composition corresponds to simply grouping the interfaces of several ITMs together into a single ITM.¹ Finally the

¹For example a in a hybrid world with two resources \mathcal{F} and \mathcal{R} the dummy protocol $\mathcal{D}^{\mathcal{F},\mathcal{R}}$ can be seen as the parallel composition of the protocols $\mathcal{D}^{\mathcal{F}}$ and $\mathcal{D}^{\mathcal{R}}$.

neutral element of the constructor set is the dummy protocol (also called dummy adversary depending on the party).

Choice Sets and Abstractions. In the CP framework an $2n$ -choice setting is function defined implicitly by the specification of all ITMs taking part in an execution. It maps $2n$ -inputs (one per party) to the output random variable of \mathcal{Z} . By extension the CP framework is primarily concerned with $2n$ -specifications defined by sets of $2n$ -choice settings that share the same ideal functionalities and protocols (but different adversaries). A statement about CP-emulation is really a statement relating two such sets. Moreover the primary instance of i^{th} guaranteed choice domains in a CP “real” world is the set of “ideal” inputs available to P_i while the i^{th} guaranteed choice domain in of interest in “ideal” CP worlds are the set of real world inputs available to adversary A_i .

Viewed in this light a π -abstraction in CP is given by the specifications of all ITMs taking part in a pair of executions whenever one CP-emulates the other. Intuitively it tells us how to obtain the same behavior from \mathcal{Z} using a different (presumably more realistic) arrangement of ITMs. Specifically, by viewing parties as dummy entities which forward their input to the resource a CP abstraction describes how to map an input x_i for (ideal world) player P_i to the “input” $\pi_i(x_i)$. Moreover for every input \bar{x}_i accepted by an ideal world adversary $\text{Sim}_i(A_i)$ a CP π -abstraction tells us that in the real world with adversary A_i the same input can be used. Moreover the facts that:

1. The code of the i^{th} protocol depends only the interfaces available to player P_i .
2. The transformations describing the i^{th} simulator depends only on the code

of the i^{th} adversary.

imply that the π -abstraction of CP induce a *complete factorisable relation* between choice sets satisfying Def.11 of [36]. We note that by taking this (admittedly rather informal) view of the CP framework Theorem 1 of [36] provides an alternative proof of the CP composition theorem.

Simplified CP Security. We note that compared to CP, SCP is even closer in spirit to the Abstract Cryptography framework. In particular AC does not distinguish between honest players and adversaries. In contrast to the flexibility (and variability) available to adversaries in the CP model, SCP essentially fixes the ITMs of all adversaries in both the ideal and real worlds bringing it one step closer to the spirit of AC.

Bibliography

- [1] I. Abraham, D. Dolev, R. Gonen, and J. Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. pages 53–62, 2006.
- [2] I. Abraham, D. Dolev, and J. Halpern. Lower bounds on implementing robust and resilient mediators. pages 302–319, 2008.
- [3] J. Alwen, J. Katz, Y. Lindell, G. Persiano, A. Shelat, and I. Visconti. Collusion-free multiparty computation in the mediated model. pages 524–540, 2009.
- [4] J. Alwen, A. Shelat, and I. Visconti. Collusion-free protocols in the mediated model. pages 497–514, 2008.
- [5] M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (rsim) framework for asynchronous systems. *Inf. Comput.*, 205(12):1685–1720, 2007.
- [6] I. Barany. Fair distribution protocols, or how the players replace fortune. *Mathematics of Operations Research*, 17:327–340, 1992.
- [7] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

- [8] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. pages 136–145, 2001. Full version at <http://eprint.iacr.org/2000/067/>.
- [9] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001. For a more recent version see <http://eprint.iacr.org/2000/067>.
- [10] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. pages 61–85, 2007.
- [11] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In S. P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.
- [12] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.
- [13] R. Canetti and T. Rabin. Universal composition with joint state. In D. Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2003.
- [14] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11(1), September 1971.
- [15] P. Cramton and J. Schwartz. Collusive bidding in the fcc spectrum auctions. Technical Report 02collude, University of Maryland, Department of Economics, Dec. 2002. available at <http://ideas.repec.org/p/pcc/pccumd/02collude.html>.

- [16] V. Crawford and J. Sobel. Strategic information transmission. *Econometrica*, 50:1431–1451, 1982.
- [17] Y. Dodis, J. Katz, A. Smith, and S. Walfish. Composability and on-line deniability of authentication. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography, TCC '09*, pages 146–162, Berlin, Heidelberg, 2009. Springer-Verlag.
- [18] Y. Dodis, J. Katz, A. Smith, and S. Walfish. Composability and on-line deniability of authentication. pages 146–162, 2009.
- [19] Y. Dodis and S. Micali. Parallel reducibility for information-theoretically secure computation. In M. Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 74–92. Springer, 2000.
- [20] F. Forges. Universal mechanisms. *Econometrica*, 58:1342–1364, 1990.
- [21] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications*. Cambridge University Press, Cambridge, UK, 2004.
- [22] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. pages 218–229, 1987.
- [23] T. Groves. Incentives in teams. *Econometrica*, 41(4):617–31, July 1973.
- [24] F. Hess. Efficient identity based signature schemes based on pairings. pages 310–324, 2003.

- [25] S. Izmalkov, M. Lepinski, and S. Micali. Perfect implementation. *Games and Economic Behavior (to appear)*. Available at <http://hdl.handle.net/1721.1/50634>.
- [26] S. Izmalkov, M. Lepinski, and S. Micali. 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.
- [27] S. Izmalkov, M. Lepinski, and S. Micali. Verifiably secure devices. pages 273–301, 2008.
- [28] 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.
- [29] S. Izmalkov, S. Micali, and M. Lepinski. Rational secure computation and ideal mechanism design. pages 585–595, 2005. Full version available at <http://dspace.mit.edu/handle/1721.1/38208>.
- [30] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous computation. Cryptology ePrint Archive, Report 2011/310, 2011. <http://eprint.iacr.org/>.
- [31] E. Kushilevitz, Y. Lindell, and T. Rabin. Information-theoretically secure protocols and security under composition. In *STOC 2006*, pages 109–118, 2006.

- [32] M. Lepinski, 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.
- [33] M. Lepinski, S. Micali, C. Peikert, and A. Shelat. Completely fair SFE and coalition-safe cheap talk. pages 1–10, 2004.
- [34] M. Lepinski, S. Micali, and A. Shelat. Collusion-free protocols. pages 543–552, 2005.
- [35] M. Lepinski, S. Micali, and A. Shelat. Fair zero knowledge. pages 245–263, 2005.
- [36] U. Maurer and R. Renner. Abstract cryptography. In *Innovations in Computer Science*. Tsinghua University Press, 2011.
- [37] E. J. B. Nielsen, C. J. Alwen, C. Cachin, O. Pereira, A.-R. Sadeghi, B. Schoenmakers, A. Shelat, and I. Visconti. Summary report on rational cryptographic protocols. ECRYPT Deliverable D.PROVI.7, March 2007.
- [38] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.
- [39] G. J. Simmons. The prisoners’ problem and the subliminal channel. pages 51–67, 1984.
- [40] G. J. Simmons. Cryptanalysis and protocol failures. *Commun. ACM*, 37(11):56–65, 1994.
- [41] G. J. Simmons. The history of subliminal channels. In *Information Hiding Workshop*, volume 1174, pages 237–256, 1996.

- [42] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders.
Journal of Finance, 16:8–37, 1961.