Cryptographic Complexity of Multi-party Computation Problems: Classifications and Separations

Manoj Prabhakaran* and Mike Rosulek*

Department of Computer Science
University of Illinois, Urbana-Champaign
{mmp,rosulek}@uiuc.edu

Abstract. We develop new tools to study the relative complexities of secure multi-party computation tasks in the Universal Composition framework. When one task can be securely realized using another task as a black-box, we interpret this as a qualitative, complexity-theoretic reduction between the two tasks. Virtually all previous characterizations of MPC functionalities, in the UC model or otherwise, focus exclusively on secure function evaluation. In comparison, the tools we develop do not rely on any special internal structure of the functionality, thus applying to functionalities with arbitrary behavior. Our tools additionally apply uniformly to both the PPT and unbounded computation models.

Our first main tool is an exact characterization of realizability in the UC framework with respect to a large class of communication channel functionalities. Using this characterization, we can rederive all previously-known impossibility results as immediate and simple corollaries. We also complete the combinatorial characterization of 2-party secure function evaluation initiated by [10] and partially extend the combinatorial conditions to the multi-party setting. Our second main tool allows us to translate complexity separations in simpler MPC settings (such as the honest-but-curious corruption model) to the standard (malicious) setting. Using this tool, we demonstrate the existence of functionalities which are neither realizable nor complete, in the unbounded computation model.

1 Introduction

In this work, we seek to investigate the intrinsic "cryptographic complexity" of secure multiparty computation (MPC) functionalities. MPC functionalities can have a rich structure, being interactive, often randomized, computations involving more than one party. Clearly not all functionalities have equal cryptographic sophistication. For instance, one expects a task like oblivious transfer to be much more sophisticated than the mere task of communication or local computation. One could ask if the two-party task of commitment is any more complex than

* Partially supported by NSF grant CNS 07-47027.

the task of two (mutually distrusting) parties generating unbiased coin-flips. We present a complexity-theoretic approach to asking and answering such questions.

At the heart of such an approach is identifying meaningful (or useful) notions of *reductions* between MPC functionalities, that would allow us to form "complexity classes" of functionalities with similar cryptographic complexity. The most natural notion of reduction for MPC functionalities is in terms of "secure realizability:" can one functionality $\mathcal{F}$ be securely realized given access to another functionality $\mathcal{G}$? Indeed, this notion of reduction has been extensively used in literature. Yet, the way this "reduction" was traditionally defined, it was *not transitive*. This severely restricted its usefulness as a reduction for studying cryptographic complexity. In the recently developed framework of Universal Composition (UC) [7], however, the *Universal Composition theorem* guarantees that the reduction based on secure realizability in that framework is indeed a transitive relation. It is in this framework that we ground our study.

Our results presented below can be viewed as relating to an abstract notion of complexity of MPC functionalities. More concretely, these can be interpreted as results on secure realizability in the UC framework.

Our Results. We introduce new techniques and tools to better understand and classify cryptographic complexity classes (as defined using secure realizability in the UC framework). We focus on tools that apply broadly to *arbitrary* functionalities, whereas most previous work either focused on secure function evaluation or involved *ad hoc* arguments specific to particular functionalities. Further, the main tools we develop apply in the standard UC model, as well as in the information theoretic (or computationally unbounded) variant.

We then apply our new tools to give more concrete results for specific functionalities and characterizations for important subclasses of functionalities. Our main results mostly involve showing *separations* in complexity among functionalities, as opposed to new protocol constructions. Our main results fall into two categories based on the techniques used:

Classifying Functionalities Using Splittability. We define a very general aspect of cryptographic complexity called *splittability*. We show that splittable functionalities are exactly the ones that have secure protocols in the plain model, with respect to static corruptions, using an idealized communication channel (Theorem 1). This is the first alternate characterization of realizability in the UC model.

Superficially, the definition of splittability is similar to the definition of realizability in the UC framework, and indeed, showing that a functionality is splittable is not much easier than directly showing that it is realizable. However, the main utility of the splittability characterization is that *it is often extremely easily to show that a functionality is unsplittable*. We rederive the impossibility of zero-knowledge proofs [7], bit commitment, coin-tossing, and oblivious transfer [9] as simple and easy consequences of this characterization. We also use splittability to complete the combinatorial characterization of 2-party secure function evaluation (SFE) initiated in [10, 11] (Theorem 5).

We generalize the notion of splittability as a transitive *binary relation* on functionalities, which we view as a complexity-theoretic reduction. Using this definition, we identify a class that includes all natural communication channels, and which we argue defines a natural class of "low cryptographic complexity." Then we show that for all $\mathcal{G}$ in this class, our exact characterization generalizes; that is, $\mathcal{F}$ is splittable with respect to $\mathcal{G}$ if and only if $\mathcal{F}$ has a secure protocol on the channel $\mathcal{G}$ (Theorem 3), with respect to static corruptions.

Furthemore, if a functionality is unsplittable according to the simpler, less general definition, then it has no secure protocol on any natural channel. Thus, splittability provides a powerful and easy way to separate the cryptographic complexities of many functionalities.

Our main technical results hold for multi-party functionalities, although the definitions become complicated and less intuitive for more than 2 parties. However, we show that the 2-party case yields some necessary conditions for multi-party functionalities (Theorem 7). We leave open the question of whether they are sufficient in general.

Passive Corruption and Deviation-Revealing Functionalities. A functionality's realizability depends crucially on the model of the adversary's corruption. For instance, in the unbounded computation model, functionalities like coin-flipping and commitment become trivial if the adversary is passive (honest-but-curious), while oblivious transfer still remains unrealizable. This motivates using alternate (and possibly unrealistic) corruption models to study the complexity of functionalities. We develop an effective technique to "lift" realizability separations from restricted corruption settings to the standard malicious corruption setting. While the techniques of splittability can give separations involving only relatively low-complexity functionalities, this second technique can yield separations among higher-complexity functionalities.

Translating separations in the restricted corruption settings to the standard setting is possible only for certain "well-behaved" functionalities. We identify such a well-behavedness property called *deviation revealing* and formulate an appropriate translation recipe (Theorem 4). As in the case of splittability, the deviation-revealing property is applicable to completely arbitrary functionalities.

Combining this recipe with known separations in various corruption models (as well as some easy observations), we show a sequence of four *natural* functionalities that form a hierarchy of *strictly increasing* complexity, in the unbounded computation model (Theorem 8). This implies that the two intermediate functionalities in this sequence are neither complete nor realizable (using any natural communication channel), and that there is more than one distinct intermediate level of complexity. Our result separating these two functionalities of intermediate complexity is perhaps unique since most previous works focused on only the extremes of complexity.

Related Work. Multiparty computation was introduced in the eighties, and secure protocols were given early on for realizing all functionalities [38, 18, 13, 24, 5]. However, the notion of security used was stand-alone security. MPC was also studied in an information theoretic setting, and with weaker models of adversarial behavior: honest-but-curious adversaries [16, 28, 27] and honest majority [13, 5]. In these models, much work has focused on developing alternate characterizations for the extremes of complexity: realizability [15, 16, 28, 29] and completeness [25, 30, 26, 27, 20, 4]; see the full version [35] for a more comprehensive survey of these results.

Canetti [7, 6] (and independently Pfitzmann and Waidner [33]) introduced the general framework of network-aware security used in this work, known as UC security. The first impossibility results in the UC framework were already given in [7], followed by more in [9, 10, 23]. Our splittability characterization is motivated by common techniques underlying these results. A somewhat similar technique appears in a different context in an impossibility proof by Dwork et al. [17]. Network-aware secure protocols for all functionalities were given under different variations of the model [12, 1, 8, 36, 3, 22]. Impossibility results were also shown for various settings with less standard notions of composability [31, 32, 2]. We remark that our results are meaningful in variations of the model which simply involve augmenting the UC model with a "set-up" functionality. However, some of our theory does not apply to the models in [36, 3], which effectively allow different computational powers for adversaries/environments and simulators.

2 Preliminaries

Some of our conventions differ from the original UC model. We now give an overview of the model while highlighting these (cosmetic) differences, which are motivated by our "complexity theoretic" view of MPC.

Modeling conventions. The network-aware security framework for MPC includes four kinds of entities: an *environment*, multiple *parties*, an *adversary*, and a *functionality*. The functionality's program fully specifies an MPC problem, and as such, is the primary object we classify in this paper.

Emphasizing the generality of our theory, we do not specify any computational limitations on these network entities, but instead consider abstract classes of *admissible machines*. We only require that a machine that internally simulates several other admissible machines is itself admissible.[1] Our general results apply uniformly for any such system, the two most natural of which are *computationally unbounded systems* (which admit all probabilistic machines) and *PPT systems* (which admit all probabilistic, polynomial-time machines).

Unlike the original UC model, we model the communication among the environment, parties, and functionalities as an ideal, private, tamper-proof channel. In the UC model, an adversary can tamper with and delay such communications. Instead, we assume that functionalities themselves achieve the same effect by directly interacting with the adversary each time a party communicates with

[1] As such, our theory is *not* directly applicable to the network-aware security model introduced in [36, 34] and also used in [3], where an adversary can sometimes access extra computational power that an environment cannot.

the functionality. This difference is significant in defining *non-trivial protocols*, which we address later in this section. Furthermore, there is no built-in communication mechanism among the parties; all communication must be facilitated by a functionality. In this way, we are able to uniformly consider arbitrary channels.

We require that a protocol interact only with a *single instance* of some functionality. This is without loss of generality, since we can always consider a single "augmented" functionality that provides an interface to multiple independent sessions of simpler functionalities. This convention maintains the strict binary nature of our complexity reduction. Also, for simplicity, we assume that parties and communication ports of the functionality are numbered, and that a protocol which uses $\mathcal{F}$ must have the ith party interact only as the ith party to $\mathcal{F}$. Again, this is without loss of generality, as an "augmented" functionality could provide an interface to multiple different "port-mappings" of a simpler functionality. To emphasize a qualitative measure of cryptographic complexity, we generally (implicitly) consider reductions among such augmented functionalities. In the UC model, $\mathcal{F}$ and its augmented version $\mathcal{F}^+$ can be realized in terms of one another (though not following our notational conventions). Thus all of our results may be interpreted as being in terms of augmented or unaugmented functionalities, whichever is appropriate.

Notation. $\textsc{exec}[\mathcal{Z}, \mathcal{A}, \pi^{\mathcal{G}}]$ denotes the probability of the environment $\mathcal{Z}$ outputting 1 when it interacts with parties running the protocol $\pi^{\mathcal{G}}$ (i.e., π using $\mathcal{G}$ as the sole medium for interaction), in the presence of an adversary $\mathcal{A}$. We denote the "dummy protocol" used to access a functionality by ∂ (i.e., an ideal-world direct interaction with $\mathcal{F}$ will be denoted as running the protocol $\partial^{\mathcal{F}}$). We say π is a *secure realization* of $\mathcal{F}$ with respect to $\mathcal{G}$ if if for all adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$, $\textsc{exec}[\mathcal{Z}, \mathcal{A}, \pi^{\mathcal{G}}]$ and $\textsc{exec}[\mathcal{Z}, \mathcal{S}, \partial^{\mathcal{F}}]$ are negligibly close. When there is a secure realization of $\mathcal{F}$ with respect to $\mathcal{G}$, we write $\mathcal{F} \sqsubseteq \mathcal{G}$. We define the natural complexity class $\textsc{realiz}^{\mathcal{G}} = \{\mathcal{F} \mid \mathcal{F} \sqsubseteq \mathcal{G}\}$, the class of functionalities that can be securely realized using $\mathcal{G}$. Our main results apply to both PPT and unbounded systems in a unified way, so our notation does not distinguish between them. To explicitly refer to PPT or unbounded systems, we write $\sqsubseteq_p, \textsc{realiz}_p$ and $\sqsubseteq_u, \textsc{realiz}_u$, respectively.

Non-trivial protocols. In the standard UC model where an adversary can delay communications between functionality and parties, a protocol which does nothing is trivially a secure realization (since the same effect can be achieved in the ideal world by an adversary who indefinitely blocks all outputs). Thus it is necessary to restrict attention to *non-trivial* protocols [12, 10], which are secure even when the ideal-world adversary eventually delivers all messages.

In our model, all communication between parties and functionality is on an idealized channel that does not allow blocking, but we may consider functionalities that explicitly interact with the adversary, allowing it to block or delay outputs to honest parties. For such functionalities, we must also consider a definition of non-triviality for our results to be meaningful.

Definition 1. *Let* $\mathsf{wrap}(\mathcal{F})$ *be the functionality that runs* $\mathcal{F}$*, except that outputs generated by* $\mathcal{F}$ *are kept in an internal queue.* $\mathsf{wrap}(\mathcal{F})$ *informs the adversary of each such output (source, destination, and length) and delivers it only if/when the adversary instructs it to.*

Definition 2. *Let* π *be a secure realization of* $\mathsf{wrap}(\mathcal{F})$ *with respect to* $\mathsf{wrap}(\mathcal{G})$*. We say* π *is* non-trivial*, and write* $\mathsf{wrap}(\mathcal{F}) \sqsubseteq^{\mathsf{nt}} \mathsf{wrap}(\mathcal{G})$*, if* π *is also a realization of* $\mathcal{F}$ *with respect to* $\mathcal{G}$*.*

In other words, a secure realization is *non-trivial* if, in the optimistic case where the adversary delivers all messages on $\mathsf{wrap}(\mathcal{G})$, the protocol realizes $\mathcal{F}$ (which may guarantee delivery of outputs, for example).[2] Indeed, it is often the case that one would consider $\mathsf{wrap}(\mathcal{F}_{\mathsf{pvt}})$ (or similar) as one's communication channel, and would be willing to settle for the security of $\mathsf{wrap}(\mathcal{F})$.

The important implication of Definition 2 is that $\mathcal{F} \not\sqsubseteq \mathcal{G}$ implies $\mathsf{wrap}(\mathcal{F}) \not\sqsubseteq^{\mathsf{nt}} \mathsf{wrap}(\mathcal{G})$. Thus the complexity separations we obtain (between more simply defined functionalities) also imply corresponding separations for the weaker, more realistic wrapped functionalities, with respect to non-trivial protocols.

3 Structural Results

In this section we present our two new tools for studying the realizability of functionalities. These tools apply to arbitrary functionalities and to both PPT and unbounded computational systems. We call this set of results our "structural results" to emphasize their generality. Later, in Section 4, we apply these structural results to specific settings and classes of functionalities to obtain concrete results.

3.1 Splittability of (Regular) 2-Party Functionalities

The main tool we develop to characterize classes $\text{REALIZ}^{\mathcal{G}}$ is a theory of *splittability.* For expositional clarity, we first present a special case of our splittability theory which captures the essential intuition and still has useful consequences in the more general setting. Then we remove these restrictions and present the general theory for 2-party functionalities. The general theory for the multi-party setting is more complicated and is defered to the full version.

In this section we restrict attention to a class of functionalities called 2REGULAR: the 2-party functionalities which do not directly interact with the adversary when no parties are corrupted, and whose behavior does not depend on which parties are corrupted. This class already includes all secure function evaluation (SFE) functionalities as they are typically defined. We also restrict attention to

[2] This definition is slightly stronger than the non-triviality condition in [12, 10]. Their definition was arguably sufficient for secure function evaluation, but must be strengthened to be appropriate for more general functionalities. See the full version for more detailed justification.

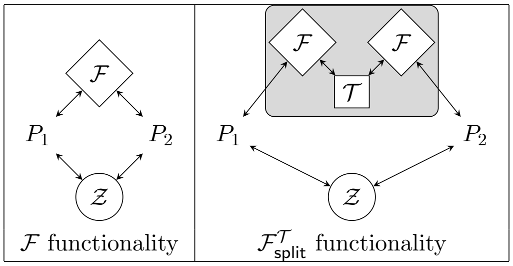

Fig. 1. 2-party splittability (for $\mathcal{F} \in$ 2REGULAR). The shaded box shows $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$.

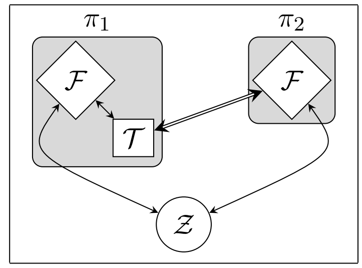

Fig. 2. Secure protocol π for a splittable functionality $\mathcal{F}$.

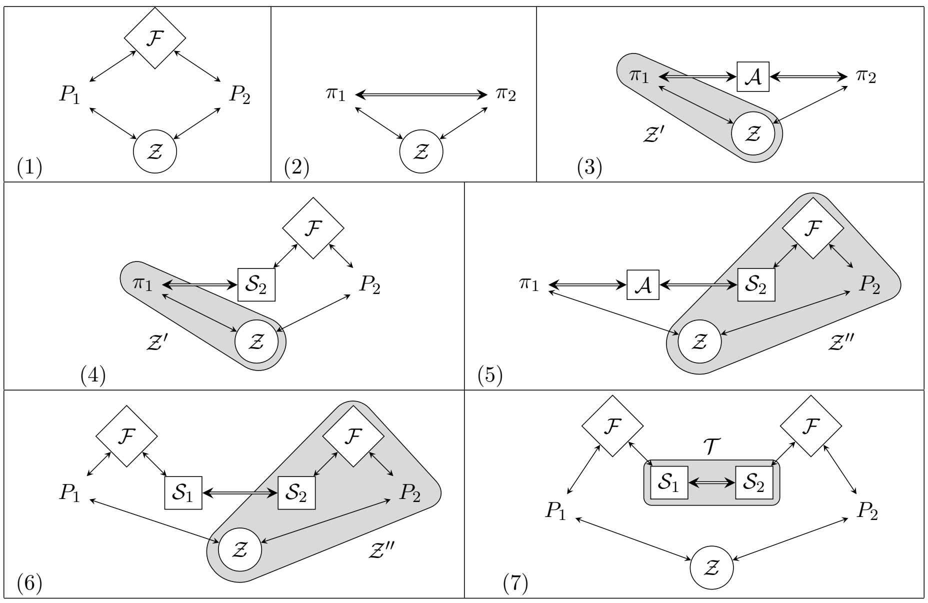

Fig. 3. Steps in the proof of Theorem 1. Given a protocol securely realizing $\mathcal{F}$, we apply the security guarantee three times: first with no parties corrupted (between boxes 1 and 2), then with a corrupt party P_1 which plays a man-in-the-middle between P_2 and an honest P_1 inside the environment (between boxes 3 and 4), and finally with a corrupt party P_2 which plays a man-in-the-middle between P_1 and the simulator from the previous step (between boxes 5 and 6), all with appropriately defined environments. The machine $\mathcal{T}$ required by the definition of splittability is derived from the simulators for the last two cases, by letting them simulate the protocol to each other.

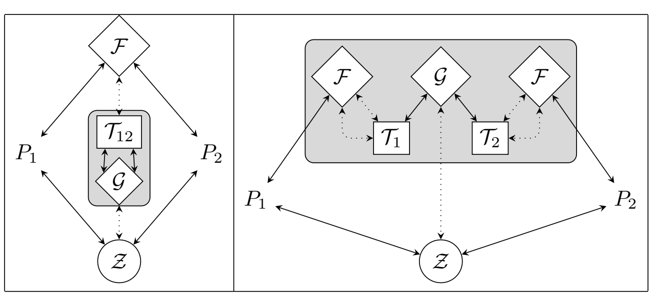

Fig. 4. General 2-party splittability (Definition 7). The shaded box in the far left shows $\mathcal{S}^{\mathcal{G},\mathcal{T}_{12}}$, and the other shaded box shows $\mathcal{F}^{\mathcal{G},\mathcal{T}_1,\mathcal{T}_2}_{\mathsf{split}}$. Dotted lines indicate interaction as adversary or corrupted party.

secure protocols that use an ideal communication channel. We let $\mathcal{F}_{\mathsf{pvt}}$ denote the *completely private channel* functionality, which allows parties to privately send messages to other parties of their choice, and does not interact with the adversary at all (not even to notify that a message was sent).

Definition 3. *Let $\mathcal{F}$ be a 2-party functionality and $\mathcal{T}$ an admissible machine. Define $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ as the compound functionality that does the following (See Fig. 1):*

$\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ internally simulates an instance of $\mathcal{T}$ and two independent instances of $\mathcal{F}$, which we call $\mathcal{F}_L$ and $\mathcal{F}_R$. The first party of $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ directly interacts with $\mathcal{F}_L$ as its first party. The second party of $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ directly interacts with $\mathcal{F}_R$ as its second party. The machine $\mathcal{T}$ interacts only with $\mathcal{F}_L$ and $\mathcal{F}_R$, as the other parties to these functionalities.

Definition 4 (Splittability). $\mathcal{F} \in$ 2REGULAR *is* splittable *if there exists a machine $\mathcal{T}$ such that $\mathcal{F}$ is indistinguishable from $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$. That is, for all environments $\mathcal{Z}$ and the dummy adversary $\mathcal{A}$ that corrupts no one, we have* $\textsc{exec}[\mathcal{Z}, \mathcal{A}, \partial^{\mathcal{F}}] \approx \textsc{exec}[\mathcal{Z}, \mathcal{A}, \partial^{\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}}]$*, where ∂ denotes the dummy protocol. We define* 2REGSPLIT *as the class of all splittable functionalities in* 2REGULAR*.*

At a very high level, $\mathcal{F}$ is *splittable* if there is a way to successfully mount an undetectable man-in-the-middle "attack" in the ideal world, between two independent instances of the functionality.

A crucial property of this definition is that it is often relatively easy to show that a functionality is unsplittable. See Section 4.1 for several examples involving some common functionalities.

Theorem 1. $\mathcal{F} \in$ 2REGULAR *is securely realizable using $\mathcal{F}_{\mathsf{pvt}}$ if and only if $\mathcal{F}$ is splittable. That is,* $\textsc{2regular} \cap \textsc{realiz}^{\mathcal{F}_{\mathsf{pvt}}} = \textsc{2regsplit}$.

Proof (Sketch). The easier direction is to see that $\textsc{2regsplit} \subseteq \textsc{realiz}^{\mathcal{F}_{\mathsf{pvt}}} \cap$ 2REGULAR. If $\mathcal{F}$ is splittable, then a protocol (π_1, π_2) can be derived as shown in Fig. 2. Note that the protocol uses a perfectly private channel for communication, which in our model is essentially the same kind of channel that the network entities use to communicate. Interestingly, the protocol at each end simulates a copy of the ideal functionality. Then the protocol's simulator can faithfully simulate the honest party's protocol merely by accessing the functionality in the ideal world.

The more interesting direction is showing that every realizable functionality is splittable. It generalizes the "split-adversary" technique used by Canetti, Kushilevitz, and Lindell [10], and also has parallels with an impossibility proof by Dwork et al. [17].[3] A visual overview is given in Fig. 3.

[3] The common thread in these proofs is to construct two separate corruption scenarios and consider a man-in-the-middle attack which pits the honest players in the two scenarios against each other.

3.2 General Theory of Splittability

Protocols in network-aware frameworks are usually considered to use less idealized channels than $\mathcal{F}_{\mathsf{pvt}}$. For instance, even the standard model of a private channel reveals to the adversary the fact that a message was sent, and often its length. It is also quite common to model functionalities which interact directly with the adversary, or whose behavior depends on which parties are corrupted.

In this section, we generalize the theory to apply to securely realizing *arbitrary* functionalities, using protocols which also use *arbitrary* functionalities as their "communication channels." Our theory also generalizes to multi-party functionalities; however, we present only the restriction of our results to 2-party functionalities. To state and prove our results for multi-party functionalities requires specialized notation, and is defered to the full version.

Definition 5. *Let $\mathcal{F}$ and $\mathcal{G}$ be 2-party functionalities and $\mathcal{T}_1$ and $\mathcal{T}_2$ be admissible machines. We define $\mathcal{F}_{\mathsf{split}}^{\mathcal{G},\mathcal{T}_1,\mathcal{T}_2}$ as the compound functionality that does the following, when interacting with honest parties P_1 and P_2 and an adversary (see Fig. 4):*

$\mathcal{F}_{\mathsf{split}}^{\mathcal{G},\mathcal{T}_1,\mathcal{T}_2}$ internally simulates instances of $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{G}$, and two independent instances of $\mathcal{F}$, which we call $\mathcal{F}_L$ and $\mathcal{F}_R$. P_1 directly interacts with $\mathcal{F}_L$ as its honest first party. P_2 directly interacts with $\mathcal{F}_R$ as its honest second party. The machine $\mathcal{T}_1$ interacts with $\mathcal{F}_L$ as its adversary, and as its corrupt second party. The machine $\mathcal{T}_2$ interacts with $\mathcal{F}_R$ as its adversary, and as its corrupt first party. Finally, $\mathcal{T}_1$ and $\mathcal{T}_2$ interact as honest first and second parties to $\mathcal{G}$, respectively. $\mathcal{F}_{\mathsf{split}}^{\mathcal{G},\mathcal{T}_1,\mathcal{T}_2}$'s adversary interacts directly as the adversary to $\mathcal{G}$.

Definition 6. *Let $\mathcal{G}$ be a functionality and $\mathcal{T}$ an admissible machine. We define $\mathcal{S}^{\mathcal{G},\mathcal{T}}$ as a simulator which internally runs a copy of $\mathcal{G}$ and $\mathcal{T}$, where $\mathcal{T}$ interacts as the adversary to the external functionality and as the two honest parties to $\mathcal{G}$. $\mathcal{S}^{\mathcal{G},\mathcal{T}}$ lets the external dummy adversary interact with $\mathcal{G}$ as its adversary (see Fig. 4).*

Definition 7 (General Splittability). *(See Fig. 4) For two 2-party functionalities $\mathcal{F}$ and $\mathcal{G}$, we say that $\mathcal{F}$ is* splittable with respect to $\mathcal{G}$ *(written $\mathcal{F} \prec \mathcal{G}$) if there exist machines $\mathcal{T}_{12}, \mathcal{T}_1, \mathcal{T}_2$ such that for all environments $\mathcal{Z}$, and the dummy adversaries $\mathcal{A}$ which corrupts no parties, we have* $\textsc{exec}[\mathcal{Z}, \mathcal{S}^{\mathcal{G},\mathcal{T}_{12}}, \partial^{\mathcal{F}}] \approx \textsc{exec}[\mathcal{Z}, \mathcal{A}, \partial^{\mathcal{F}_{\mathsf{split}}^{\mathcal{G},\mathcal{T}_1,\mathcal{T}_2}}]$. *We define* $\textsc{split}^{\mathcal{G}} = \{\mathcal{F} | \mathcal{F} \prec \mathcal{G}\}$ *and* $\textsc{split}^* = \bigcup_{\mathcal{G}} \textsc{split}^{\mathcal{G}}$.

As in the previous section, our definitions (and results) apply to both PPT and computationally unbounded systems. We write $\textsc{split}_u^{\mathcal{G}}$ or $\textsc{split}_p^{\mathcal{G}}$, $\textsc{split}_u^*$ or $\textsc{split}_p^*$ and $\mathcal{F} \prec_u \mathcal{G}$ or $\mathcal{F} \prec_p \mathcal{G}$ to explicitly specify the type of the systems.

As in Section 3.1, we aim to establish a relationship between splittability and realizability. Our main technical tools relating the two notions are given below:

Theorem 2. *For any 2-party functionalities $\mathcal{F}$, $\mathcal{G}$ and $\mathcal{H}$, the following hold:*

1. *If $\mathcal{F} \prec \mathcal{G}$ then $\mathcal{F} \sqsubseteq \mathcal{G}$.* *[Splittability implies realizability]*
2. *If $\mathcal{F} \sqsubseteq \mathcal{G} \prec \mathcal{H}$, then $\mathcal{F} \prec \mathcal{H}$.* *["Cross-transitivity"]*
3. *If $\mathcal{F} \prec \mathcal{G} \prec \mathcal{H}$, then $\mathcal{F} \prec \mathcal{H}$.* *[Transitivity of $\prec$]*
4. *If $\mathcal{F} \sqsubseteq \mathcal{G} \sqsubseteq \mathcal{H}$, then $\mathcal{F} \sqsubseteq \mathcal{H}$.* *[UC Theorem [7]]*

Proof (Sketch).

1. Analogous to the proof of Theorem 1, we can construct a protocol for $\mathcal{F}$ in the following way: The first party simulates $\mathcal{F}$ along with $\mathcal{T}_1$, while the second party simulates $\mathcal{F}$ along with $\mathcal{T}_2$ (as they interact in $\mathcal{F}_{\mathsf{split}}^{\mathcal{G},\mathcal{T}_1,\mathcal{T}_2}$). The simulator for a dummy adversary who corrupts no parties is $\mathcal{S}^{\mathcal{G},\mathcal{T}_{12}}$, and the simulator for a dummy adversary who corrupts party i can be constructed from $\mathcal{T}_i$ and $\mathcal{G}$.

2. This is the main technical tool. The proof is analogous to that of Theorem 1, but accounts for the fact that the communication channel used by a protocol for $\mathcal{F}$ is not $\mathcal{F}_{\mathsf{pvt}}$ but an arbitrary functionality $\mathcal{G}$, which in turn is splittable with respect to $\mathcal{H}$.

3. This is an immediate consequence of claims 1 and 2.

4. This is just a restatement of the universal composition theorem [7] in our notation. Though the original statement and proof of the UC theorem uses PPT systems, it is easy to see that it extends to the computationally unbounded systems as well.

In the simplified setting of Section 3.1, splittability provided an exact characterization of realizability with respect to completely private channels; namely, $\textsc{realiz}^{\mathcal{F}_{\mathsf{pvt}}} = \textsc{split}^*$. Ideally, we would like this characterization to generalize as $\textsc{realiz}^{\mathcal{F}} = \textsc{split}^{\mathcal{F}}$ for all $\mathcal{F}$, but this is not the case. For instance $\mathcal{F} \in \textsc{realiz}^{\mathcal{F}}$ for all $\mathcal{F}$, but $\mathcal{F} \notin \textsc{split}^{\mathcal{F}}$ for several functionalities, e.g., commitment. However, the characterization does generalize for a certain class of functionalities.

Definition 8. *$\mathcal{F}$ is called* self-splittable *if $\mathcal{F} \prec \mathcal{F}$. We denote the class of all self-splittable functionalities as* $\textsc{simplechannels}$.

The class $\textsc{simplechannels}$ can be viewed as a natural class of low cryptographic complexity in our landscape of complexity theory. Intuitively, $\mathcal{F} \prec \mathcal{F}$ means that $\mathcal{F}$ does not carry out any irreversible computation on its inputs. It can be easily seen that all typical communication channels (e.g., authenticated or unauthenticated, public or private, multicast or point-to-point, completely adversarially controlled), which are often implicitly incorporated into the network model, are in $\textsc{simplechannels}$.

Theorem 3. $\textsc{split}^{\mathcal{F}} = \textsc{realiz}^{\mathcal{F}}$ *for all* $\mathcal{F} \in \textsc{simplechannels}$.

In other words, functionalities which are realizable using a simple communication channel $\mathcal{F}$ are exactly those which are splittable with respect to $\mathcal{F}$. The proof follows as an easy consequence of Theorem 2. Interestingly, the simple communication channels are *exactly those functionalities for which this characterization holds*. That is, $\textsc{simplechannels} = \{\mathcal{F} \mid \textsc{split}^{\mathcal{F}} = \textsc{realiz}^{\mathcal{F}}\}$. As before, these statements hold for both PPT and computationally unbounded systems. However, note that $\textsc{simplechannels}_u$ and $\textsc{simplechannels}_p$ are different classes. For instance, a channel which applies a one-way permutation to its input is in $\textsc{simplechannels}_u \setminus \textsc{simplechannels}_p$.

Relation to the simplified definition. The simplified definition of splittability (Definition 4) was elegant and easy to apply. We would still like to be able to use this simplified definition to say as much as possible about the complexity of functionalities, even in the general setting. The following lemma gives us a tool to do just that:

Lemma 1. $\text{SPLIT}^* = \text{SPLIT}^{\mathcal{F}_{\mathsf{pvt}}}$ $(= \text{REALIZ}^{\mathcal{F}_{\mathsf{pvt}}})$.

Intuitively, $\mathcal{F}_{\mathsf{pvt}}$ is the "easiest" functionality to split with respect to. Equivalently, $\mathcal{F}_{\mathsf{pvt}}$ is the most secure channel possible in the model. The special status of $\mathcal{F}_{\mathsf{pvt}}$ is due to the fact that the network entities in our model communicate using essentially such a channel.

Most importantly, combining Lemma 1 with the characterization of Theorem 3, we see that if $\mathcal{F}$ is unsplittable according to the simple definition, then there is no secure $\mathcal{F}_{\mathsf{pvt}}$-protocol for $\mathcal{F}$, and hence no secure protocol using *any* natural communication channel. As we shall see in Section 4, it is often very easy to show that a functionality is unsplittable according to the simpler definition. Thus, splittability gives us a convenient tool to easily show impossibility results of this kind.

3.3 Deviation Revealing Functionalities

Splittability provides a convenient way to give separations that involve the relatively low-complexity functionalities of SIMPLECHANNELS. However, splittability is virtually useless in distinguishing among higher complexity functionalities, which nevertheless exhibit a rich variety in their (intuitive) cryptographic complexities. For instance, one may ask whether $\mathcal{F}_{\mathsf{OT}}$ (oblivious transfer) and $\mathcal{F}_{\mathsf{com}}$ (commitment) have different cryptographic complexities or not. In this section we develop a tool to answer many of these questions.

We introduce a notion called *deviation revealing* functionalities, which will allow us to lift existing separations of functionalities derived in simpler settings (such as the honest-but-curious model) to the standard UC setting.

Relating passive and active corruption. Consider the 2-party SFE functionality $\mathcal{F}_{\mathsf{OR}}$ that evaluates the boolean OR of Alice and Bob's input bits and outputs it to only Bob. $\mathcal{F}_{\mathsf{OR}}$ has a secure protocol in which Alice sends her input bit to Bob, and Bob locally computes the OR using that and his own input. This protocol is secure because if Bob wants to, he can learn Alice's bit even in the ideal world (by sending 0 to $\mathcal{F}_{\mathsf{OR}}$). However, this is too much information for an "honest-but-curious" Bob when his input is 1. In fact it is known [16] that there is no secure protocol for $\mathcal{F}_{\mathsf{OR}}$ in the honest-but-curious, unbounded computation setting (where corruption in the ideal world must also be passive).

As such, in general, we cannot expect results about realizability in restricted corruption scenarios to imply anything about realizability in the unrestricted corruption model. However, several natural and important functionalities do not share this odd nature of $\mathcal{F}_{\mathsf{OR}}$. We formulate the deviation revealing condition to capture such "nicely behaved" functionalities.

Corruption schemes. First we need to generalize the corruption model, to allow regular (active) corruption as well as the restricted passive (a.k.a honest-but-curious) corruption. For an m-party functionality a *corruption scheme* C is a subset of $\{\mathsf{none}, \mathsf{passive}, \mathsf{active}\}^m$. We say that a (static) adversary $\mathcal{A}$ *C-corrupts* (with respect to a protocol π) if the sequence of corruptions γ effected by $\mathcal{A}$ is in C. We will be interested in what we call *uniform corruption schemes*, wherein in each corruption sequence the corrupt parties either are all actively corrupted or are all passively corrupted: i.e., C is a uniform corruption scheme if it is a subset of $\{\mathsf{none}, \mathsf{passive}\}^m \cup \{\mathsf{none}, \mathsf{active}\}^m$.

For a corruption scheme C, we say that $\mathcal{F} \sqsubseteq^C \mathcal{G}$ if there exists a protocol π such that for all C-corrupting (with respect to π) $\mathcal{A}$, there exists a C-corrupting (with respect to the dummy protocol ∂) $\mathcal{S}$ such that for all environments $\mathcal{Z}$, $\text{EXEC}[\mathcal{Z}, \mathcal{A}, \pi^{\mathcal{G}}] \approx \text{EXEC}[\mathcal{Z}, \mathcal{S}, \partial^{\mathcal{F}}]$.

Deviation Revealing Functionalities. Intuitively, a deviation revealing functionality is one for which it is easy for the environment to detect whether an adversary is C-corrupting or not. However, to consider adversaries which deviate from C-corruption in benign ways, we use a more sophisticated definition. Note that, as with our other definitions, the following definition is given in terms of an ideal interaction with the functionality, and thus applies uniformly to arbitrary functionalities.

Definition 9. *$\mathcal{F}$ is C-*deviation-revealing *if for all adversaries $\mathcal{A}$, either:*
- *there exists a correctness environment $\mathcal{Z}$ such that $\text{EXEC}[\mathcal{Z}, \mathcal{A}, \partial^{\mathcal{F}}] \not\approx \text{EXEC}[\mathcal{Z}, \widetilde{\mathcal{A}}, \partial^{\mathcal{F}}]$, where $\widetilde{\mathcal{A}}$ is the dummy C-corrupting adversary;*
- *or, there exists a C-corrupting adversary $\mathcal{A}'$ such that for all environments $\mathcal{Z}$, $\text{EXEC}[\mathcal{Z}, \mathcal{A}, \partial^{\mathcal{F}}] \approx \text{EXEC}[\mathcal{Z}, \mathcal{A}', \partial^{\mathcal{F}}]$.*

A *correctness environment* is one which only interacts with the adversary by sending inputs to the corrupted parties and receiving their outputs.

Following is our toolkit for lifting relations in a C-corruption setting to the standard corruption setting

Theorem 4. *For any functionalities $\mathcal{F}$, $\mathcal{G}$ and $\mathcal{H}$, the following hold:*

1. *If $\mathcal{F} \sqsubseteq^C \mathcal{G} \sqsubseteq^C \mathcal{H}$, then $\mathcal{F} \sqsubseteq^C \mathcal{H}$.* *[Universal Composition.]*
2. *If $\mathcal{F}$ is C-deviation revealing for a uniform C, then*
   a. $\mathcal{F} \sqsubseteq \mathcal{G} \implies \mathcal{F} \sqsubseteq^C \mathcal{G}$ *[C-realizability from realizability.]*
   b. $(\mathcal{F} \not\sqsubseteq^C \mathcal{H} \wedge \mathcal{G} \sqsubseteq^C \mathcal{H}) \implies \mathcal{F} \not\sqsubseteq \mathcal{G}$ *[Separation from C-separation.]*

4 Applications of the Theory

In this section, we apply the general theory developed in the previous section to specific settings and classes of functionalities, to obtain several new, concrete results, as easy consequences.

4.1 Simple Impossibility Results

A compelling aspect of our splittability characterization is that all previous impossibility results for the UC model can be obtained quite easily, because the splittability definition involves only interactions with ideal functionalities.

For instance, the *bit commitment* functionality ($\mathcal{F}_{\mathsf{com}}$) is unsplittable: Consider a simple environment which asks Alice to commit to a random bit, waits for Bob to receive acknowledgement of the commeitment, instructs Alice to reveal the bit, and finally checks whether Bob received the correct bit. In any potential split of $\mathcal{F}_{\mathsf{com}}$, $\mathcal{T}$ must at some point commit to a bit in one of the instances of $\mathcal{F}_{\mathsf{com}}$, but its view at that time is by definition independent of the environment's choice, and thus the bit that Bob eventually receives will be wrong with probability $1/2$.

Similarly, the coin-tossing functionality $\mathcal{F}_{\mathsf{coin}}$ is unsplittable, because any split of $\mathcal{F}_{\mathsf{coin}}$ simulates two *independent* copies of $\mathcal{F}_{\mathsf{coin}}$, and so (regardless of $\mathcal{T}$— it sends no input to either copy of $\mathcal{F}_{\mathsf{coin}}$) the two parties' outputs will disagree with probability $1/2$. Using similar arguments, it is a very easy exercise to see that several other important 2-party functionalities, such as *oblivious transfer* ($\mathcal{F}_{\mathsf{OT}}$) and, in PPT systems, *zero-knowledge proof* for languages in NP $\setminus$ BPP, are unsplittable.

Applying Theorem 3 and Lemma 1, we can further see that these functionalities are unrealizable via protocols that use *any* simple communication channel (i.e., one from SIMPLECHANNELS). These impossibility results also rule out the possibility of *non-trivial protocols* for variants of these functionalities which allow the adversary to delay the honest parties' outputs.

4.2 Combinatorial Characterization for 2-party SFE

We use the splittability characterization for 2REGULAR to give an explicit, *combinatorial* characterization for 2-party secure function evaluation. This subsumes and completes the characterizations initiated in [10, 11]. The impossibility results in [10, 11] were later extended in [23], to the setting where certain "trusted setup" functionalities $\mathcal{F}$ are also available for protocols to use. These extensions can also be easily derived in our framework by observing that these particular functionalities $\mathcal{F}$ are self-splittable, thus impossibility under $\mathcal{F}_{\mathsf{pvt}}$ implies impossibility under $\mathcal{F}$.

Definition 10. *$\mathcal{F}$ is a* 2-party secure function evaluation (SFE) *functionality if it waits for inputs x and y from the two parties, respectively, computes two (randomized) functions $f_1(x, y)$ and $f_2(x, y)$, and sends these values to the two parties, respectively. In this case, we write $\mathcal{F} = (f_1, f_2)$.*

Note that SFE functionalities are in the class 2REGULAR. We now define two properties of 2-party SFE functionalities which will be used in our characterization. We write $f(x, y) \approx g(x', y')$ to indicate that the two distributions are indistinguishable (computationally or statistically, depending on the system).

Definition 11. *We say that* $\mathcal{F} = (f_1, f_2)$ *has* unidirectional influence *if one party's output does not depend on the other party's input. That is, if* $f_1(x, y) \approx f_1'(x)$ *for some function* f_1'*, or* $f_2(x, y) \approx f_2'(y)$ *for some function* f_2'*. Otherwise* $\mathcal{F}$ *has* bidirectional influence*.*

Definition 12. *Let* $\mathcal{F} = (f_1, f_2)$ *be a 2-party SFE functionality with unidirectional influence; say,* $f_1(x, y) \approx f_1'(x)$*. We say that* $\mathcal{F}$ *is* negligibly hiding *if there exists machines* R_1, R_2 *such that:*

$$\forall x, y : \Pr\Big[(y^*, s) \leftarrow R_1; f_2\Big(R_2\big(s, f_2(x, y^*)\big), y\Big) \not\approx f_2(x, y)\Big] \text{ is negligible}$$

The probability is over the randomness of f_1*,* f_2*,* R_1 *and* R_2*.*

If $\mathcal{F}$ is *deterministic* and has input domains of polynomial size (in the security parameter), then negligibly hiding is a simple combinatorial property: $\mathcal{F}$ is negligibly hiding if and only if there exists y such that $f_2(x, y) = f_2(x', y) \Rightarrow f_2(x, \cdot) \equiv f_2(x', \cdot)$. Our definition succinctly incorporates both the *completely revealing* and *efficiently invertible* properties of [10].

Theorem 5. *Let* $\mathcal{F} = (f_1, f_2)$ *be a 2-party SFE functionality.* $\mathcal{F}$ *is securely realizable (using* $\mathcal{F}_{\mathsf{pvt}}$*) if and only if* $\mathcal{F}$ *has unidirectional influence and is negligibly hiding.*

Proof (Sketch). We show that $\mathcal{F}$ is splittable if and only if it has unidirectional influence and is negligibly hiding. If it has bidirectional influence or is not negligibly hiding, then for every $\mathcal{T}$, it is straight-forward to construct an environment that distinguishes between $\mathcal{F}$ and $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$.

On the other hand, if $\mathcal{F}$ has unidirectional influence (say, from the first party to the second) and is negligibly hiding, then a successful strategy for $\mathcal{T}$ is to choose its input to $\mathcal{F}_L$ according to R_1, then choose its input to $\mathcal{F}_R$ according to R_2. The definition of negligible hiding implies that the second party's output from such a $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ is correct with overwhelming probability. Unidirectional influence implies that the first party's output is correct, since (roughly) it depends only on the first party's input, not on anything provided by $\mathcal{T}$).

Again, we reiterate that Theorem 5 also characterizes the existence of *nontrivial protocols* for SFE functionalities in which the adversary can delay honest parties' outputs from the functionality.

4.3 Results for Multi-party Functionalities

For multi-party functionalities involving more than two parties, where the splittability definition is much more complicated, combinatorial characterizations like that of Theorem 5 seem difficult to come by. Nonetheless, we can use 2-party results to obtain some strong necessary conditions for the multi-party setting.

A well-known technique for studying m-party SFE functionalities is the *partitioning argument*: consider *2-party SFE functionalities* induced by partitioning of the m parties into two sets. If the original functionality is realizable, then clearly so is each induced 2-party functionality.

To exploit the partitioning argument, first we extend the notion of *influence* from Definition 11 to multi-party SFE: For $i \neq j$, if there exist inputs $x_1, \ldots, x_m$ and x_i' such that $f_j(x_1, \ldots, x_m) \not\approx f_j(x_1, \ldots, x_i', \ldots, x_m)$, then we say party i *influences* the output of party j, and write $i \overset{\mathcal{F}}{\rightsquigarrow} j$.

Corollary 6 (of Theorem 5) *If $\mathcal{F}$ is an m-party SFE functionality securely realizable using completely private channels, then in the directed graph induced by $\overset{\mathcal{F}}{\rightsquigarrow}$, either all edges have a common source, or all edges have a common destination.*

We see that there are only two simple kinds of securely realizable SFE functionalities. Let p be the common vertex in the graph induced by $\overset{\mathcal{F}}{\rightsquigarrow}$. If all edges are directed towards p, then we say that $\mathcal{F}$ is *aggregated* (via party p). If all edges are directed away from p, then we say that $\mathcal{F}$ is *disseminated via party p*.

3-party characterization. We now show that in some instances, the above partioning argument does lead to a sufficient condition for realizability of multi-party functionalities. We consider the class 3REGULAR of 3-party functionalities which do not interact with the adversary, and whose behavior does not depend on which parties are corrupted. We show that for functionalities in 3REGULAR that have a secure *honest-majority protocol*,[4] the partitioning argument along with our previous 2-party characterization suffices to characterize realizability.

Theorem 7. *If $\mathcal{F} \in$ 3REGULAR has an honest-majority protocol using channel $\mathcal{G} \in$ SIMPLECHANNELS, then $\mathcal{F} \sqsubseteq \mathcal{G}$ if and only if all 2-party restrictions of $\mathcal{F}$ are UC-realizable using $\mathcal{F}_{\mathsf{pvt}}$.*

Proof (Sketch). Informally, we construct a protocol for $\mathcal{F}$ by combining the honest-majority protocol for $\mathcal{F}$ and the machines guaranteed by each 2-party splitting of $\mathcal{F}$. For adversaries that corrupt only one party, a simulator is derived from the simulator for the honest-majority protocol. For adversaries that corrupt two parties, we use the splittability criterion to construct a simulator.

In particular, this implies that realizable 3-party SFE functionalities have a simple combinatorial characterization analogous to Theorem 5. Our protocol requires each player to internally simulate executions of another protocol with a weaker/different security guarantee (in this case, the 2-party restrictions and the honest-majority protocol). This is somewhat comparable to the "MPC in the head" approach [21, 19], where significant efficiency gains are achieved in the standard corruption model by leveraging MPC protocols with security in the

[4] Honest majority protocols are known to exist for essentially all SFE functionalities using a broadcast channel. As outlined in [7], such protocols can be constructed by adapting the well-known information theoretically secure (stand-alone) protocols of [13, 5, 37].

honest-majority settings. Our constructions indicate the possibility of extending this approach by having the parties carry out not a direct protocol execution, but a related simulation.

The same approach does not seem to apply for functionalities which interact with the adversary, whose behavior depends on which parties are corrupt, or which involve more than three parties (so that two parties do not form a strict majority). We leave it as an important open problem whether the partitioning argument along with our previous 2-party characterizations suffice for characterizing multi-party functionalities *in general*. Indeed, the analogous partitioning argument has been studied for the honest-but-curious setting and shown to be insufficient in this regard [14].

4.4 A Strict Hierarchy of Intermediate Complexities

Finally, we apply the main structural result of our deviation-revealing theory (Theorem 4) to identify a sequence of functionalities with strictly increasing complexities (in unbounded computation systems).

Theorem 8. $\textsc{realiz}_u^{\mathcal{F}_{\mathsf{pvt}}} \subsetneq \textsc{realiz}_u^{\mathcal{F}^+_{\mathsf{simex}}} \subsetneq \textsc{realiz}_u^{\mathcal{F}^+_{\mathsf{com}}} \subsetneq \textsc{realiz}_u^{\mathcal{F}^+_{\mathsf{OT}}}$.

Here, $\mathcal{F}^+_{\mathsf{simex}}$, $\mathcal{F}^+_{\mathsf{com}}$, and $\mathcal{F}^+_{\mathsf{OT}}$ denote "augmented" versions of simultaneous exchange,[5] bit commitment, and oblivious transfer, respectively, in which the functionality provides multiple "sessions" instead of just one (as outlined in Section 2).

Proof (Sketch). The non-trivial observations in proving this are that $\mathcal{F}_{\mathsf{OT}} \not\sqsubseteq_u \mathcal{F}^+_{\mathsf{com}}$ and $\mathcal{F}_{\mathsf{com}} \not\sqsubseteq_u \mathcal{F}^+_{\mathsf{simex}}$. For the former, we consider the passive corruption scheme and for the latter we consider the corruption scheme where the sender (for $\mathcal{F}_{\mathsf{com}}$) could be actively corrupt and the receiver could be passively corrupt. We exploit the fact that both $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{com}}$ are deviation revealing for these corruption schemes respectively; that $\mathcal{F}_{\mathsf{com}}$ and $\mathcal{F}_{\mathsf{simex}}$ respectively have trivial protocols (using, e.g., $\mathcal{F}_{\mathsf{pvt}}$) in these settings; and that $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{com}}$ respectively do not have secure protocols (using $\mathcal{F}_{\mathsf{pvt}}$) in these settings. Then by Theorem 4, $\mathcal{F}_{\mathsf{OT}} \not\sqsubseteq_u \mathcal{F}^+_{\mathsf{com}}$ and $\mathcal{F}_{\mathsf{com}} \not\sqsubseteq_u \mathcal{F}^+_{\mathsf{simex}}$.

The significance of Theorem 8 is to establish several distinct levels of *intermediate complexity* (i.e., distinct *degrees* of the $\sqsubseteq$ reduction). That is, $\mathcal{F}_{\mathsf{simex}}$ and $\mathcal{F}_{\mathsf{com}}$ are neither realizable nor complete for computationally unbounded systems. Incidentally, both of these functionalities *are complete* for PPT systems [12]. We leave it as an important open problem whether there is a zero-one law of complexity in PPT systems (i.e., whether all functionalities not in $\textsc{realiz}^{\mathcal{F}_{\mathsf{pvt}}}$ are complete).

[5] The simulataneous exchange functionality $\mathcal{F}_{\mathsf{simex}}$ takes two inputs bits x and y from the two parties, respectively, and outputs (x, y) to both. It is called simultaneous exchange because x must be chosen without knowledge of y, and vice-versa.

Acknowledgements

We would like to thank Ran Canetti, Cynthia Dwork, Yuval Ishai and Tal Malkin for useful discussions.

References

1. B. Barak, R. Canetti, J. B. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195. IEEE, 2004.
2. B. Barak, M. Prabhakaran, and A. Sahai. Concurrent non-malleable zero knowledge. In *Proc. 47th FOCS*. IEEE, 2006.
3. B. Barak and A. Sahai. How to play almost any mental game over the net - concurrent composition using super-polynomial simulation. In *Proc. 46th FOCS*. IEEE, 2005.
4. A. Beimel, T. Malkin, and S. Micali. The all-or-nothing nature of two-party secure computation. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 1999.
5. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th STOC*, pages 1–10. ACM, 1988.
6. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067. Revised version of [7].
7. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Electronic Colloquium on Computational Complexity (ECCC) TR01-016, 2001. Previous version "A unified framework for analyzing security of protocols" availabe at the ECCC archive TR01-016. Extended abstract in FOCS 2001.
8. R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *TCC*, 2007.
9. R. Canetti and M. Fischlin. Universally composable commitments. Report 2001/055, Cryptology ePrint Archive, July 2001. Extended abstract appeared in CRYPTO 2001.
10. R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*. Springer, 2003.
11. R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.
12. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party computation. In *Proc. 34th STOC*, pages 494–503. ACM, 2002.
13. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th STOC*, pages 11–19. ACM, 1988.
14. B. Chor and Y. Ishai. On privacy and partition arguments. *Information and Computation*, 167(1):2–9, 2001.
15. B. Chor and E. Kushilevitz. A zero-one law for boolean privacy (extended abstract). In *STOC*, pages 62–72. ACM, 1989.
16. B. Chor and E. Kushilevitz. A zero-one law for boolean privacy. *SIAM J. Discrete Math.*, 4(1):36–47, 1991.
17. C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

18. O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In ACM, editor, *Proc. 19th STOC*, pages 218–229. ACM, 1987.
19. D. Harnik, Y. Ishai, E. Kushilevitz, and J. B. Nielsen. Ot-combiners via secure computation. To appear in TCC 2008, 2008.
20. D. Harnik, M. Naor, O. Reingold, and A. Rosen. Completeness in two-party secure computation: A computational view. *J. Cryptology*, 19(4):521–552, 2006.
21. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30. ACM, 2007.
22. Y. T. Kalai, Y. Lindell, and M. Prabhakaran. Concurrent general composition of secure protocols in the timing model. In *STOC*, pages 644–653. ACM, 2005.
23. D. Kidron and Y. Lindell. Impossibility results for universal composability in public-key models and with fixed inputs. Cryptology ePrint Archive, Report 2007/478, 2007. `http://eprint.iacr.org/2007/478`.
24. J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31. ACM, 1988.
25. J. Kilian. A general completeness theorem for two-party games. In *STOC*, pages 553–560. ACM, 1991.
26. J. Kilian. More general completeness theorems for secure two-party computation. In *Proc. 32th STOC*, pages 316–324. ACM, 2000.
27. J. Kilian, E. Kushilevitz, S. Micali, and R. Ostrovsky. Reducibility and completeness in private computations. *SIAM J. Comput.*, 29(4):1189–1208, 2000.
28. E. Kushilevitz. Privacy and communication complexity. In *FOCS*, pages 416–421. IEEE, 1989.
29. E. Kushilevitz. Privacy and communication complexity. *SIAM J. Discrete Math.*, 5(2):273–284, 1992.
30. E. Kushilevitz, S. Micali, and R. Ostrovsky. Reducibility and completeness in multi-party private computations. In *FOCS*, pages 478–489. IEEE, 1994.
31. Y. Lindell. General composition and universal composability in secure multi-party computation. In *Proc. 44th FOCS*. IEEE, 2003.
32. Y. Lindell. Lower bounds for concurrent self composition. In *Theory of Cryptography Conference (TCC)*, volume 1, 2004.
33. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *ACM Conference on Computer and Communications Security*, pages 245–254, 2000.
34. M. Prabhakaran. *New Notions of Security*. PhD thesis, Department of Computer Science, Princeton University, 2005.
35. M. Prabhakaran and M. Rosulek. Cryptographic complexity of multi-party computation problems: Classifications and separations. *Electronic Colloquium on Computational Complexity (ECCC)*, 15(50), 2008.
36. M. Prabhakaran and A. Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC*, pages 242–251. ACM, 2004.
37. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st STOC*, pages 73–85. ACM, 1989.
38. A. C. Yao. Protocols for secure computation. In *Proc. 23rd FOCS*, pages 160–164. IEEE, 1982.