Electronic Colloquium on Computational Complexity, Report No. 50 (2008)

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

Manoj Prabhakaran
University of Illinois
Urbana-Champaign, IL
mmp@uiuc.edu

Mike Rosulek
University of Illinois
Urbana-Champaign, IL
rosulek@uiuc.edu

March 12, 2008

Abstract

We develop new tools to study the relative complexities of secure multi-party computation tasks (functionalities) 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 the notion of *splittability*, which 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 [12] and partially extend the combinatorial conditions to the multi-party setting.

Our second main tool is the notion of *deviation-revealing* functionalities, which allows us to translate complexity separations in simpler MPC settings (such as the honest-but-curious corruption model) to the standard (malicious) setting. Applying this tool, we demonstrate the existence of functionalities which are neither realizable nor complete, in the unbounded computation model.

ISSN 1433-8092

Contents

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 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 (a secure realization of) 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) [8], 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; most previous work either focused on non-reactive functionalities (secure function evaluation), or involved arguments specific to particular functionalities. Further, the main tools we develop apply in the computationally bounded 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 [8], bit commitment, coin-tossing, and oblivious transfer [11] 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 [12, 13] (Theorem 5).

We generalize the notion of splittability as a transitive *binary relation* on functionalities. 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. We derive some simple necessary combinatorial conditions for multi-party realizability, which turn out to be sufficient for the 3-party case (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 only passive (honest-but-curious) corruption is possible, while oblivious transfer 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 arbitrary functionalities (possibly reactive, or randomized).

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 previous works focused on only the extremes of complexity.

1.1 Related Work on Complexity of MPC Functionalities

Classification of functionalities in terms of realizability has been extensively studied in many different models of MPC. Here we highlight the results most relevant to ours, arranged by the details of the particular model. Several other important results in somewhat different models are omitted. Essentially all of this previous work focused on characterizing the extremes of complexity: realizability and completeness, and further was mostly confined to SFE functionalities.

Against passively corrupting (honest-but-curious), computationally unbounded adversaries. Kilian, Kushilevitz, Micali, and Ostrovsky [33] (building on [31, 36]) showed that in the setting of secure function evaluation (SFE) of *boolean, symmetric* functions (i.e., each party receives the same single bit output), a functionality is *complete if and only if it is unrealizable* (when the adversary is allowed to corrupt an unlimited number of parties). They also show an example of a *non-boolean* function which is neither realizable nor complete. Their notion of completeness is explicitly in terms of black-box reductions (i.e., the protocols implementing all other functionalities used the complete functionality as a black-box, as in our definition of the $\sqsubseteq$ reduction).

Chor and Kushilevitz [17, 18] gave a combinatorial characterization of the set of realizable functionalities for the above class (boolean, symmetric SFE). The realizable functions in this setting are exactly those which are isomorphic to the distributed XOR function. Applying a result by Ben-Or, Goldwasser, and Wigderson [6], they observe that all such functions either remain realizable for an arbitrary number of corruptions or have secure protocols when the adversary can corrupt only a strict minority of parties.

In the setting of *two-party, symmetric* SFE (multi-bit output), Kushilevitz [34, 35] gave a complete combinatorial characterization of the realizable functionalities.

Kilian [31, 32] gave combinatorial characterizations for completeness of *two-party, probabilistic, asymmetric* (only one party receives output) SFE and *two-party, probabilistic, symmetric* SFE. The definition of completeness used here is explicitly black-box. In the asymmetric case, if the functionality is not complete, then it in fact has a simple secure realization. In the symmetric case, there are indeed functionalities with intermediate complexity: neither complete nor realizable, as observed in [33].

In a more quantitative approach to classifying unrealizable functionalities, Beaver [3] showed there is an infinite, dense hierarchy defined by the number of (black-box) invocations of the complete AND functionality needed to securely realize incomplete functionalities. Building on this, Beimel and Malkin [4] gave combinatorial characterizations for the *minimal number of invocations of the AND functionality needed* to realize symmetric SFE functionalities via deterministic protocols. They also demonstrated that randomized protocols can sometimes do exponentially better than their bounds for deterministic protocols.

Against passively corrupting, computationally bounded adversaries. Yao [41] and Goldreich, Micali, and Wigderson [22] constructed secure protocols for essentially every multi-party functionality in this setting, under standard cryptographic assumptions. Thus realizability is a trivial property of functionalities in this model. Consequently, little research has focused on relative complexities of functionalities in this setting.

One exception is the work of Harnik, Naor, Reingold, and Rosen [26], who studied completeness for *two-party, asymmetric SFE* in this setting. They give a computational analog of the combinatorial characterization for completeness in the computationally unbounded setting.

Against actively corrupting adversaries, in the stand-alone model. Against computationally bounded adversaries, essentially every MPC functionality has a secure realization via the protocols constructed by Goldreich, Micali, and Wigderson [22]. Ben-Or, Goldwasser, and Wigderson [6] and Chaum, Crépeau, and Damgård [15] constructed protocols for essentially any MPC functionality secure against unbounded adversaries that are allowed to corrupt a strict minority of parties.

One of the first functionalities observed to be complete against malicious adversaries was oblivious transfer, by Kilian [30]. Later, Kilian [32] gave a combinatorial characterization of the *two-party, asymmetric, deterministic SFE* functionalities that are complete in this setting against computationally unbounded adversaries. To show these functionalities complete, Kilian gave a secure realization of oblivious transfer using black-box access to the (augmented) functionality. We note that the protocol's simulator is a straight-line black-box simulator, and thus these completeness results also hold in the UC model.

Later, many oblivious transfer-like functionalities were studied. Damgård, Kilian, and Salvail [19] considered "information-leaking" variants of oblivious transfer (where both the sender and receiver might learn the other party's complete inputs with some probabilities), showing a sharp threshold for which relaxations remain complete. They also considered variants where the received bit may be flipped with some additional probability, and gave partial results for this case. Cachin [7] studied a very broad generalization of oblivious transfer-like functionalities, and gave information-theoretic classifications for those which remained complete. Though in the stand-alone model, these two sets of completeness results can be seen to hold in the UC model as well.

Beimel, Malkin, and Micali [5] showed a combinatorial criterion for a *two-party, asymmetric* SFE functionality to be complete. However, their definition of completeness is unique in not being based on black-box reductions among functionalities.

In the UC model. Canetti [8, 9] (and independently Pfitzmann and Waidner [37]) introduced the general framework of network-aware security known as UC security. Many modifications to the original UC framework have been proposed, which allow for all MPC functionalities to have secure realizations [14, 1, 39, 2, 28, 10], however, not all functionalities are realizable in the original framework:

The first impossibility results in the framework were already given for coin-tossing, zero-knowledge, and oblivious transfer by Canetti [8]. Later, commitment was shown to be unrealizable in the framework by Canetti and Fischlin [11]. Canetti, Kushilevitz, and Lindell [12] showed several broad impossibility results for many classes of *two-party* SFE in the framework. For *single-input* functions, *symmetric* functions, and for *asymmetric* functions over finite domains, their results provide a complete characterization of realizability.

The impossibility results developed in [12] were later extended by Kidron and Lindell [29] and shown to hold even in the presence of certain "set-up" functionalities. They also showed that certain related set-up functionalities would admit secure protocols for all functionalities. Our splittability classification can be viewed as a generalization and extension of the techniques used in [12, 29] to show impossibilities.

2 Preliminaries

Some of our conventions differ from the original UC model. We now overview the model, highlighting these 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 (modeled as interactive Turing Machines or IO-automata): 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. The specifics of how the entities interact can be formalized in different ways; see Appendix A for one which aims to abstractly capture a realistic execution of protocols in a network.

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 remains admissible itself.[1] Our main results (as well as the main universal composition theorem) apply uniformly for any such system, the two most natural of which are *computationally unbounded systems* (which

[1]As such, our theory is *not* directly applicable to the network-aware security model introduced in [39, 38] and also used in [2], where an adversary can sometimes access extra computational power that an environment cannot.

admit all probabilistic machines) and *PPT systems* (which admit all probabilistic, polynomial-time machines).

Unlike the 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 would be able to tamper with and delay such communications. Instead, we assume that functionalities can achieve the same effect by directly interacting with the adversary each time a party communicates with the functionality. This difference is significant in considering *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 via a functionality. In this way, we are able to uniformly consider protocols over 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 one or more simpler functionalities. This convention maintains the strict binary nature of the 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" variant of $\mathcal{F}$ could provide an interface to multiple different "port-mappings" of $\mathcal{F}$. 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 convention of the $\sqsubseteq$ notation. Thus all of our results may be interchangeably 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 distribution of the environment $\mathcal{Z}$'s (single-bit) output 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 $\mathcal{Z}$, $\textsc{exec}[\mathcal{Z}, \mathcal{A}, \pi^{\mathcal{G}}] \approx \textsc{exec}[\mathcal{Z}, \mathcal{S}, \partial^{\mathcal{F}}]$. ($\approx$ can denote statistical, computational or perfect indistinguishability, depending on the system.) 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 which have secure realizations with respect to $\mathcal{G}$. Our main results apply to both PPT and unbounded systems in a unified way, so we often do not distinguish between the two systems in our notation. To explicitly refer to realizability in 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 may delay communications between a functionality and parties, a protocol which does nothing is a secure realization since the same effect can be achieved in the ideal world by an adversary who indefinitely blocks all communication. Thus it is necessary to restrict attention to so-called *non-trivial* protocols [14, 12] 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, but we may consider functionalities that explicitly interact with the adversary, allowing it to block or delay its communication with its honest parties. For such functionalities, we must also consider a definition of non-triviality for our results to be meaningful.

Definition 2.1 *Let* $\mathsf{wrap}(\mathcal{F})$ *be the functionality that runs* $\mathcal{F}$*, except all outputs generated by* $\mathcal{F}$ *are not immediately delivered, but entered in a 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.*

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 $\mathsf{wrap}(\mathcal{F})$, provided that some liveness condition were guaranteed.

Definition 2.2 *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).

The important implication of Definition 2.2 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.

Comparison to existing definition. We note that our definition of non-triviality is slightly stronger than the non-triviality condition in [14, 12]. Translated into our conventions, the original definition insists that the protocol π (for $\mathsf{wrap}(\mathcal{F}) \sqsubseteq \mathsf{wrap}(\mathcal{G})$) also realize $\mathcal{F} \sqsubseteq \mathcal{G}$, but only against adversaries who corrupt *no parties*. While this is arguably sufficient for simple functionalities such as secure function evaluation, we claim that it still leaves room for some pathological functionalities with unsatisfactory non-trivial realizations.

Consider the following "pseudo oblivious transfer" functionality $\mathcal{F}_{\mathsf{pOT}}$: It receives bits x_0, x_1 from Alice. Then Bob may send a bit b to the functionality and receive x_b in return. Unlike normal oblivious transfer, Bob may do this as many times as he likes, but each time he does, $\mathcal{F}_{\mathsf{pOT}}$ tells Alice that Bob fetched a bit (though it does not tell Alice which bit he fetched). $\mathcal{F}_{\mathsf{pOT}}$ itself is easily seen to be unsplittable, and thus has no secure protocol using private channels. But now consider the protocol for $\mathsf{wrap}(\mathcal{F}_{\mathsf{pOT}})$ where Alice simply sends Bob both of her bits to Bob. Bob only outputs the bit he desires to pick up. This protocol is secure because in the ideal world, a malicious Bob can learn both of Alice's inputs, but block the functionality's second notification to Alice. It is non-trivial according to the weaker non-triviality definition. Nevertheless it provides none of the cryptographic properties $\mathcal{F}_{\mathsf{pOT}}$ has, and the simulator can be viewed as "exploiting" the message delivery wrapper.

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 $\textsc{realiz}^{\mathcal{G}}$ is a theory of *splittability*. For expositional clarity, first we present a special case of our splittability theory, which captures the essential intuition and still has useful consequences in the more general setting. The simplification involves restricting

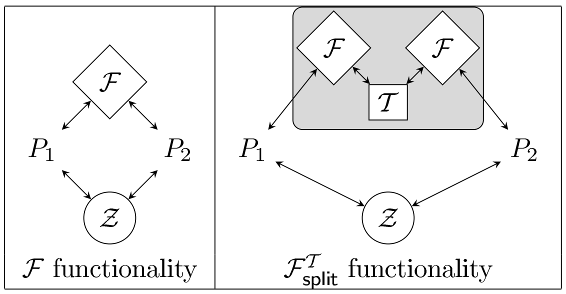
$\mathcal{F}$
P_1
P_2
$\mathcal{Z}$
$\mathcal{F}$ functionality
$\mathcal{T}$
$\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ functionality

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

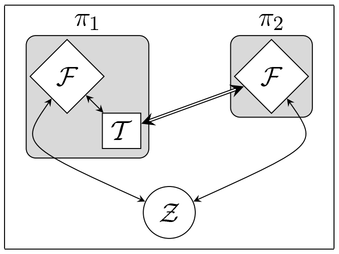

Figure 2: Secure protocol (over private channels) for a splittable functionality $\mathcal{F}$.

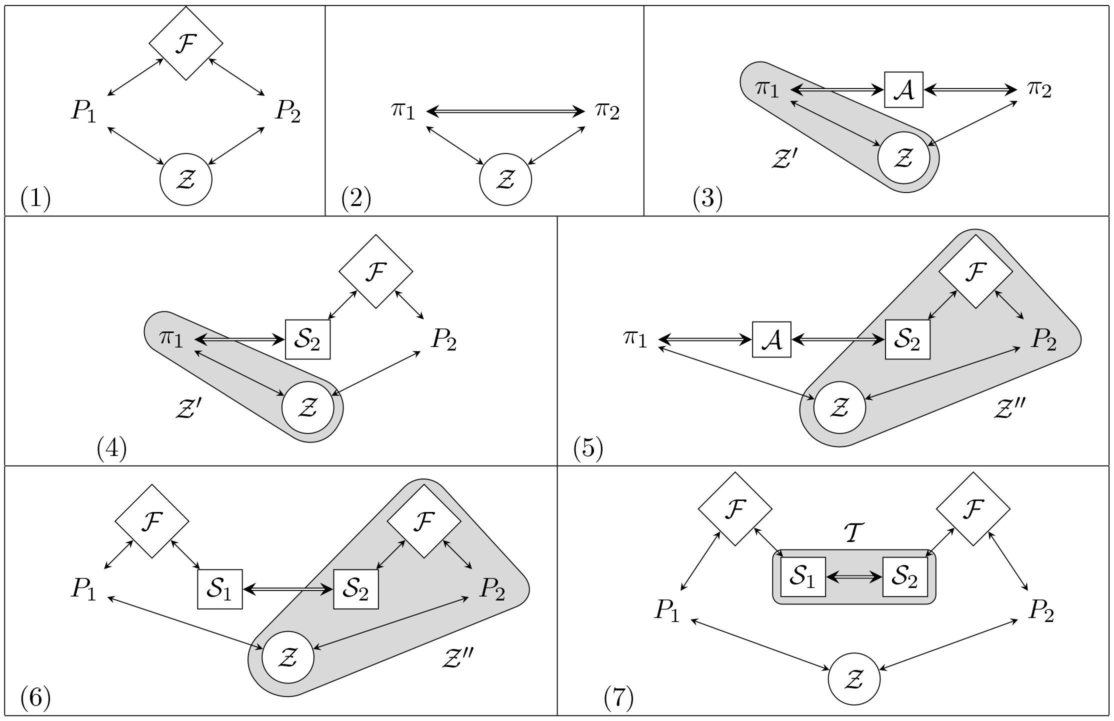

Figure 3: Steps in the proof of Theorem 1. Given a protocol securely realizing $\mathcal{F}$, we apply the security guarantee three times: first with no parties corrupted (between boxes 1 and 2), then with a corrupt party P_1 which plays a man-in-the-middle between P_2 and an honest P_1 inside the environment (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.

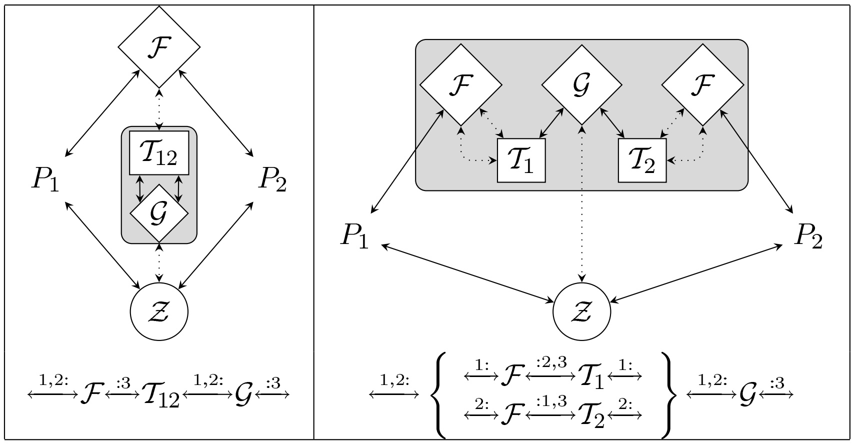
$\mathcal{F}$
$\mathcal{G}$
$\mathcal{T}_{12}$
$\mathcal{T}_1$
$\mathcal{T}_2$
P_1
P_2
$\mathcal{Z}$
$\xleftrightarrow{1,2:}\mathcal{F}\xleftrightarrow{:3}\mathcal{T}_{12}\xleftrightarrow{1,2:}\mathcal{G}\xleftrightarrow{:3}$
$\xleftrightarrow{1,2:}\left\{\begin{matrix}\xleftrightarrow{1:}\mathcal{F}\xleftrightarrow{:2,3}\mathcal{T}_1\xleftrightarrow{1:}\\ \xleftrightarrow{2:}\mathcal{F}\xleftrightarrow{:1,3}\mathcal{T}_2\xleftrightarrow{2:}\end{matrix}\right\}\xleftrightarrow{1,2:}\mathcal{G}\xleftrightarrow{:3}$

Figure 4: A visual representation of general 2-party splittability (Definition 3.4). If these two compound functionalities are indistinguishable then $\mathcal{F} \prec \mathcal{G}$.

ourselves 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. We also restrict attention to secure protocols which use an idealized communication channel. We let $\mathcal{F}_{\mathsf{pvt}}$ denote the *completely private channel* functionality, which allows parties to send private messages to other parties of their choice, but does not interact with the adversary *at all* — not even to notify it that a message was sent. In the next section we shall remove these restrictions and present the general theory.

Definition 3.1 *Let $\mathcal{F}$ be a 2-party functionality and $\mathcal{T}$ an admissible machine. Define $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ as the compound functionality that does the following (See Figure 1):*

$\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ 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 $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ directly interacts with $\mathcal{F}_L$ as its first party. The second party of $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ 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.

We define splittability of $\mathcal{F}$ in terms of $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$.

Definition 3.2 *$\mathcal{F} \in$ 2REGULAR is* splittable *if there exists a machine $\mathcal{T}$ such that $\mathcal{F}$ is indistinguishable from $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$. That is, for all environments $\mathcal{Z}$ and the dummy adversary $\mathcal{A}$ that corrupts no one, we have* $\text{EXEC}[\mathcal{Z}, \mathcal{A}, \partial^{\mathcal{F}}] \approx \text{EXEC}[\mathcal{Z}, \mathcal{A}, \partial^{\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}}]$, *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.

Theorem 1 *$\mathcal{F} \in$ 2REGULAR is securely realizable (using $\mathcal{F}_{\mathsf{pvt}}$) if and only if $\mathcal{F}$ is splittable. That is,* 2REGULAR $\cap$ $\text{REALIZ}^{\mathcal{F}_{\mathsf{pvt}}}$ = 2REGSPLIT.

The easier direction is to see that 2REGSPLIT $\subseteq$ $\text{REALIZ}^{\mathcal{F}_{\mathsf{pvt}}} \cap$ 2REGULAR. If $\mathcal{F}$ is splittable, then a protocol (π_1, π_2) can be derived as shown in Figure 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 [12], and also has parallels with an impossibility proof by Dwork et al. [20].[2] A visual overview is given in Figure 3.

PROOF: (If $\mathcal{F}$ is realizable, then $\mathcal{F}$ is splittable) Suppose $\mathcal{F}$ is securely realizable with protocol π, where $\mathcal{S}_X$ is the simulator for the dummy adversary which corrupts the parties indexed by $\overline{X} = \{1, 2\} \setminus X$.

Take any environment $\mathcal{Z}$ and consider an ideal-world interaction where the players communicate directly with $\mathcal{F}$, and neither party is corrupted (box 1 in Figure 3). There is no need for an external adversary since $\mathcal{F}$ does not communicate with the adversary when no parties are corrupted. By the security of π, this is indistinguishable to the real-world execution where the parties engage in

[2]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.

protocol π on a completely private channel (box 2 in Figure 3). The channel does not leak any information to the adversary.

From this real-world interaction, consider letting the environment engulf the honest first party (call this new environment $\mathcal{Z}'$). Then, add a dummy adversary $\mathcal{A}$, which corrupts the first party and simply relays the messages between the second party and the honest first party inside $\mathcal{Z}'$ (box 3 in Figure 3). By construction, both parties are are engaging in the prescribed protocol (though with the dummy in between), so this interaction is identical from the previous ones.

By the security properties of π, this is indistinguishable from an ideal world interaction, where the second party communicates directly with $\mathcal{F}$, and the dummy adversary is replaced with its simulator $\mathcal{S}_2$. $\mathcal{S}_2$ still interacts with the embedded first party inside the environment (simulating the π protocol), and interacts with $\mathcal{F}$ as a corrupted first party (box 4 in Figure 3).

Now, embed $\mathcal{F}$, the honest second party, and $\mathcal{S}_2$ inside the environment, and take the embedded first party (still honestly running the π protocol) outside the environment. Call this new environment $\mathcal{Z}''$. Insert another dummy adversary $\mathcal{A}$ as a corrupted second party, which relays messages between the honest first party (outside the environment) and $\mathcal{S}_2$ (now embedded inside the environment) (box 5 in Figure 3). We have done nothing but insert honest message relaying and moved things in and out of the environment, so the interaction is identical to the previous ones.

This new interaction involves an honest first party running the π protocol with a corrupted second. Similar to before, we apply π's security guarantee to show that this is indistinguishable from the ideal world where the first party now interacts directly with $\mathcal{F}$ (an instance of $\mathcal{F}$ separate from the one embedded inside the environment), while the dummy adversary is replaced by its simulator $\mathcal{S}_1$. $\mathcal{S}_1$ interacts with (the external) $\mathcal{F}$ as a corrupted second party, and simulates the π protocol interaction with the $\mathcal{S}_2$ inside the environment (box 6 in Figure 3).

Finally, take out everything from the environment except for the original $\mathcal{Z}$: the ideal second party, the simulators $\mathcal{S}_1$ and $\mathcal{S}_2$, and the completely private channel on which they are still simulating the π protocol to each other (box 7 in Figure 3). By composing $\mathcal{S}_1$, $\mathcal{S}_2$, and their channel, we obtain the machine $\mathcal{T}$ as required by the splittability definition. By our construction, this split functionality $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ is indistinguishable to the environment $\mathcal{Z}$.

(If $\mathcal{F}$ is splittable, then $\mathcal{F}$ is realizable) Suppose $\mathcal{F}$ is splittable; then the following is a secure protocol for $\mathcal{F}$ (see Figure 2): Party P_1 simulates a local copy of $\mathcal{F}_L$ and $\mathcal{T}$, interacting as in the splittability definition. He relays his inputs and outputs as the first party to $\mathcal{F}_L$. Party P_2 simulates a local copy of $\mathcal{F}_R$, and relays his inputs and outputs as the second party to $\mathcal{F}_R$. The messages sent across the completely private channel as the protocol are those messages between $\mathcal{T}$ and $\mathcal{F}_R$. In $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$, the communication between $\mathcal{T}$ and $\mathcal{F}_R$ is across a channel of the same kind. Thus, the interaction between two parties honestly running this protocol is exactly that of two parties accessing $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ using the dummy protocol. By the splittability condition, the interaction is indistinguishable from an ideal interaction with $\mathcal{F}$.

The simulators for this protocol are quite simple: For a dummy adversary that corrupts P_1, the simulator is also a dummy adversary; for one that corrupts P_2, the simulator is $\mathcal{T}$. $\square$

As with all our structural results, Theorem 1 can be specialized as $2\textsc{regular}_u \cap \textsc{realiz}_u^{\mathcal{F}_{\mathsf{pvt}}} = 2\textsc{regsplit}_u$ and $2\textsc{regular}_p \cap \textsc{realiz}_p^{\mathcal{F}_{\mathsf{pvt}}} = 2\textsc{regsplit}_p$, where the complexity classes are subscripted to explicitly indicate unbounded and PPT systems.

3.2 General Theory of Splittability

Protocols in network-aware frameworks are generally modeled to use less idealized channels than the one considered in the previous section ($\mathcal{F}_{\mathsf{pvt}}$). 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 realization of *arbitrary* functionalities, using protocols which also use *arbitrary* functionalities as their "communication channels." Furthermore, our theory extends to multi-party (instead of only 2-party) functionalities.

3.2.1 Notational Conventions

In this section, we construct many "compound" functionalities which simulate the interactions of several machines. The notation we use describes the complicated interactions between the many components, and allows for some simple symbolic manipulation of equalities. Using this notation, we can carry out the proof at a reasonably high level.

Let $\mathcal{F}$ be an m-party functionality. $\mathcal{F}$ may interact with an adversary even if no parties are corrupted, and we consider such an adversary the $(m+1)$th party to the functionality. By convention, we let $X \subseteq \{1, \ldots, m\}$ represent the indices of uncorrupted parties, and $\overline{X} = \{1, \ldots, m+1\} \setminus X$ the indices of the corresponding corrupted parties (always containing party $m+1$).

The machines in our network model are interactive machines with "communication ports" that allow them to interact with other machines. When one of the interacting machines is a functionality, we can identify these ports by which parties they correspond to from the functionality's point of view. We take this approach in our notation, writing "$\xleftrightarrow{a}\mathcal{M}\xleftrightarrow{b}$" to denote a machine $\mathcal{M}$ with two distinguished "bundles" of ports, labeled by a and b. These labels may be of the form "$: X$" or "$X :$", where $X \subseteq \{1, \ldots, m+1\}$ denotes a set of indices for parties. These communication ports are "duplex", and thus the placement of the colon is significant; it determines the "parity" of the ports. It is our convention to place the colon in the direction of a machine that acts as a functionality. Thus, "$\xleftrightarrow{X:}\mathcal{M}$" denotes that $\mathcal{M}$ is acting as a functionality for parties indexed by X, while "$\xleftrightarrow{:X}\mathcal{M}$" denotes that $\mathcal{M}$ is expecting to interact with a functionality as the parties indexed by X.

The left-to-right ordering of in our notation is also significant, since for functionalities, the two bundles of communication ports correspond to the corrupted and uncorrupted parties. Below we describe our conventions for different kinds of machines in the interactions we consider, which can conveniently be identified by the "parities" of their left and right communication ports:

- We write $\xleftrightarrow{X:}\mathcal{F}\xleftrightarrow{:\overline{X}}$ to denote an interaction with a functionality $\mathcal{F}$ where the parties indexed by X are uncorrupted and the parties indexed by $\overline{X}$ are corrupted (note that the behavior of $\mathcal{F}$ may indeed depend on which parties are corrupted).

- We write $\xleftrightarrow{i:}\pi_i\xleftrightarrow{i:}$ to denote the ith party's protocol program for π. On the left, it passes inputs and outputs (presumably with the environment) as party i, and on the right it interacts with its "channel" as party i.

- We write $\xleftrightarrow{:\overline{X}}\mathcal{S}\xleftrightarrow{:\overline{X}}$ to denote the simulator for a dummy adversary who corrupts the parties indexed by $\overline{X}$. On the left, it interacts with an ideal functionality as corrupt parties $\overline{X}$. On the right, it simulates a real-world view of the corrupt parties $\overline{X}$ to the adversary (or generally, directly with the environment).

- Finally, we write $\xleftrightarrow{:\overline{X}}\mathcal{T}\xleftrightarrow{X:}$ to denote a "translator" machine $\mathcal{T}$ of the kind we will require for our splittability definitions. It interacts on the left (presumably with some functionality) as corrupt parties indexed by $\overline{X}$, and on the right (presumably with some other functionality) as uncorrupted parties indexed by X.

These notational conventions allow us to express compositions and complicated combinations of several machines in the following ways:

- When we have two machines $\xleftrightarrow{a}\mathcal{M}_1\xleftrightarrow{b}$ and $\xleftrightarrow{b}\mathcal{M}_2\xleftrightarrow{c}$, where the labels b match completely (their set of indices and their "parity" both match), we may write $\xleftrightarrow{a}\mathcal{M}_1\xleftrightarrow{b}\mathcal{M}_2\xleftrightarrow{c}$ to denote the "compound" machine which internally simulates both $\mathcal{M}_1$ and $\mathcal{M}_2$, and their interaction together, while interacting externally according to the label a on the left and c on the right.

- Let $X = \{1, \ldots, k\}$ without loss of generality, and let b be any label in our notation. When we have machines $\xleftrightarrow{X:}\mathcal{M}\xleftrightarrow{b}$ and $\xleftrightarrow{i:}\pi_i\xleftrightarrow{i:}$ for each $i \in X$, we may write:

$$\xleftrightarrow{X:}\left\{\begin{matrix}\xleftrightarrow{1:}\pi_1\xleftrightarrow{1:}\\ \vdots \\ \xleftrightarrow{k:}\pi_k\xleftrightarrow{k:}\end{matrix}\right\}\xleftrightarrow{X:}\mathcal{M}\xleftrightarrow{b}$$

  to denote the compound machine which internally simulates the interactions among $\mathcal{M}$ and each π_i, with communication ports connected according to their indices.

It is important that the network model's definition of *admissible* machines guarantees that such compositions of machines are also admissible.

When a (compound) machine $\mathcal{M}$ has the form $\xleftrightarrow{X:}\mathcal{M}\xleftrightarrow{:\overline{X}}$, we may view it as a functionality itself. We say that two (compound) functionalities $\xleftrightarrow{X:}\mathcal{F}_1\xleftrightarrow{:\overline{X}}$ and $\xleftrightarrow{X:}\mathcal{F}_2\xleftrightarrow{:\overline{X}}$ are *indistinguishable* if for all environments $\mathcal{Z}$,

$$\textsc{exec}[\mathcal{Z}, \mathcal{A}, \partial^{\mathcal{F}_1}] \approx \textsc{exec}[\mathcal{Z}, \mathcal{A}, \partial^{\mathcal{F}_2}]$$

where $\mathcal{A}$ is the dummy adversary which corrupts parties indexed by $\overline{X}$, and ∂ is the dummy protocol.

Under this interpretation, we can rephrase the UC security definition in terms of our notation. $\pi = (\pi_1, \ldots, \pi_m)$ is a secure realization of $\mathcal{F}$ with respect to $\mathcal{G}$ if for all $X = \{x_1, \ldots, x_k\} \subseteq \{1, \ldots, m\}$, there exists a simulator $\mathcal{S}_X$ such that the following two compound functionalities are indistinguishable:

$$\xleftrightarrow{X:}\left\{\begin{matrix}\xleftrightarrow{x_1:}\pi_{x_1}\xleftrightarrow{x_1:}\\ \vdots \\ \xleftrightarrow{x_k:}\pi_{x_k}\xleftrightarrow{x_k:}\end{matrix}\right\}\xleftrightarrow{X:}\mathcal{G}\xleftrightarrow{:\overline{X}} \qquad \text{and} \qquad \xleftrightarrow{X:}\mathcal{F}\xleftrightarrow{:\overline{X}}\mathcal{S}_X\xleftrightarrow{:\overline{X}}$$

Note that we do not need to explicitly mention the dummy protocol.

Finally, we observe the following *rewriting rule*, which generalizes the technique used in the proof of Theorem 1 of bringing components of the interaction into and out of the environment:

Lemma 1 *If functionalities* $\xleftrightarrow{X:}\mathcal{M}_1\xleftrightarrow{:\overline{X}}$ *and* $\xleftrightarrow{X:}\mathcal{M}_2\xleftrightarrow{:\overline{X}}$ *are indistinguishable, then so are*

$$\xleftrightarrow{Y:}\mathcal{M}_L\xleftrightarrow{X:}\mathcal{M}_1\xleftrightarrow{:\overline{X}}\mathcal{M}_R\xleftrightarrow{:\overline{Y}} \qquad \textit{and} \qquad \xleftrightarrow{Y:}\mathcal{M}_L\xleftrightarrow{X:}\mathcal{M}_2\xleftrightarrow{:\overline{X}}\mathcal{M}_R\xleftrightarrow{:\overline{Y}},$$

for any $Y \subseteq \{1, \ldots, m\}$ *and machines* $\mathcal{M}_L, \mathcal{M}_R$.

Proof: Consider an arbitrary environment $\mathcal{Z}$ interacting with $\xleftrightarrow{Y:}\mathcal{M}_L\xleftrightarrow{X:}\mathcal{M}_1\xleftrightarrow{:\overline{X}}\mathcal{M}_R\xleftrightarrow{:\overline{Y}}$. We may subsume $\mathcal{M}_L$ and $\mathcal{M}_R$ into the environment,[3] obtaining a new environment $\mathcal{Z}'$ in such a way that the overall interaction is identical, but $\mathcal{Z}'$ interacts only with $\xleftrightarrow{X:}\mathcal{M}_1\xleftrightarrow{:\overline{X}}$. By their indistinguishability, we may replace the functionality with $\mathcal{M}_2$ accordingly, and then remove $\mathcal{M}_L, \mathcal{M}_R$ from the environment. We end with the original environment $\mathcal{Z}$ interacting with $\xleftrightarrow{Y:}\mathcal{M}_L\xleftrightarrow{X:}\mathcal{M}_2\xleftrightarrow{:\overline{X}}\mathcal{M}_R\xleftrightarrow{:\overline{Y}}$, and have not changed the outcome of the interaction by more than a negligible amount at each step. □

3.2.2 General Splittability Definition

Definition 3.3 *We say that an m-party functionality $\mathcal{F}$ is* splittable *with respect to another m-party functionality $\mathcal{G}$ if there exist admissible machines $\{\mathcal{T}_X | X \subseteq \{1, \ldots, m\}\}$ such that the following two compound functionalities are indistinguishable for every $X = \{x_1, \ldots, x_k\} \subseteq \{1, \ldots, m\}$:*

$$\xleftrightarrow{X:}\mathcal{F}\xleftrightarrow{:\overline{X}}\mathcal{T}_X\xleftrightarrow{X:}\mathcal{G}\xleftrightarrow{:\overline{X}} \qquad \textit{and} \qquad \xleftrightarrow{X:}\left\{\begin{array}{c}\xleftrightarrow{x_1:}\mathcal{F}\xleftrightarrow{:\overline{x_1}}\mathcal{T}_{x_1}\xleftrightarrow{x_1:}\\ \vdots \\ \xleftrightarrow{x_k:}\mathcal{F}\xleftrightarrow{:\overline{x_k}}\mathcal{T}_{x_k}\xleftrightarrow{x_k:}\end{array}\right\}\xleftrightarrow{X:}\mathcal{G}\xleftrightarrow{:\overline{X}}$$

When this is the case, we write $\mathcal{F} \prec \mathcal{G}$. We define $\textsc{split}^{\mathcal{G}} = \{\mathcal{F} \,|\, \mathcal{F} \prec \mathcal{G}\}$ *and* $\textsc{split}^* = \bigcup_{\mathcal{G}} \textsc{split}^{\mathcal{G}}$.

Intuitively, a copy of $\mathcal{F}$ interacting honest parties indexed by X may be "split" into separate copies of $\mathcal{F}$, one for each of the honest parties. As in the previous section, our definitions (and results) apply to both PPT and computationally unbounded systems. We write $\textsc{split}^{\mathcal{G}}_u$ or $\textsc{split}^{\mathcal{G}}_p$, $\textsc{split}^*_u$ 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.

Examples. For comparison to the simpler splittability definition for 2regular, we show the general definition of splittability restricted to the two-party case.

Definition 3.4 *Let $\mathcal{F}$ and $\mathcal{G}$ be 2-party functionalities. Then $\mathcal{F}$ is splittable with respect to $\mathcal{G}$ if there exist machines $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_{12}$ such that:*

$$\xleftrightarrow{1,2:}\mathcal{F}\xleftrightarrow{:3}\mathcal{T}_{12}\xleftrightarrow{1,2:}\mathcal{G}\xleftrightarrow{:3} \qquad \approx \qquad \xleftrightarrow{1,2:}\left\{\begin{array}{c}\xleftrightarrow{1:}\mathcal{F}\xleftrightarrow{:2,3}\mathcal{T}_1\xleftrightarrow{1:}\\ \xleftrightarrow{2:}\mathcal{F}\xleftrightarrow{:1,3}\mathcal{T}_2\xleftrightarrow{2:}\end{array}\right\}\xleftrightarrow{1,2:}\mathcal{G}\xleftrightarrow{:3}$$

See Figure 4 for a visual overview of the two-party case. The splittability definition for the two-party case only involves the requirement for $X = \{1, 2\}$, since the indistinguishability conditions for the other three subsets are tautologies. See also Figure 5 for one of the splittability conditions ($X = \{1, 2, 3\}$) for 3-party functionalities.

3.2.3 Theorems

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 functionalities $\mathcal{F}$, $\mathcal{G}$ and $\mathcal{H}$, the following hold:*

[3] According to the definition of *admissible* machines, the composition of these machines with the environment machine is itself admissible.

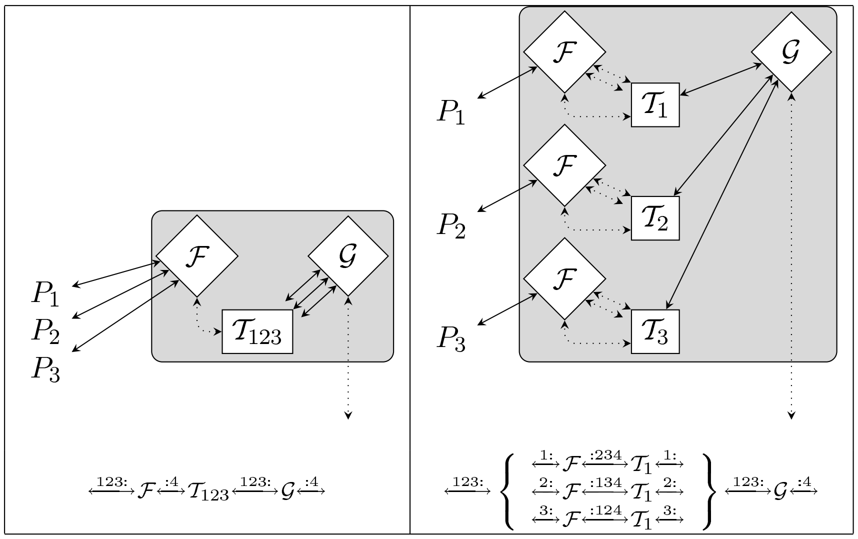
$\mathcal{F}$
$\mathcal{G}$
P_1
P_2
P_3
$\mathcal{T}_{123}$
$\mathcal{T}_1$
$\mathcal{T}_2$
$\mathcal{T}_3$
$\xleftrightarrow{123:}\mathcal{F}\xleftrightarrow{:4}\mathcal{T}_{123}\xleftrightarrow{123:}\mathcal{G}\xleftrightarrow{:4}$
$\xleftrightarrow{123:}\left\{\begin{array}{c}\xleftrightarrow{1:}\mathcal{F}\xleftrightarrow{:234}\mathcal{T}_1\xleftrightarrow{1:}\\ \xleftrightarrow{2:}\mathcal{F}\xleftrightarrow{:134}\mathcal{T}_1\xleftrightarrow{2:}\\ \xleftrightarrow{3:}\mathcal{F}\xleftrightarrow{:124}\mathcal{T}_1\xleftrightarrow{3:}\end{array}\right\}\xleftrightarrow{123:}\mathcal{G}\xleftrightarrow{:4}$

Figure 5: One of the conditions in required by the definition of $\mathcal{F} \prec \mathcal{G}$, for 3-party functionalities, and with $X = \{1, 2, 3\}$. Dotted lines indicate interactions as adversary and corrupted players.

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" of splittability and realizability]*
3. If $\mathcal{F} \prec \mathcal{G} \prec \mathcal{H}$, then $\mathcal{F} \prec \mathcal{H}$.	*[Transitivity of splittability]*
4. If $\mathcal{F} \sqsubseteq \mathcal{G} \sqsubseteq \mathcal{H}$, then $\mathcal{F} \sqsubseteq \mathcal{H}$.	*[UC Theorem [8]]*

Note that (3) is an immediate consequence of (1) and (2); (4) is simply a restatement of the UC theorem in our notation.

We prove (1) and (2) in the following lemmas:

Lemma 2 *If $\mathcal{F} \prec \mathcal{G}$ then $\mathcal{F} \sqsubseteq \mathcal{G}$.*

Proof: Suppose that $\mathcal{F} \prec \mathcal{G}$, as witnessed by machines $\{\mathcal{T}_X^{\mathcal{F}\prec\mathcal{G}}\}_X$. Then we claim that protocol wherein party i executes $\xleftrightarrow{i:}\mathcal{F}\xleftrightarrow{:\overline{i}}\mathcal{T}_i^{\mathcal{F}\prec\mathcal{G}}\xleftrightarrow{i:}$ is a secure protocol for $\mathcal{F}$ with respect to $\mathcal{G}$. To show the security of this protocol, we must show that the UC security definition holds. For each subset of parties $X = \{x_1, \ldots, x_k\}$, we apply the splittability condition to obtain:

$$\xleftrightarrow{X:}\left\{\begin{array}{c}\xleftrightarrow{x_1:}[\mathcal{F}\xleftrightarrow{:\overline{x_1}}\mathcal{T}_{x_1}^{\mathcal{F}\prec\mathcal{G}}]\xleftrightarrow{x_1:}\\ \vdots \\ \xleftrightarrow{x_k:}[\mathcal{F}\xleftrightarrow{:\overline{x_k}}\mathcal{T}_{x_k}^{\mathcal{F}\prec\mathcal{G}}]\xleftrightarrow{x_k:}\end{array}\right\}\xleftrightarrow{X:}\mathcal{G}\xleftrightarrow{:\overline{X}} \qquad \approx \qquad \xleftrightarrow{X:}\mathcal{F}\xleftrightarrow{:\overline{X}}[\mathcal{T}_X^{\mathcal{F}\prec\mathcal{G}}\xleftrightarrow{X:}\mathcal{G}]\xleftrightarrow{:\overline{X}}$$

Thus, the machine $\xleftrightarrow{:\overline{X}}\mathcal{T}_X^{\mathcal{F}\prec\mathcal{G}}\xleftrightarrow{X:}\mathcal{G}\xleftrightarrow{:\overline{X}}$ satisfies the condition to be the simulator for the dummy adversary who corrupts parties indexed by $\overline{X}$. We conclude that the above protocol realizing $\mathcal{F}$ with respect to $\mathcal{G}$ is secure. $\square$

One interesting consequence of the above lemma is that without loss of generality, the protocol for $\mathcal{F}$ treats $\mathcal{F}$ as a black box, and the simulator for the protocol simulates $\mathcal{G}$ as a black box as well.

Lemma 3 *If $\mathcal{F} \sqsubseteq \mathcal{G} \prec \mathcal{H}$, then $\mathcal{F} \prec \mathcal{H}$.*

Proof: Suppose $\mathcal{F} \sqsubseteq \mathcal{G}$ as witnessed by a protocol π and simulators/transvisors $\{\mathcal{S}_X^{\mathcal{F}\sqsubseteq\mathcal{G}}\}_X$, and that $\mathcal{G} \prec \mathcal{H}$ as witnessed by $\{\mathcal{T}_X^{\mathcal{G}\prec\mathcal{H}}\}_X$.

To show that $\mathcal{F} \prec \mathcal{H}$, we must demonstrate suitable machines $\{\mathcal{T}_X^{\mathcal{F}\prec\mathcal{H}}\}_X$. Applying the above conditions, we have that for each subset of parties $X = \{x_1, \dots, x_k\}$:

$$\begin{aligned}
\xleftrightarrow{X:}\mathcal{F}\xleftrightarrow{:\overline{X}}[\mathcal{S}_X^{\mathcal{F}\sqsubseteq\mathcal{G}}\xleftrightarrow{:\overline{X}}\mathcal{T}_X^{\mathcal{G}\prec\mathcal{H}}]\xleftrightarrow{X:}\mathcal{H}\xleftrightarrow{:\overline{X}}
&\approx \xleftrightarrow{X:}\left\{\begin{array}{c}\xleftrightarrow{x_1:}\pi_{x_1}\xleftrightarrow{x_1:}\\ \vdots \\ \xleftrightarrow{x_k:}\pi_{x_k}\xleftrightarrow{x_k:}\end{array}\right\}\xleftrightarrow{X:}\mathcal{G}\xleftrightarrow{:\overline{X}}\mathcal{T}_X^{\mathcal{G}\prec\mathcal{H}}\xleftrightarrow{X:}\mathcal{H}\xleftrightarrow{:\overline{X}} \\
&\approx \xleftrightarrow{X:}\left\{\begin{array}{c}\xleftrightarrow{x_1:}\pi_{x_1}\xleftrightarrow{x_1:}\\ \vdots \\ \xleftrightarrow{x_k:}\pi_{x_k}\xleftrightarrow{x_k:}\end{array}\right\}\xleftrightarrow{X:}\left\{\begin{array}{c}\xleftrightarrow{x_1:}\mathcal{G}\xleftrightarrow{:\overline{x_1}}\mathcal{T}_{x_1}^{\mathcal{G}\prec\mathcal{H}}\xleftrightarrow{x_1:}\\ \vdots \\ \xleftrightarrow{x_k:}\mathcal{G}\xleftrightarrow{:\overline{x_k}}\mathcal{T}_{x_k}^{\mathcal{G}\prec\mathcal{H}}\xleftrightarrow{x_k:}\end{array}\right\}\xleftrightarrow{X:}\mathcal{H}\xleftrightarrow{:\overline{X}} \\
&\equiv \xleftrightarrow{X:}\left\{\begin{array}{c}\xleftrightarrow{x_1:}\pi_{x_1}\xleftrightarrow{x_1:}\mathcal{G}\xleftrightarrow{:\overline{x_1}}\mathcal{T}_{x_1}^{\mathcal{G}\prec\mathcal{H}}\xleftrightarrow{x_1:}\\ \vdots \\ \xleftrightarrow{x_k:}\pi_{x_k}\xleftrightarrow{x_k:}\mathcal{G}\xleftrightarrow{:\overline{x_k}}\mathcal{T}_{x_k}^{\mathcal{G}\prec\mathcal{H}}\xleftrightarrow{x_k:}\end{array}\right\}\xleftrightarrow{X:}\mathcal{H}\xleftrightarrow{:\overline{X}} \\
&\approx \xleftrightarrow{X:}\left\{\begin{array}{c}\xleftrightarrow{x_1:}\mathcal{F}\xleftrightarrow{:\overline{x_1}}[\mathcal{S}_{x_1}^{\mathcal{F}\sqsubseteq\mathcal{G}}\xleftrightarrow{:\overline{x_1}}\mathcal{T}_{x_1}^{\mathcal{G}\prec\mathcal{H}}]\xleftrightarrow{x_1:}\\ \vdots \\ \xleftrightarrow{x_k:}\mathcal{F}\xleftrightarrow{:\overline{x_k}}[\mathcal{S}_{x_k}^{\mathcal{F}\sqsubseteq\mathcal{G}}\xleftrightarrow{:\overline{x_k}}\mathcal{T}_{x_k}^{\mathcal{G}\prec\mathcal{H}}]\xleftrightarrow{x_k:}\end{array}\right\}\xleftrightarrow{X:}\mathcal{H}\xleftrightarrow{:\overline{X}}
\end{aligned}$$

The steps in this derivation follow due to the security of the π protocol, splittability of $\mathcal{G}$ with respect to $\mathcal{H}$, simple rearranging/simplification, and the security of the π protocol again (applied k times), respectively.

Thus, the set of machines $\xleftrightarrow{:\overline{X}}\mathcal{S}_X^{\mathcal{F}\sqsubseteq\mathcal{G}}\xleftrightarrow{:\overline{X}}\mathcal{T}_X^{\mathcal{G}\prec\mathcal{H}}\xleftrightarrow{X:}$ satisfies the requirements needed of $\mathcal{T}_X^{\mathcal{F}\prec\mathcal{H}}$ to show that $\mathcal{F} \prec \mathcal{H}$. $\square$

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

Definition 3.5 *$\mathcal{F}$ is called* self-splittable *if $\mathcal{F} \prec \mathcal{F}$. We denote the class of all self-splittable functionalities as* SIMPLECHANNELS.

The class 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 SIMPLECHANNELS.

Theorem 3 $\text{SPLIT}^{\mathcal{F}} = \text{REALIZ}^{\mathcal{F}}$ *for all $\mathcal{F} \in$ SIMPLECHANNELS.*

In other words, functionalities which are realizable using a simple communication channel $\mathcal{F}$ are exactly those which are splittable with respect to $\mathcal{F}$. In fact, we prove the stronger claim that the simple communication channels are *exactly those functionalities for which this characterization holds*. That is, $\text{SIMPLECHANNELS} = \{\mathcal{F} \mid \text{SPLIT}^{\mathcal{F}} = \text{REALIZ}^{\mathcal{F}}\}$. As before, this holds for

PPT systems 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 $\textsc{simplechannels}_u$ but not in $\textsc{simplechannels}_p$.

Proof: ($\Rightarrow$) $\mathcal{F} \prec \mathcal{G}$ unconditionally implies $\mathcal{F} \sqsubseteq \mathcal{G}$ by part 1 of Theorem 2. When $\mathcal{G}$ is self-splittable, substituting $\mathcal{G} = \mathcal{F}$ into part 2 of Theorem 2 gives that $\mathcal{F} \sqsubseteq \mathcal{G}$ implies $\mathcal{F} \prec \mathcal{G}$.

($\Leftarrow$) Suppose $\forall \mathcal{F} : \mathcal{F} \prec \mathcal{G} \iff \mathcal{F} \sqsubseteq \mathcal{G}$. Then take $\mathcal{F} = \mathcal{G}$. Since it is always the case that $\mathcal{G} \sqsubseteq \mathcal{G}$ (via the dummy protocol), we have that $\mathcal{G}$ is self-splittable. □

3.2.4 Relation to the simplified definition.

The simplified definition of splittability (Definition 3.2) was elegant and easy to apply. We would still like to be able to use this simplified definition, as opposed to the more complicated general definition, to say as much as possible about the complexity of functionalities. The following lemma gives us a tool to do just that:

Lemma 4 $\textsc{split}^* = \textsc{split}^{\mathcal{F}_{\mathsf{pvt}}}$ $(= \textsc{realiz}^{\mathcal{F}_{\mathsf{pvt}}})$.

Proof: The $\textsc{split}^* \supseteq \textsc{split}^{\mathcal{F}_{\mathsf{pvt}}}$ direction is trivial by the definition of $\textsc{split}^*$. For the other direction, assume $\mathcal{F} \in \textsc{split}^{\mathcal{G}}$ for some $\mathcal{G}$, witnessed by machines $\{\mathcal{T}_X\}_X$. We must demonstrate a split of $\mathcal{F}$ with respect to $\mathcal{F}_{\mathsf{pvt}}$.

First, define $\widehat{\mathcal{G}}$ as the modification of $\mathcal{G}$ that does not interact on its "adversary port" (in our convention, the communication port indexed by $m+1$ in an m-party functionality).

Modify the machines $\mathcal{T}_X$ to obtain machines $\mathcal{T}'_X$ as follows: Each $\mathcal{T}_X$ is interacting (on the right) with $\mathcal{G}$, and through what is essentially a $\mathcal{F}_{\mathsf{pvt}}$ kind of channel. For $\mathcal{T}_X$ where $1 \in X$, we will subsume a copy of $\widehat{\mathcal{G}}$ into $\mathcal{T}_X$ to obtain $\mathcal{T}'_X$. Now consider $\mathcal{T}_Y$ for $1 \notin Y$. It is expecting to interact on the right with $\mathcal{G}$. Instead, it can interact on the right with $\mathcal{F}_{\mathsf{pvt}}$ as follows: It sends a message on $\mathcal{F}_{\mathsf{pvt}}$ to party 1. Whenever $\mathcal{T}'_X$ receives an input for party 1 on the $\mathcal{F}_{\mathsf{pvt}}$ (or when $\mathcal{T}_X$ outputs something for $\mathcal{G}$), it gives this input to its local copy of $\widehat{\mathcal{G}}$ on behalf of the originating party. Whenever the simulation of $\widehat{\mathcal{G}}$ outputs something for party i, $\mathcal{T}'_X$ sends it via $\mathcal{F}_{\mathsf{pvt}}$ to party i.

In this way, the entire splittability interaction is as before, except the functionality $\mathcal{G}$ is being simulated by one of the $\mathcal{T}'$ machines, and they are actually communicating over a $\mathcal{F}_{\mathsf{pvt}}$ functionality. Thus, these machines witness of split of $\mathcal{F}$ with respect to $\mathcal{F}_{\mathsf{pvt}}$. □

In terms of splittability, we interpret this as indicating that $\mathcal{F}_{\mathsf{pvt}}$ is the "easiest" functionality to split with respect to. In terms of realizability, we interpret it as as indicating that $\mathcal{F}_{\mathsf{pvt}}$ is the most secure channel one can have. The special status of $\mathcal{F}_{\mathsf{pvt}}$ is due to the fact that in our model, the entities communicate with each other using essentially such a channel.

Most importantly, combining Lemma 4 with the characterization of Theorem 3, we see that if $\mathcal{F}$ is unsplittable according to the simplified 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 [18] 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. 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.[4] A *corruption scheme* for m-party functionalities is a set $C \subseteq \{\mathsf{active}, \mathsf{passive}, \mathsf{none}\}^m$. We say that a (static) adversary $\mathcal{A}$ *C-corrupts* (in a protocol π) if the sequence of corruptions γ effected by $\mathcal{A}$ is in C.[5] For example, the normal corruption setting corresponds to $C = \{\mathsf{active}, \mathsf{none}\}^m$ and the honest-but-curious setting corresponds to $C = \{\mathsf{passive}, \mathsf{none}\}^m$.

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 $\{\mathsf{none}, \mathsf{passive}\}^m \cup \{\mathsf{none}, \mathsf{active}\}^m$. Note that the normal and the honest-but-curious corruption schemes are indeed uniform. However, one can (and will, later on) consider a uniform corruption scheme (for a 2-party functionality) like $\{(\mathsf{none}, \mathsf{passive}), (\mathsf{active}, \mathsf{none})\}$ too.

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 $\mathcal{A}$, there exists a C-corrupting $\mathcal{S}$ such that for all environments $\mathcal{Z}$, $\textsc{exec}[\mathcal{Z}, \mathcal{A}, \pi^{\mathcal{G}}] \approx \textsc{exec}[\mathcal{Z}, \mathcal{S}, \partial^{\mathcal{F}}]$.

Note that if $\mathcal{A}$ uses a corruption sequence $\gamma \in C$, then $\mathcal{S}$ must also use γ (because the environment receives the corruption sequence as an input). If $C = \{\mathsf{active}, \mathsf{none}\}^m$, any corruption is considered active. So even if $\mathcal{A}$ chooses not to alter the execution of a corrupt party, $\mathcal{S}$ is free to actively control the party in the ideal-world. On the other hand if C has an element

[4]We say that an adversary *passively corrupts* a party P_i in a protocol π if it receives inputs for P_i from the environment, and runs the protocol π honestly on behalf of P_i in the interaction. This definition is the same both in the real-world (protocol) and the ideal-world interactions (where the protocol considered is the dummy protocol).

[5]The adversary is allowed to choose the corruption sequence $\gamma \in C$, but the environment receives γ as an input.

$\gamma \in \{\mathsf{passive}, \mathsf{none}\}^m$, then when $\mathcal{A}$ uses γ making only passive corruptions, $\mathcal{S}$ must also make only passive corruptions in the ideal world.

Defining Deviation Revealing Functionalities. Informally, deviation revealing means that an environment can detect, *based solely on the functionality's outputs* (as reported by the parties), whether the adversary is actively corrupting the parties. That is, the functionality's outputs will reflect any "deviation" by the adversary from a passive corruption strategy. A simple definition of deviation would be to consider whether an adversary's inputs to the functionality are the "correct" ones, based one its inputs from the environment and the protocol. However, this definition is not robust to slight syntactical changes in the functionality. For instance, the functionality may accept an extra bit in its inputs, which it ignores. Then, deviating from the dummy protocol by changing the last bit of an input is a benign deviation which we would like to not count as a deviation.

This motivates the following definition of deviation-revealing. It is in terms of a *correctness environment* — namely, an environment that does not interact with the adversary at all, except by interacting with parties (some of which are corrupt) to give inputs and receive outputs. Also we refer to the dummy C-corrupting adversary, $\widetilde{\mathcal{A}}$ that, for all the corrupt players acts passively, and does not interact with the environment except through the input/output of the corrupt parties.

Definition 3.6 *A functionality is C-*deviation-revealing *if for all adversaries $\mathcal{A}$: either*

- *there exists a correctness environment $\mathcal{Z}$ such that* $\textsc{exec}[\mathcal{Z}, \mathcal{A}, \partial^{\mathcal{F}}] \not\approx \textsc{exec}[\mathcal{Z}, \widetilde{\mathcal{A}}, \partial^{\mathcal{F}}]$*, where $\widetilde{\mathcal{A}}$ is the dummy C-corrupting adversary;*
- *or, there exists a C-corrupting adversary $\mathcal{A}'$ such that for all environments $\mathcal{Z}$,* $\textsc{exec}[\mathcal{Z}, \mathcal{A}, \partial^{\mathcal{F}}] \approx \textsc{exec}[\mathcal{Z}, \mathcal{A}', \partial^{\mathcal{F}}]$.

In other words, a functionality $\mathcal{F}$ is C-deviation-revealing if for every adversary $\mathcal{A}$ (in the ideal world with $\mathcal{F}$), $\mathcal{A}$ is either "equivalent" to a C-corrupting adversary $\mathcal{A}'$, or an environment which simply checks the outputs of the interaction can detect that $\mathcal{A}$ is not a C-corrupting adversary.

As simple examples, 2-party functionalities $\mathcal{F}_{\mathsf{OT}}$ (oblivious transfer) and $\mathcal{F}_{\mathsf{com}}$ (commitment) are C-deviation-revealing functionalities with $C = \{\mathsf{passive}, \mathsf{none}\}^2$ (i.e., honest-but-curious corruption model). In particular, with a corrupt sender and an honest receiver (either in $\mathcal{F}_{\mathsf{OT}}$ or $\mathcal{F}_{\mathsf{com}}$), if the sender deviates from what the environment instructs it to do, the outputs produced by the receiver will reflect this, provided the environment picks the inputs randomly. That is, a correctness environment can detect the deviation. For $\mathcal{F}_{\mathsf{OT}}$, one should also consider a corrupt receiver: if the receiver deviates, and sends the wrong index to the functionality, it will not be able to output the correct value (with noticeable probability); so, again, an environment which chooses the inputs randomly can detect this deviation.

Also note that for the same corruption scheme, $\mathcal{F}_{\mathsf{OR}}$ (with only Bob receiving the output) *is not deviation revealing*, because a correctness environment cannot detect if a deviating Bob sends 0 to the functionality (and thereby learns more than what a non-deviating Bob could learn).

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.]*

We prove the three statements in Theorem 4 as three lemmas below. We start with the transitivity of the $\sqsubseteq^C$ relation. This is a generalization of the UC theorem to C-corruption schemes.[6] This holds for all corruption schemes C (not just uniform corruption schemes).

Lemma 5 *If $\mathcal{F} \sqsubseteq^C \mathcal{G} \sqsubseteq^C \mathcal{H}$, then $\mathcal{F} \sqsubseteq^C \mathcal{H}$.*

PROOF: The proof mimics the proof of the plain UC theorem, but handling passive corruptions differently. Let π be a protocol witnessing $\mathcal{F} \sqsubseteq^C \mathcal{G}$, and let ρ be a protocol witnessing $\mathcal{G} \sqsubseteq^C \mathcal{H}$. We use π^ρ to denote the natural composition of the two protocols, which does the following: Maintain a separate components for executing each protocol π and ρ. For each message sent out by π, feed it as input to ρ, and for each output of ρ, feed it as a response to π. Let ρ run on $\mathcal{H}$, and let π exchange its inputs and outputs with the environment.

Consider any real-world, C-corrupting adversary $\mathcal{A}$ for the protocol π^ρ. Let M be the set of parties which are only passively corrupted by $\mathcal{A}$, and which $\mathcal{A}$ is not allowed to actively corrupt according to the C corruption scheme. For each of these parties, $\mathcal{A}$ runs the protocol honestly on their behalf, and then computes some information based on the views in the protocol to interact with the environment. Without loss of generality, we can assume that $\mathcal{A}$ is the following kind of dummy adversary: it receives inputs for parties M from the environment, runs the protocol on their behalf, and returns their views in the protocol to the environment. For all other corrupted parties, $\mathcal{A}$ acts as a total dummy adversary (the environment controls all messages sent and received by the adversary on behalf of these parties).

Without loss of generality, $\mathcal{A}$ has the same two separate components for π and ρ, each of which generate a view for their respective protocol. $\mathcal{A}$ receives and assembles the views from these two components into a coherent view for π^ρ.

Now consider an interaction with this $\mathcal{A}$ and arbitrary environment $\mathcal{Z}$. Subsume into the environment the π components of all honest parties, and the components in $\mathcal{A}$ for π and for assembling the two view into one. Call the adversary's remaining external components $\mathcal{A}'$. This remaining adversary is C-corrupting for the ρ protocol, as it honestly runs the ρ protocol on behalf of the M parties and returns their view, and acts as a dummy adversary for the other corrupt parties. By the guarantee of ρ, there is a C-corrupting simulator $\mathcal{S}$ (for the dummy protocol) which achieves the same effect interacting with $\mathcal{G}$.

By the standard UC security guarantee, $\mathcal{S}$ corrupts the same set of parties as $\mathcal{A}'$. $\mathcal{S}$ must only passively corrupt each of the parties in M, by its definition. It is legal according to C for $\mathcal{S}$ to actively corrupt the other parties. Now remove from the environment the components that were subsumed previously. The adversary now runs the π protocol on behalf of parties in M, sends the protocol messages to $\mathcal{S}$, which runs the dummy protocol sending them to $\mathcal{G}$. $\mathcal{S}$ returns a "view" of the ρ protocol, and the adversary assembles the views of both protocols into a view for π^ρ. Because the $\mathcal{S}$ component runs a dummy protocol, the entire adversary as a whole is honestly executing the π protocol on behalf of the parties in M, and thus is C-corrupting for π. Thus, by the guarantee of π, there is a C-corrupting $\mathcal{S}'$ for this adversary. Taking this to be our final simulator, we see that any C-corrupting adversary for π^ρ has a C-corrupting simulator in the $\mathcal{H}$ ideal world, and thus $\mathcal{F} \sqsubseteq^C \mathcal{H}$. □

[6]We remark that the UC theorem has one more implication other than the transitivity of the realizability: it links realizability of $\mathcal{F}^+$ to that of $\mathcal{F}$ (namely, $\mathcal{F} \sqsubseteq \mathcal{G} \implies \mathcal{F}^+ \sqsubseteq \mathcal{G}^+$). However, this *does not* extend to C-corruption schemes, if $\mathcal{F}^+$ is considered to provide multiple instances of $\mathcal{F}$*with different role assignments to the parties in the different instances.* This will not be important to us in our applications, as we will be able to directly consider augmented functionalities when necessary. Note that a converse (namely, $\mathcal{F}^+ \sqsubseteq \mathcal{G}^+ \implies \mathcal{F} \sqsubseteq \mathcal{G}^+$) does extend to C-corruption schemes too.

Next we show that deviation-revealing property does indeed rule out the odd behavior of functionalities like $\mathcal{F}_{\mathsf{OR}}$ which has a simple secure protocol in the normal (active corruption) setting, which is not secure for the honest-but-curious setting (and indeed, has no secure protocol in that setting). However, it is still necessary to restrict ourselves to uniform corruption schemes for this to hold.[7]

Lemma 6 *If $\mathcal{F}$ is C-deviation-revealing for a uniform corruption scheme C, and $\mathcal{F} \sqsubseteq \mathcal{G}$, then $\mathcal{F} \sqsubseteq^C \mathcal{G}$.*

PROOF: Suppose π is a secure realization of $\mathcal{F}$ with respect to $\mathcal{G}$. For any adversary $\mathcal{A}$ with a corruption sequence $\gamma \in C$, we need to show a good simulator $\mathcal{S}$ using γ. For $\gamma \in \{\mathsf{none}, \mathsf{active}\}^m$, we are already given simulators by the (normal) security of π. Since C is a uniform corruption scheme, this leaves us with $\gamma \in \{\mathsf{none}, \mathsf{passive}\}^m$.

Consider any adversary $\mathcal{A}$ effecting the corruption $\gamma \in \{\mathsf{none}, \mathsf{passive}\}^m$, and let $\mathcal{S}$ be the simulator for $\mathcal{A}$ guaranteed by the security of π. (That is, for all environments $\mathcal{Z}$, $\textsc{exec}[\mathcal{Z}, \mathcal{A}, \pi^{\mathcal{G}}] \approx \textsc{exec}[\mathcal{Z}, \mathcal{S}, \partial^{\mathcal{F}}]$.) Also consider the adversary $\mathcal{A}^*$ corrupting no party at all (i.e., with $\gamma^* = \{\mathsf{none}\}^m$) and its corresponding simulator which is a dummy adversary corrupting no parties in the ideal-world execution (called $\widetilde{\mathcal{A}}$ in Definition 3.6). (That is, for all environments $\mathcal{Z}$, $\textsc{exec}[\mathcal{Z}, \mathcal{A}^*, \pi^{\mathcal{G}}] \approx \textsc{exec}[\mathcal{Z}, \widetilde{\mathcal{A}}, \partial^{\mathcal{F}}]$.)

We observe that a *correctness environment* cannot distinguish the real-world interaction involving $\mathcal{A}$ from one with the real-world dummy adversary $\mathcal{A}^*$. That is, if $\mathcal{Z}$ is a correctness environment, then $\textsc{exec}[\mathcal{Z}, \mathcal{A}, \pi^{\mathcal{G}}] \approx \textsc{exec}[\mathcal{Z}, \mathcal{A}^*, \pi^{\mathcal{G}}]$. Hence, $\textsc{exec}[\mathcal{Z}, \mathcal{S}, \partial^{\mathcal{F}}] \approx \textsc{exec}[\mathcal{Z}, \widetilde{\mathcal{A}}, \partial^{\mathcal{F}}]$.

Since $\mathcal{F}$ is C-deviation-revealing, this implies that $\mathcal{S}$ must satisfy the second condition of the definition of C-deviation revealing. Thus there exists a C-deviation-revealing adversary $\mathcal{S}'$ which is indistinguishable from $\mathcal{S}$*for all environments*.

Putting everything together, for every C-corrupting $\mathcal{A}$, there is a C-corrupting $\mathcal{S}'$ such that the real-world interaction with $\mathcal{A}$ is indistinguishable from the ideal-world interaction with $\mathcal{S}'$. Thus $\mathcal{F} \sqsubseteq^C \mathcal{G}$ via the protocol π. □

Deviation-revealing functionalities allow us to leverage results in restricted settings to prove statements about realizability in the malicious network-aware setting, via the following lemma:

Lemma 7 *If $\mathcal{F}$ is C-deviation-revealing for a uniform corruption scheme C, and $\mathcal{F} \not\sqsubseteq^C \mathcal{H}$, $\mathcal{G} \sqsubseteq^C \mathcal{H}$, then $\mathcal{F} \not\sqsubseteq \mathcal{G}$.*

PROOF: This is an easy corollary of the preceding results. Suppose the lemma does not hold; i.e., the conditions in the hypothesis hold, but $\mathcal{F} \sqsubseteq \mathcal{G}$. Then since $\mathcal{F}$ is C-deviation revealing, we must have $\mathcal{F} \sqsubseteq^C \mathcal{G}$. Since also $\mathcal{G} \sqsubseteq^C \mathcal{H}$, and $\sqsubseteq^C$ is transitive, we get $\mathcal{F} \sqsubseteq^C \mathcal{H}$, a contradiction. □

[7]As a simple counter-example where a secure protocol for the normal corruption scheme $C = \{\mathsf{none}, \mathsf{active}\}^m$ is not secure in a different corruption scheme C' consider the following. Let $\mathcal{F}$ be a 2-party protocol with no inputs, and provides Alice with no output and always gives 0 to Bob. Clearly this is a deviation revealing functionality (and indeed, any variation in the definition should still retain this deviation revealing).

Now consider a contrived protocol π for securely realizing $\mathcal{F}$: Bob picks a random k-bit string r (and keeps it locally); Alice sends Bob a string r'; If $r = r'$ Bob outputs 1, else it outputs 0. If C' contains the corruption sequence $(\mathsf{active}, \mathsf{passive})$ (i.e., malicious Alice and honest-but-curious Bob), then π is not a C'-secure realization of $\mathcal{F}$.

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 is by definition independent of the environment's choice, and thus the bit that Bob eventually receives will be wrong with probability 1/2.

Using similar arguments, it is a very easy exercise to check that several other important 2-party functionalities, including *coin-tossing* ($\mathcal{F}_{\mathsf{coin}}$), *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 4, 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 [12, 13]. The impossibility results in [12, 13] were later extended in [29], to the setting where certain "trusted setup" functionalities $\mathcal{F}$ are also available for protocols to use. These extensions can also be shown 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 4.1 *$\mathcal{F}$ is a* 2-party secure function evaluation (SFE) *functionality if it waits for inputs x and y from the two parties, respectively, computes two deterministic 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.

Definition 4.2 *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 $f_1(x, y) = f_1'(x)$ for some function f_1', or $f_2(x, y) = f_2'(y)$ for some function f_2'. Otherwise $\mathcal{F}$ has* bidirectional influence.

Definition 4.3 *Let $\mathcal{F} = (f_1, f_2)$ be a 2-party SFE functionality with unidirectional influence; say, the first party's output does not depend on the second party's input. We say that $\mathcal{F}$ is* negligibly hiding *if there exists machines R_1, R_2 such that:*

$$\forall x, y : \mathbf{Pr}\Big[(y^*, s) \leftarrow R_1; f_2\Big(R_2\big(s, f_2(x, y^*)\big), y\Big) \neq f_2(x, y)\Big] \text{ is negligible}$$

The probability is over the randomness of R_1 and R_2.

For functionalities $\mathcal{F}$ with input domains of polynomial size (in the security parameter), negligibly hiding is a simple combinatorial property: $\mathcal{F}$ is negligibly hiding if and only if there exists y such that $f_2(x, y) = f_2(x', y) \implies f_2(x, \cdot) \equiv f_2(x', \cdot)$ for all x, x'.

Note that negligible hiding is essentially a property of f_2. It implies that it is possible to generate an input y^* for f_2 in such a way that given $f_2(x, y^*)$, one can compute a value x^* which is "equivalent" to x (equivalent in that $f_2(x^*, y) = f_2(x, y)$ with high probability). This definition succinctly incorporates both the *completely revealing* and *efficiently invertible* properties of [12].

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: ($\Leftarrow$) Suppose $\mathcal{F}$ has unidirectional influence and is negligibly hiding. Without loss of generality, suppose the first party's output is not affected by the second party's input. Then the following protocol is secure: The first party gives its input to a local copy of $\mathcal{F}$, and simulates the interaction of $\mathcal{F}$ and R (from the definition of negligibly hiding) to compute a value x which is equivalent to its own input. It then sends this value to the second party, and locally computes its own output (as this does not depend on the second party's input). The second party receives x and simulates $\mathcal{F}$ on this input and its own, to obtain its output.

The simulator for a corrupt first party simply sends the value x from the protocol to the functionality. By the correctness of R, this value x will induce the correct output for the other party with overwhelming probability. The simulator for a corrupt second party is simply the machine R from the definition of negligibly hiding. Because the sender in the prescribed protocol generates the message x by interacting with a local copy of $\mathcal{F}$, the output of the simulator is a perfect simulation of the protocol.

($\Rightarrow$) Suppose $\mathcal{F}$ is realizable. Then by Theorem 1, $\mathcal{F}$ is splittable. Let $\mathcal{T}$ be as in the definition of splittable.

Suppose for contradiction that $\mathcal{F}$ has bidirectional influence. Then there exist $x_0, x_1, x_3 \in D_1$ and $y_0, y_1, y_2 \in D_2$ such that

$$f_1(x_0, y_1) \neq f_1(x_0, y_2) \text{ and } f_2(x_1, y_0) \neq f_2(x_2, y_0)$$

Consider an environment $\mathcal{Z}$ which chooses random $i, j \leftarrow \{0, 1, 2\}$, sends x_i and y_j to the two parties respectively, and outputs 1 if the parties respond with $f_1(x_i, y_j)$ and $f_2(x_i, y_j)$, respectively ($\mathcal{Z}$ does not talk to any adversary). When interacting with the original functionality, it outputs 1 with probability 1. Consider what happens when it interacts with the split functionality in which $\mathcal{T}$ interacts with two independent copies of $\mathcal{F}$. Without loss of generality, suppose $\mathcal{T}$ sends an input first to $\mathcal{F}_L$, the functionality which receives input from the first party. Call y^* the input it sends. Its choice of y^* is independent of the environment's choices of i and j, as the non-reactive functionalities $\mathcal{F}_L$ and $\mathcal{F}_R$ give no output until they receive both inputs. The probability that the environment outputs 0 is at least:

$$\mathbf{Pr}_{i,j}[f_1(x_i, y^*) \neq f_1(x_i, y_j)] \geq \mathbf{Pr}_{j \leftarrow \{1,2\}}[f_1(x_0, y^*) \neq f_1(x_0, y_j)] \cdot \mathbf{Pr}[i = 0 \wedge j \neq 0] \geq \frac{1}{2} \cdot \frac{1}{3} \cdot \frac{2}{3} = \frac{1}{9}$$

Thus, $\mathcal{Z}$ distinguishes between the split and unsplit functionalities, contradicting Theorem 1. We conclude that $\mathcal{F}$ has unidirectional influence.

Without loss of generality, let the influence of $\mathcal{F}$ be from the first party to the second party. Now suppose for contradiction that $\mathcal{F}$ is not negligibly hiding.

Consider a class of environments $\mathcal{Z}_{x,y}$ which send x and y to the respective two parties and output 1 if the parties respectively respond with $f_1(x, y)$ and $f_2(x, y)$ (and ignore the adversary). Each of these environments outputs 1 with probability 1 when interacting with the unsplit functionality. Again by Theorem 1, for each of these environments, $\mathcal{F}$ is indistinguishable from the equivalent split functionality in which a machine $\mathcal{T}$ interacts with two independent copies of $\mathcal{F}$.

We now consider the output of the second party. Without loss of generality, suppose that $\mathcal{T}$ sends an input to $\mathcal{F}_L$ before interacting with $\mathcal{F}_R$ (if $\mathcal{T}$ interacts with $\mathcal{F}_R$ first, the same output can be achieved with respect to the second player's output by an $\mathcal{T}$ which does interact with $\mathcal{F}_L$ first and ignores the result). Now $\mathcal{T}$ must be a machine as in the definition of negligibly hiding; it computes an input y^* for $\mathcal{F}_L$ (independently of the environment $\mathcal{Z}_{x,y}$) and uses the resulting output to compute a value x^* (that it sends as input to $\mathcal{F}_R$) such that $f_2(x, y) = f_2(x^*, y)$. Since $\mathcal{F}$ is not negligibly hiding, this condition fails with noticeable probability for some x, y, and the corresponding environment $\mathcal{Z}_{x,y}$ rejects with noticeable probability. Again, this contradicts our assumption, so we conclude that $\mathcal{F}$ is negligibly hiding. □

Again, we reiterate that Theorem 5 also characterizes the existence of *non-trivial 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 4.2 to multi-party SFE: If in $\mathcal{F}$ there is a fixed setting of inputs for parties other than i, such that there exist two inputs for party i which induce different outputs for party $j \neq i$, then we say party i *influences* 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.*

PROOF: Suppose the graph induced by $\overset{\mathcal{F}}{\rightsquigarrow}$ has two edges ij and $i'j'$, where $i \neq i'$ and $j \neq j'$. Then any bipartition of $[m]$ which separates $\{i, j'\}$ and $\{i', j\}$ induces a 2-party functionality which has bidirectional influence. Thus $\mathcal{F}$ cannot be realizable. □

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 give a partial characterization of realizable functionalities in the class 3REGULAR, the 3-party analog of 2REGULAR (namely, 3-party functionalities which do not interact with the adversary, and whose behavior does not depend on which parties are corrupted). We restrict our attention to realizability with respect to completely private channels. We show that for functionalities in 3REGULAR that have a secure *honest-majority protocol*,[8] 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 on $\mathcal{F}_{\mathsf{pvt}}$, then $\mathcal{F}$ is UC-realizable using $\mathcal{F}_{\mathsf{pvt}}$ if and only if all 2-party restrictions of $\mathcal{F}$ are UC-realizable using $\mathcal{F}_{\mathsf{pvt}}$.*

In particular, this implies that realizable 3-party SFE functionalities have a simple combinatorial characterization analogous to Theorem 5.

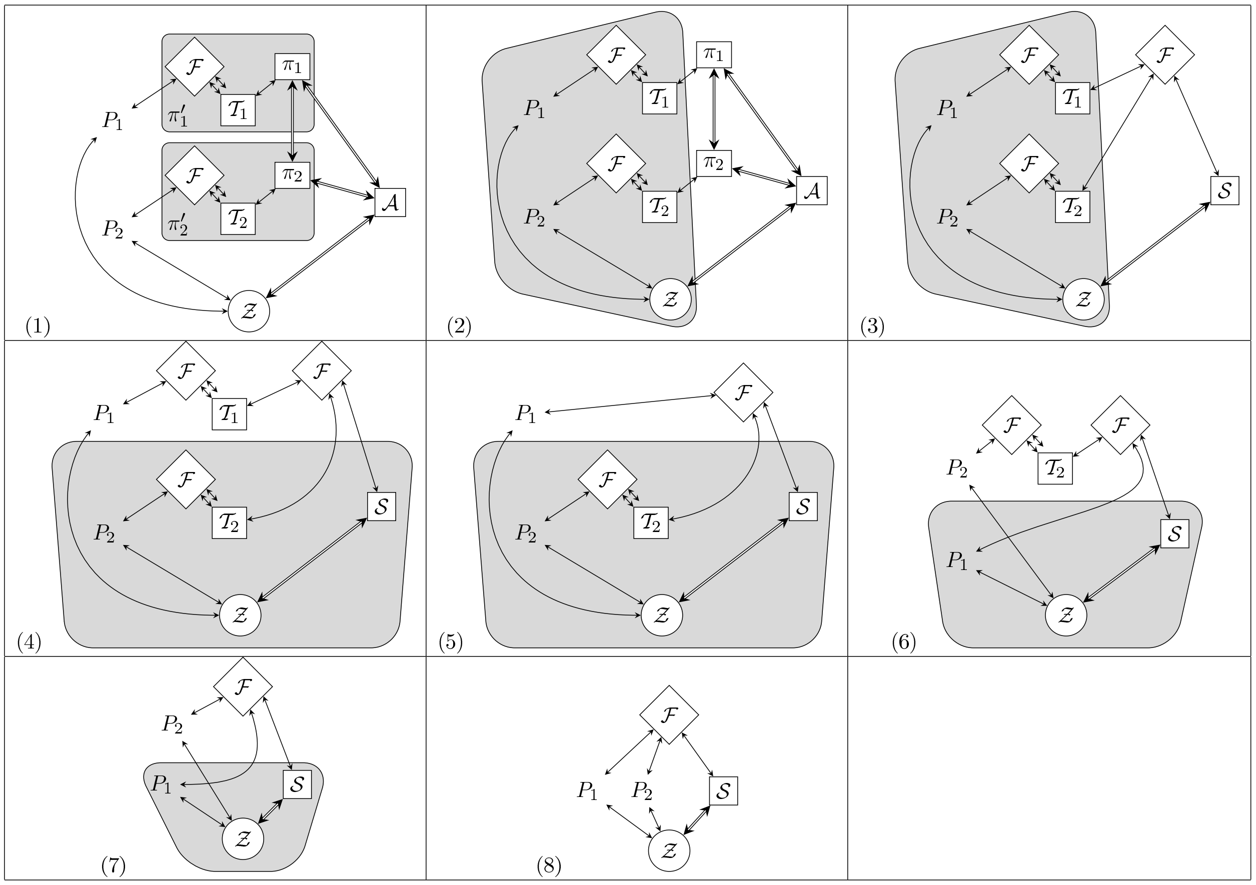

Figure 6: Steps in the proof of Theorem 7, when one party is corrupted.

PROOF: The forward implication is trivially true. To show the other direction, assume each 2-party restriction is realizable (and therefore splittable according to the simplified definition Definition 3.2). Let $\mathcal{T}_i$ be the machine guaranteed by splittability on the 2-party restriction induced by the partition $\{i\}, (\{1,2,3\} \setminus \{i\})$. Let π be the honest-majority protocol for $\mathcal{F}$.

Now the protocol π' we construct is as follows: Party i simulates $\mathcal{F}$, $\mathcal{T}_i$, and π_i. He feeds his inputs to $\mathcal{F}$ (as party i), and simulates the interaction of $\mathcal{F}$ and $\mathcal{T}_i$ as in the splittability interaction

[8]Honest majority protocols are known to exist for essentially all SFE functionalities if modified to let corrupt parties block outputs to other players. This follows by adapting the well-known information theoretically secure (stand-alone) protocols [15, 6, 40] (see [8]), and replacing their use of broadcast channel by a UC-secure protocol as suggested in [24]. Note that the modified functionalities, though no more SFEs (as the adversary may decide on blocking, based on the outcome of the function), continue to be in 3REGULAR, because the adversary can block output delivery only by corrupting at least one party.

($\mathcal{T}_i$ interacts with $\mathcal{F}$ as the other two parties). Whenever $\mathcal{T}_i$ would output something (it expects to interact with another functionality as party i), we input it to π_i running on a private channel with other parties.

To show that the protocol π' is secure, we must demonstrate a simulator for each (dummy) adversary. The cases when the adversary corrupts all or no parties are trivial. We focus on the case when the adversary corrupts 1 or 2 parties.

Suppose the adversary corrupts 2 parties (without loss of generality, parties 2 and 3). This adversary is interacting on the private channel on behalf of parties 2 and 3, while the honest party 1 is running π'. Let us compose $\mathcal{T}_1$, π_1, and the private channel in the same way that they interact within the π' protocol, and use that as the simulator. Then in the ideal world, party 1 interacts directly with $\mathcal{F}$; $\mathcal{T}_1$ inside the simulator interacts with $\mathcal{F}$ as parties 2 and 3; $\mathcal{T}_1$'s additional outputs are passed to π_1. This is exactly the same interaction as the real-world, only with the individual components bundled together differently. Thus the ideal-world interaction with this simulator is indistinguishable from the real-world interaction.

Now suppose the adversary corrupts 1 party, without loss of generality, party 3 (see Figure 6). In the real-world interaction, parties 1 and 2 are each simulating a (different) copy of $\mathcal{F}$, as well as their respective $\mathcal{T}_i$ and π_i (box 1 in Figure 6). Consider subsuming both copies of $\mathcal{F}$, and the $\mathcal{T}_i$'s into the environment (box 2 in Figure 6). What remains is an interaction wherein parties 1 and 2 are running π_1 and π_2, respectively, on the private channels. This is an honest majority, and the security of π holds. Thus there is a simulator $\mathcal{S}$ for the adversary so that this interaction is indistinguishable from an ideal-world interaction with $\mathcal{F}$ itself (box 3 in Figure 6). Now, take party 1's copy of $\mathcal{F}$ and $\mathcal{T}_1$ out of the environment, and place $\mathcal{S}$ inside the environment (box 4 in Figure 6). Now all that is outside the environment is an interaction with two copies of $\mathcal{F}$, with $\mathcal{T}_1$ coordinating between them. This is exactly the splitting of the 2-party restriction of $\mathcal{F}$ induced by $\{1\}, \{2, 3\}$. Thus, this interaction is indistinguishable from the interaction with a single $\mathcal{F}$ alone (box 5 in Figure 6).

Finally, take party 2's copy of $\mathcal{F}$ and $\mathcal{T}_2$ out of the environment (box 6 in Figure 6). Again, what remains is exactly the splitting of another 2-party restriction of $\mathcal{F}$. We can similarly replace this split version of $\mathcal{F}$ by a copy of $\mathcal{F}$ itself (box 7 in Figure 6). The last step is to remove the simulator $\mathcal{S}$ from the environment. What remains is now the original environment, in an ideal-world interaction with $\mathcal{F}$ and simulator $\mathcal{S}$ (box 8 in Figure 6). By construction, this interaction is indistinguishable from the real-world interaction. □

Note that our protocol requires each player to indirectly 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 recently introduced and explored in [27, 25]. There, significant efficiency gains are achieved in the standard corruption model by leveraging MPC protocols with security in the 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 [16].

As a result of independent interest, in Appendix B we present simpler direct protocols for certain important non-trivial cases without using generic honest-majority protocols.

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 $\text{REALIZ}_u^{\mathcal{F}_{\mathsf{pvt}}} \subsetneq \text{REALIZ}_u^{\mathcal{F}^+_{\mathsf{simex}}} \subsetneq \text{REALIZ}_u^{\mathcal{F}^+_{\mathsf{com}}} \subsetneq \text{REALIZ}_u^{\mathcal{F}^+_{\mathsf{OT}}}$.

Here, $\mathcal{F}^+_{\mathsf{simex}}$, $\mathcal{F}^+_{\mathsf{com}}$, and $\mathcal{F}^+_{\mathsf{OT}}$ denote "augmented" versions of simultaneous exchange,[9] bit commitment, and oblivious transfer, respectively, in which the functionality provides multiple "sessions" instead of just one.

Proof: We note that $\mathcal{F}_{\mathsf{pvt}} \sqsubseteq_u \mathcal{F}^+_{\mathsf{simex}} \sqsubseteq_u \mathcal{F}^+_{\mathsf{com}} \sqsubseteq_u \mathcal{F}^+_{\mathsf{OT}}$ (these hold due to straight-forward or well-known protocols, omitted here for brevity), which, along with the UC theorem, implies that $\text{REALIZ}_u^{\mathcal{F}_{\mathsf{pvt}}} \subseteq \text{REALIZ}_u^{\mathcal{F}^+_{\mathsf{simex}}} \subseteq \text{REALIZ}_u^{\mathcal{F}^+_{\mathsf{com}}} \subseteq \text{REALIZ}_u^{\mathcal{F}^+_{\mathsf{OT}}}$.

To see that $\text{REALIZ}_u^{\mathcal{F}_{\mathsf{pvt}}} \neq \text{REALIZ}_u^{\mathcal{F}^+_{\mathsf{simex}}}$ we note that $\mathcal{F}_{\mathsf{simex}}$ has bidirectional influence and hence $\mathcal{F}_{\mathsf{simex}} \notin \text{REALIZ}_u^{\mathcal{F}_{\mathsf{pvt}}}$ (but $\mathcal{F}_{\mathsf{simex}} \in \text{REALIZ}_u^{\mathcal{F}^+_{\mathsf{simex}}}$).

It remains to show $\text{REALIZ}_u^{\mathcal{F}^+_{\mathsf{simex}}} \neq \text{REALIZ}_u^{\mathcal{F}^+_{\mathsf{com}}} \neq \text{REALIZ}_u^{\mathcal{F}^+_{\mathsf{OT}}}$. For this we will show that $\mathcal{F}_{\mathsf{com}} \not\sqsubseteq_u \mathcal{F}^+_{\mathsf{simex}}$ and $\mathcal{F}_{\mathsf{OT}} \not\sqsubseteq_u \mathcal{F}^+_{\mathsf{com}}$.

$\mathcal{F}_{\mathsf{OT}} \not\sqsubseteq_u \mathcal{F}^+_{\mathsf{com}}$ follows by applying Theorem 4 with $C = \{\mathsf{passive}, \mathsf{none}\} \times \{\mathsf{passive}, \mathsf{none}\}$. There is a trivial protocol that C-realizes $\mathcal{F}_{\mathsf{com}}$ using $\mathcal{F}_{\mathsf{pvt}}$ (the sender sends the string "commit", then later reveals the committed bit), while $\mathcal{F}_{\mathsf{OT}}$ does not have a C-realization on $\mathcal{F}_{\mathsf{pvt}}$ for unbounded systems [32] Also, it is easily shown that $\mathcal{F}_{\mathsf{OT}}$ is C-deviation-revealing, by considering an environment which chooses random inputs for $\mathcal{F}_{\mathsf{OT}}$. Whenever an ideal-world adversary does not send the correct inputs to $\mathcal{F}_{\mathsf{OT}}$ (i.e., deviates from being C-corrupting), with constant probability, its view will be independent of the correct outputs.

$\mathcal{F}_{\mathsf{com}} \not\sqsubseteq_u \mathcal{F}^+_{\mathsf{simex}}$ follows by a similar argument, considering $C = (\{\mathsf{active}, \mathsf{none}\} \times \{\mathsf{passive}, \mathsf{none}\}) \cup (\{\mathsf{none}, \mathsf{active}\} \times \{\mathsf{none}, \mathsf{passive}\})$. Under this corruption scheme, at most one party may be corrupted (either actively or passively) For unbounded systems, there is no $\mathcal{F}_{\mathsf{com}}$ protocol using $\mathcal{F}_{\mathsf{pvt}}$ in this corruption scheme: this follows from the information-theoretic impossibility of a commitment scheme which is both statistically binding and statistically hiding. However, there is a $\mathcal{F}^+_{\mathsf{simex}}$ protocol when one of the parties is guaranteed to be only passively corrupted (because this party can receive the other party's input and then honestly send its own input to the other party), so Theorem 4 applies. □

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 [14]. We leave it as an open problem in the complexity of functionalities, whether there is a zero-one law (i.e., whether all functionalities not in $\text{REALIZ}^{\mathcal{F}_{\mathsf{pvt}}}$ are complete).

4.5 Completeness for Uniform Corruption Schemes

The deviation revealing property of $\mathcal{F}_{\mathsf{OT}}$ has an interesting consequence, because $\mathcal{F}_{\mathsf{OT}}$ is a complete functionality. Note that $\mathcal{F}_{\mathsf{OT}}$ was already known to be a complete functionality for passive

[9]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.

corruption schemes [23] before it was shown to be complete (for asymmetric SFE functionalities) for active corruption as well [30]. But we observe that the security of the protocol in [30] extends to the passive setting as well. (This follows from the nature of the simulator described in [30]: it sends the correct inputs to the functionality, when the players are passively corrupted.) Then, in fact, the completeness result in [30] extends to *all uniform corruption schemes* C (using the same protocol, and choosing a simulator for each corruption sequence $\gamma \in C$ depending on whether γ involves only passive corruptions or only active corruptions).

By Theorem 4 (2a), and the fact that $\mathcal{F}_{\mathsf{OT}}$ is C-deviation revealing for any uniform corruption scheme C (and the universal composition theorems), we have the following result.

Theorem 9 *$\mathcal{F}$ is complete if and only if $\mathcal{F}$ is complete for all uniform corruption schemes.*

Note that "completeness" refers to completeness in the sense of [30], wherein only active corruptions are considered.

5 Open Problems and Future Directions

Using the notion of splittability we addressed what is perhaps the most basic and important complexity class, namely $\text{REALIZ}^{\mathcal{F}_{\mathsf{pvt}}}$. However, to understand the nature of cryptographic complexity, several other (higher complexity) classes need to be understood. We collect several important problems below.

Completeness. After the class of functionalities realizable with standard communication channels, perhaps the next important class is the class of all *complete* functionalities. Oblivious transfer and several related functionalities have been shown to belong to this class [30, 7, 19].[10] However there is no known structural characterization for arbitrary complete MPC functionalities.[11] Such a characterization based on black-box structural properties will be delicate, as completeness is quite different between the computationally unbounded and PPT systems. (For instance, coin-tossing is complete for PPT systems [14], but not for unbounded systems (Theorem 8).)

Zero-One Law for PPT Systems. We ask whether there is a zero-one law of complexity for PPT systems; namely, that all unsplittable (with respect to private channels) functionalities are in fact complete. The archetypal unsplittable functionalities of oblivious transfer (not negligibly hiding) and simultaneous exchange (bidirectional influence) are both complete for PPT systems. Indeed, even the arguably simplest (randomized) unsplittable functionality of coin-tossing is complete for PPT systems. However some other unsplittable SFE functionalities (such as symmetric boolean-OR) do not immediately seem to yield coin-tossing. Resolving the zero-one question requires determining the complexity of these functionalities.

Incomparable Classes. While the major classes we considered in this work formed a hierarchy, it is likely that there are incomparable degrees of cryptographic complexity (that is, functionalities $\mathcal{F}$ and $\mathcal{G}$ such that $\mathcal{F} \not\sqsubseteq \mathcal{G}$ and $\mathcal{G} \not\sqsubseteq \mathcal{F}$). An important problem is to understand if such distinct cryptographic qualities exist, and if so, can they all be characterized.

[10]These results were not derived in the UC framework. However the reductions are often easily verified to carry over to the UC framework.

[11]For the computationally unbounded setting, one of the broadest results about completeness is by Kilian [32], which considers only 2-party SFE functionalities which give output to one party.

Multi-Party functionalities. Even for SFE functionalities, our understanding of splittability is not complete when it comes to functionalities involving more than two parties. Indeed, our results already significantly narrow down the uncertainty, but we leave it as an open problem to give a complete combinatorial characterization of which SFE functionalities (wrapped to allow adversarial blocking) are realizable in general.

Identifying new classes and techniques guided by specific MPC problems. Understanding qualitative cryptographic properties is guided by considering several primitives (like oblivious transfer, commitment and coin-tossing) that have proven important over the years. We do not however know how realizability classes of some other functionalities relate to the ones we have studied. Symmetric OR functionality (which gives the output to both Alice and Bob) is an important example. Another question is to identify structural properties that can generalize separations between specific functionalities. For instance an ad hoc argument can be used to show that (in computationally unbounded systems) coin-tossing (augmented with a private channel) cannot be used to achieve simultaneous exchange. However the tools we presented in this work cannot be used to derive this separation.

Complexity as revealed in other corruption settings. The standard corruption setting of the UC framework (which we also adopt in this work) is not the most general corruption notion. Indeed we already used more general corruption schemes involving passive corruption for deriving some separations. Nevertheless, there is more to be learned about complexity of functionalities by studying their realizability in different corruption settings. New techniques and notions of reductions will be required to derive structural results in such settings, and to translate these to other corruption settings.

A complete map of all interesting cryptographic properties. All the above problems are aimed at mapping out different cryptographic properties of various functionalities. The ambitious goal is to understand all atomic cryptographic properties that can be combined to explain the complexity of any functionality. Needless to say, our techniques and results can only be considered a small part of such a project.

Quantitative Complexity. In furthering a theory of cryptographic complexity it is necessary that qualitative complexity notions are refined using quantitative measures. There are at least two aspects to this: firstly, when the cryptographic complexity of a functionality is *quantitatively diluted* beyond a threshold, it may change its qualitative behavior. (For instance, results in [19] show this for a generalization of Rabin-OT with error, and a threshold on the error.) Another aspect is to understand the optimality of various protocols: for instance one could ask about the "capacity" of one functionality $\mathcal{G}$ for another functionality $\mathcal{F}$ ($\mathcal{F}$-capacity of $\mathcal{G}$): i.e., how many "units" of $\mathcal{F}$ are needed to realize one "unit" of $\mathcal{G}$ (possibly in terms of an asymptotic rate). (Here $\mathcal{F}$ and $\mathcal{G}$ may be composite functionalities, with possibly a vector unit, rather than a scalar.) For some pairs of functionalities this problem has already been looked into in recent literature (though not posed in such generality as we do here).

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 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.

[3] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proc. 28th STOC*, pages 479–488. ACM, 1996.

[4] A. Beimel and T. Malkin. A quantitative approach to reductions in secure computation. In M. Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 238–257. Springer, 2004.

[5] 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.

[6] 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.

[7] C. Cachin. On the foundations of oblivious transfer. In K. Nyberg, editor, *EUROCRYPT*, volume 1403 of *Lecture Notes in Computer Science*, pages 361–374. Springer, 1998.

[8] 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.

[9] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Revised version of [8].

[10] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *TCC*, 2007.

[11] R. Canetti and M. Fischlin. Universally composable commitments. Report 2001/055, Cryptology ePrint Archive, July 2001. Extended abstract appeared in CRYPTO 2001.

[12] 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.

[13] 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.

[14] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party computation. In *Proc. 34th STOC*, pages 494–503. ACM, 2002.

[15] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th STOC*, pages 11–19. ACM, 1988.

[16] B. Chor and Y. Ishai. On privacy and partition arguments. *Information and Computation*, 167(1):2–9, 2001.

[17] B. Chor and E. Kushilevitz. A zero-one law for boolean privacy (extended abstract). In *STOC*, pages 62–72. ACM, 1989.

[18] B. Chor and E. Kushilevitz. A zero-one law for boolean privacy. *SIAM J. Discrete Math.*, 4(1):36–47, 1991.

[19] I. Damgård, J. Kilian, and L. Salvail. On the (im)possibility of basing oblivious transfer and bit commitment on weakened security assumptions. In J. Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 1999.

[20] C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[21] O. Goldreich. *Foundations of Cryptography: Basic Applications.* Cambridge University Press, 2004.

[22] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In ACM, editor, *Proc.* 19*th STOC*, pages 218–229. ACM, 1987. See [21, Chap. 7] for more details.

[23] O. Goldreich and R. Vainish. How to solve any protocol problem - an efficiency improvement. In C. Pomerance, editor, *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 1987.

[24] S. Goldwasser and Y. Lindell. Secure computation without agreement. In *DISC*, volume 2508 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2002.

[25] D. Harnik, Y. Ishai, E. Kushilevitz, and J. B. Nielsen. Ot-combiners via secure computation. To appear in TCC 2008, 2008.

[26] 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.

[27] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30. ACM, 2007.

[28] 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.

[29] 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`.

[30] J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31. ACM, 1988.

[31] J. Kilian. A general completeness theorem for two-party games. In *STOC*, pages 553–560. ACM, 1991.

[32] J. Kilian. More general completeness theorems for secure two-party computation. In *Proc.* 32*th STOC*, pages 316–324. ACM, 2000.

[33] J. Kilian, E. Kushilevitz, S. Micali, and R. Ostrovsky. Reducibility and completeness in private computations. *SIAM J. Comput.*, 29(4):1189–1208, 2000.

[34] E. Kushilevitz. Privacy and communication complexity. In *FOCS*, pages 416–421. IEEE, 1989.

[35] E. Kushilevitz. Privacy and communication complexity. *SIAM J. Discrete Math.*, 5(2):273–284, 1992.

[36] E. Kushilevitz, S. Micali, and R. Ostrovsky. Reducibility and completeness in multi-party private computations. In *FOCS*, pages 478–489. IEEE, 1994.

[37] 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.

[38] M. Prabhakaran. *New Notions of Security.* PhD thesis, Department of Computer Science, Princeton University, 2005.

[39] M. Prabhakaran and A. Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC*, pages 242–251. ACM, 2004.

[40] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st STOC*, pages 73–85. ACM, 1989.

[41] A. C. Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pages 162–167. IEEE, 1986.

A Modeling Conventions

Some of the conventions in our model of network-aware security slightly differ from definitions in previous literature. We believe these conventions add clarity to out discussion of the complexity of functionalities. Below we explain the important differences.

Here we briefly summarize one formalization of the system model used for network-aware security. For a more detailed description we refer the reader to [38]. However, much of the specific modeling details are not important for the theory and the results we have presented here.

By the Network, we refer to the entire system of computation and communication. There are four kinds of entities in a Network: the parties (programs or protocols), the environment, the adversary and the functionalities. Each entity is modeled as a computable function which operates on its inputs (incoming messages) and internal state, and produces outputs (outgoing messages) and a new modified state. The input and output ports of the entities are statically connected up. The environment is connected to all the parties and the adversary. In addition the parties and the adversaries can be connected to functionalities.[12]

Formally a program $\wp$ is a collection of variables. This includes $\mathsf{internal}_\wp$, which is the "internal state" of the program (hidden from $\mathcal{A}$, $\mathcal{Z}$ and other programs in the Network). It communicates with $\mathcal{Z}$ and functionalities through input/output messages, which (for the purposes of this section) are denoted by variables $\mathsf{mesg}_{\mathcal{Z}\to\wp}$, $\mathsf{mesg}_{\wp\to\mathcal{G}}$, etc.

Behavior of a party $\wp$ is determined by its program (also denoted by $\wp$, abusing the notation), its inputs and internal state. The execution of the program is modeled as applying the (computable) function $\wp$ to its input variables $\mathsf{input}_\wp$ (which includes messages it receives, like $\mathsf{mesg}_{\mathcal{Z}\to\wp}$) and internal state variables $\mathsf{internal}_\wp$ (including the internal randomness). It produces new values for the internal state variables $\mathsf{internal}_\wp$ and for the output variables $\mathsf{output}_\wp$ (which includes messages it sends, like $\mathsf{mesg}_{\wp\to\mathcal{Z}}$). We write it as follows.

$$(\mathsf{internal}_\wp, \mathsf{output}_\wp) \leftarrow \wp(\mathsf{input}_\wp, \mathsf{internal}_\wp).$$

Since a program would typically read its inputs multiple times and produce output (or communications) more than once, we allow the the above updating to be invoked multiple times. As explained below, each of these invocations will be scheduled by the environment as a two-step process. An (instance of) m-party protocol π is defined by (instances of) its constituent programs $(\pi_1, \dots, \pi_m)$. The environment, adversary and functionalities have their own programs. In addition, the adversary includes programs for each of the corrupted parties (in case of malicious corruption), and the internal variables of the adversary includes the variables belonging to the corrupted parties as well. (We consider the special case of passive corruption as well, wherein the the program of the corrupted is the same as the one prescribed by the protocol, and the adversary does not update the program's variables by itself.)

The system execution is best described in terms of the environment's actions: $\mathcal{Z}$ repeatedly reads its variables (the ones it is allowed to read) and updates its internal variables and communication variables (the ones it is allowed to update), until it terminates with an output.[13] In addition, at each step it can invoke the computation of a program $\wp$ (existing or new), by outputting one of the two "commands": $\mathsf{startrun}_\wp$, when the $(\mathsf{input}_\wp, \mathsf{internal}_\wp)$ are read in, and $\mathsf{finisrun}_\wp$ when $(\mathsf{internal}_\wp, \mathsf{output}_\wp)$ are updated (according to the output of the function $\wp$ applied to the inputs read in the last time $\mathsf{startrun}_\wp$ was issued). The duration between the two corresponds to the time the program *runs*. For simplicity we restrict that at each step $\mathcal{Z}$ is allowed to issue at most one

[12]Communication channels that are traditionally built into the network model are also modeled as functionalities.

[13]For the purpose of defining security, we can consider the final output from the environment to be a single bit.

startrun or finisrun command. However, to have the effect of parallel execution, $\mathcal{Z}$ can issue multiple startrun commands (for different programs) before invoking the corresponding finisrun commands.[14]

Unlike the protocol programs, the functionalities and the adversary are not explicitly scheduled by $\mathcal{Z}$, but independently invoked as follows: whenever an entity outputs a message for a virtual entity, the function corresponding to latter is applied immediately, and its outputs and internal state are immediately updated with the results. (If there are messages to more than one virtual entity in a single output, all of them are invoked simultaneously and after that all their outputs and internal states are updated simultaneously.) [15]

Emphasizing the generality of our theory, we do not specify any computational limitations on the functions defining the programs of the machines, but allow abstract classes of *admissible machines*. An important requirement of a class of admissible machines is that it should be closed under composition: that is a machine internally composed of multiple admissible machines remains admissible.[16] The two main types of systems we consider are *unbounded systems*, which admit all probabilistic, resource unbounded machines; and *PPT systems*, which admit all probabilistic, polynomial-time bounded machines. We note that our results hold for both network systems, when the various definitions are understood to refer to admissible machines for the particular system.

Secure Realization. In an interaction among environment $\mathcal{Z}$, adversary $\mathcal{A}$, and parties executing a protocol π that uses functionality $\mathcal{F}$, we use $\text{EXEC}[\mathcal{Z}, \mathcal{A}, \pi^{\mathcal{F}}]$ to denote the distribution of the environment's (single-bit) output. We say that π is a *secure realization* of $\mathcal{F}$ with respect to $\mathcal{G}$, if for all adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$, $\text{EXEC}[\mathcal{Z}, \mathcal{A}, \pi^{\mathcal{G}}] \approx \text{EXEC}[\mathcal{Z}, \mathcal{S}, \partial^{\mathcal{F}}]$ (that is, the distributions differ only negligibly), where ∂ denotes the "dummy protocol" that simply relays messages between the functionality and environment. Using more prevalent terminology, π is a secure realization of $\mathcal{F}$ in the $\mathcal{G}$-hybrid world. We view $\mathcal{G}$ as the "channel" over which the protocol executes. Instead of an arbitrary external adversary in real-world interactions, it suffices (by incorporating the adversary machine into the environment itself) to consider a "dummy" adversary that simply relays messages into and out of the environment. We also only consider adversaries which corrupt parties non-adaptively.

When there is a secure realization of $\mathcal{F}$ with respect to $\mathcal{G}$, we write $\mathcal{F} \sqsubseteq \mathcal{G}$. We denote by $\text{REALIZ}^{\mathcal{G}}$ the class of functionalities which have secure realizations with respect to $\mathcal{G}$. Our main results apply to both PPT and unbounded systems in a unified way. To explicitly refer to realizability in PPT or unbounded systems, we write $\sqsubseteq_p$, REALIZ_p and $\sqsubseteq_u$, REALIZ_u, respectively.

[14]This follows the model used in [38]. In other models of network-aware security that appeared in the literature, the executions of the programs were explicitly sequentialized. In our vocabulary, this translates to the restricting that after a program $\wp$ is started by issuing a scheduling directive $\mathsf{startrun}_\wp$, the next scheduling directive must be $\mathsf{finisrun}_\wp$. That is to say, the *same program* $\wp$ must be finished before another program can be started. Clearly, the new approach here models the real-life scenario more directly, by allowing different program runs to overlap with each other.

[15]The immediate invocation of virtual entities has the effect that two or more virtual entities – say $\mathcal{A}$ and a functionality – can in principle go into an infinite loop of invoking each other, without ever letting the environment to continue. For simplicity we shall consider only adversaries which have a finite bound (possibly growing with the security parameter) on the number of interactions with functionalities at one time.

[16]To be fully formal, we consider families of machines instead of machines. In each family, the machines are indexed by the security parameter. A composition as used in the description of closure property here takes a finite number of families, and produces a new family of machines. For brevity, we omit explicit references to machine families.

B Other Protocols for 3-party Functionalities

Recall that two kinds of multi-party SFE functionalities are realizable (w.r.t private channels): *aggregated* and *disseminated*. As simple non-trivial examples which are realizable, one may think of broadcast (which is a disseminated functionality) and aggregated XOR (or sum, in a small group). Broadcast[17] was shown to be possible in [24] using a simple protocol, and it is easily seen that aggregated XOR is realized by a protocol (over private channels) in which every sending party additively shares its inputs with everyone else (including the receiver and oneself), then in a second phase sends the XOR of the shares they received to the receiver.

However, not all aggregated or supervised functionalities are that simple, even for the 3-party case. In this section we investigate two of them and give non-trivial protocols for both.

Disseminated OR. The *disseminated OR* functionality has two output parties (parties #1 and #2) and a supervisor (party #3). It expects two bits b_1 and b_2 from the supervisor, respectively. Then it sends $(b_i, b_1 \vee b_2)$ to party i. This could be considered a non-trivial generalization of broadcast (to two receivers), as both output parties will agree on the second component of their outputs. Furthermore, using disseminated OR, any (small domain) disseminated functionality can be realized: for each forbidden pair of outputs, the two parties will use an instance of disseminated OR to ensure that at least one of them has an output which is different from that in the forbidden pair.

We give a protocol for disseminated OR in the private channels model. If $b_1 = b_2 = 0$, the supervisor sends zero to both output parties; on receiving zero, an output party sends zero to the other output party, and expects back the same. If it does not receive back zero, it aborts. Else output $(0, 0)$.

Otherwise, if $b_1 \vee b_2 = 1$, we proceed as follows: First, the supervisor sends $(b_i, b_1 \vee b_2)$ to party i. Then the parties attempt to convince each other that at least one of the parties received $b_i = 1$, in a secure way, as follows:

The supervisor generates random strings $r_1, r_2 \in \{0,1\}^k$, where k is the security parameter. For each $i \in \{1, 2\}$, if $b_i = 1$, the supervisor sends to party i the value r_i, otherwise it sends r_{3-i}.

Party 1 receives r_1 only if $b_1 = 1$, and party 2 receives r_1 only if $b_2 = 0$. Thus, if party 1 has $b_1 = 1$, broadcasting the value of r_1 acts as a proof that he holds 1, which party 2 will only be able to verify if has $b_2 = 0$.

More formally, if party i received $b_i = 1$, then it broadcasts r_i to the other parties. If party i received $b_i = 0$, then instead it broadcasts a randomly chosen $s_i \leftarrow \{0,1\}^k$ as a "fake" proof. After this broadcasting phase, the supervisor checks that for each i such that $b_i = 1$, the proof string broadcast by party i matches r_i. If so, it sends $\mathsf{continue}$ to both parties, else it sends abort.

If party i receives abort, it aborts the protocol at this point. Else, if it receives $\mathsf{continue}$, it continues as follows:

- If $b_i = 0$, then it verifies the other party's proof by comparing the broadcast value to r_{3-i} which it received from the supervisor. If the values do not match, it will abort, else it outputs $(0, 1)$.
- If $b_i = 1$ it outputs $(1, 1)$.

Aggregated OR. The aggregated OR functionality is one which takes bits b_1 and b_2 from input parties 1 and 2, respectively, and outputs $b_1 \vee b_2$ to the output party 3. There is a simple protocol for aggregated OR that uses the aggregated XOR functionality described above. The parties simply

[17] More precisely it is wrap(Broadcast) that is realizable, a distinction we keep implicit in this section.

compute the aggregated XOR of strings s_1 and s_2, where $s_i = 0^k$ if $b_i = 0$, and s_i is a random k-bit string if $b_i = 1$. The output party receives $s_1 \oplus s_2$, which is 0^k if both parties received input 0, and is randomly distributed otherwise.