Universally Composable Security With Local Adversaries

Ran Canetti and Margarita Vald*

Boston University and Tel Aviv University

Abstract. The traditional approach to formalizing ideal-model based definitions of security for multi-party protocols models adversaries (both real and ideal) as centralized entities that control all parties that deviate from the protocol. While this centralized-adversary modeling suffices for capturing basic security properties such as secrecy of local inputs and correctness of outputs against coordinated attacks, it turns out to be inadequate for capturing security properties that involve restricting the sharing of information between separate adversarial entities. Indeed, to capture collusion-freeness and and game-theoretic solution concepts, Alwen et.al. [Crypto, 2012] propose a new ideal-model based definitional framework that involves a de-centralized adversary.

We propose an alternative framework to that of Alwen et. al. We then observe that our framework allows capturing not only collusion-freeness and game-theoretic solution concepts, but also several other properties that involve the restriction of information flow among adversarial entities. These include some natural flavors of anonymity, deniability, timing separation, and information confinement. We also demonstrate the inability of existing formalisms to capture these properties.

We then prove strong composition properties for the proposed framework, and use these properties to demonstrate the security, within the new framework, of two very different protocols for securely evaluating any function of the parties' inputs.

* both authors are supported by the Check Point Institute for Information Security, Marie Curie grant PIRG03-GA-2008-230640, and ISF grant 0603805843.

Table of Contents

1 Introduction

Rigorously capturing the security properties of cryptographic protocols has proven to be a tricky endeavor. Over the years, the *trusted party* (or, simulation) paradigm has emerged as a useful and general definitional methodology. The basic idea, first coined in [GM84,GMR85,GMW87], is to say that a protocol "securely realizes" a given computational task if participating in the protocol ”emulates” the process of interacting with an imaginary "trusted party" that securely receives parties' inputs and locally computes their outputs. Intuitively, this paradigm allows expressing and capturing many security properties. Moreover, it has an attractive potential "composability" property: any system using a trusted party $\mathcal{F}$ should behave the same way when we replace the trusted party with the realizing protocol.

Over the years, many security definitions were based on this intuitive idea, e.g., [GL90,MR91,Can00,DM00,PW00,Can01,PS04,CDPW07]. First, these definitions formulate an execution model; then they formalize the notion of emulating an ideal task with an "ideal world" attacker, called simulator. The security requirement is based on the inability of an external observer to distinguish an “ideal world” execution from a real one.

These formalisms differ in many ways; however, they have one major thing in common: they all model the attacker as a centralized entity, who can corrupt parties, coordinate their behavior and, intuitively, constitute an "evil coalition" against the protocol being executed. This seems to be an oversimplification of real life situations. Indeed, in real life, parties are often individuals who are not necessary controlled by the same entity or have anything in common. It would seem that letting the malicious parties coordinate their attacks should be a strengthening of the model; however, when this power is also given to the adversary in the ideal model (aka the simulator), the security guarantee can potentially be weakened. Therefore, a natural question to ask is whether it is justified to model the attacker as a centralized entity or does this modeling unduly limit its expressiveness?

Indeed, the existing formalisms do capture basic properties such as privacy of inputs, and correctness of outputs against coordinated attack. However, as has been observed in the past, there exist security concerns that are not naturally captured using the centralized adversary approach. Consider for instance the *collusion- freeness* concern: a protocol is *collusion-free* if even misbehaving protocol participants cannot use the protocol to exchange "disallowed" information without being detected. As pointed out by [ILM05], "centralized simulator" formalisms do not capture the inability of parties to collude. That is, with a centralized adversary, a protocol might allow collusions between corrupted parties even when it realizes an ideal task that is collusion-free.

An additional known limitation of standard security notions is cryptographic implementations of game-theoretic mechanisms. In contrast to cryptography, game theory considers rational players that behave according to their individual goals. In many realistic settings, the incentive structure depends on whom players can collaborate with and the cost of this collaboration. Security with a centralized adversary does not guarantee that the incentive structure with respect to collaboration is preserved when moving from the ideal protocol to the one that realizes it. Consequently, it does not correctly capture the incentive structure and does not suffice for preserving game-theoretic solution concepts that restrict the formation of coalitions.

A natural way to handle those concerns would be to strengthen the model by requiring that the simulation be “local” in some sense; that is, shattering the centralized simulator to many simulators, where each simulator has only some “local’’ information and is responsible to simulate adversarial behavior in only a “local’’ sense. However requiring local simulators while allowing the adversary to be centralized results in an unrealistically strong security requirement that fails to admit many useful schemes that have practical security guarantees. Therefore, the next promising idea would be to restrict also the adversary to be local. This approach indeed appears in the works of [LMS05,ASV08,AKL$^+$09,ILM11,MR11,AKMZ12]. In particular, [AKMZ12] gives general model with a composition theorem and application to game-theory. These works give different and incomparable definitions of collusion-freeness; a common aspect is that they all postulate an adversary/simulator for each *participant*, where a participant represents an entity that is identified via its party identifier and treated as a “single domain” (i.e., it is corrupt as a unit,

either wholly or none at all). However, as we demonstrate below, there are a number of security concerns that cannot be naturally captured even by the above formalizations of local simulation.

Our contributions. We provide an alternative formalization of the local simulators approach in a way that preserves its intuitive appeal and captures reality more tightly. In particular, we establish a general security notion that allows capturing the requirements of arbitrary tasks while preserving the local view of each individual component *and each communication link between components* in the system. This notion enables expressing variety of partitions of the system. Specifically, our first contribution is to refine the UC framework to deal with the locality of information available to clusters of components. The new formalism, called local UC (LUC), assigns a different adversary/simulator to each *ordered pair* of participants. Intuitively, the adversaries/simulators assigned to a pair of parties handle all the communication between the two parties. Informally,

> If π is an *LUC-secure* protocol that implements a trusted party $\mathcal{F}$, then each *individual component* participating in π affects each other entity in the system no more than it does so in the ideal execution with $\mathcal{F}$.

Note that this is conceptually different from the guarantees provided by the UC framework of [Can01] and the CP framework of [AKMZ12]. In the UC framework, protocols that implement a trusted party are guaranteed to have similar effect on the external environment as in the ideal execution with $\mathcal{F}$. In the CP framework, the protocol is guaranteed to have the same effect as the trusted party *individually on each entity*. In the LUC framework, it is guaranteed that *each entity affects each other entity* in the same way as in the ideal execution.

This refined granularity allows LUC to capture various concerns that cannot be captured by previous frameworks. We show how LUC captures some actual security concerns and demonstrate the inability of existing notions to capture these concerns. Specifically, we address some flavors of anonymity, deniability, collusion-freeness, information confinement, and preservation of incentive structure.

We also extend the UC composition theorem and the dummy adversary theorem to the new framework. We obtain strong composition result that enables "game theoretic composition", i.e., composition that preserves the power of coalitions (whatever they may be). Moreover, our strong composition also preserves deniability and confinement.

Next we present two protocols for secure function evaluation with LUC security. The protocols, called the Physical GMW and the Mediated SFE protocols, satisfy the new security definition. The protocols are very different from each other: The Physical GMW protocol, which is strongly inspired by [ILM05], models players sitting in a room equipped with machines and jointly computing a function. The Mediated SFE protocol is a simplified version of [AKL$^+$09]. Like there, we use a semi-trusted mediator. That is, if the mediator is honest then the protocol is LUC secure. It is also UC secure in the standard sense even if the mediator is corrupted. It is interesting to note that although these two protocols have significantly different nature, they are both analyzable within our framework.

1.1 Our contributions in more details

The new formalism. In a nutshell, the new modeling proceeds as follows. Recall that in the UC framework, the adversary is a centralized entity that not only controls the communication in the network, but also coordinates the corrupted parties' behavior. This centralization is also inherited by the simulator. As mentioned above, while this modeling captures privacy and correctness, which are "global'' properties of the execution, it certainly does not capture rationality or locality of information. This implicitly means that only the situations where corrupted parties enjoy global view of the system are being fully captured.

A first attempt to bridge this gap would be to follow the formalisms of [ASV08,AKMZ12,MR11] and consider one adversary per party. However, this modeling does not completely capture reality either. Consider for instance the following scenario: one of two parties A and B is having a conversation with a third party C. Later C is instructed to transfer this information to some potentially misbehaved fourth party D without revealing whether the source was A or B. Clearly for the protocol to make intuitive sense, A and B need to assume that C is trusted, or "incorruptible''. However, going back to the suggested model we notice that the adversary associated with C participates in both conversations and can thus correlate the communication without corrupting C at all, thereby harming the anonymity in a way that is not intended by the protocol. In other words, the suggested modeling does not distinguish between the "obviously insecure" protocol that allows C to be corrupted, and the "obviously secure" protocol that uses an incorruptible C. We conclude that having a single adversary per party does not faithfully model honest parties. This motivates us to look for a more refined model.

To adequately capture locality, we extend the UC model as follows: For each party identity (denoted PID) we consider an adversary for any PID it might communicate with. In other words, each pair of PIDs has a pair of adversaries, where each adversary is in a different side of the "potential communication line". Each local adversary is in charge of a specific communication line and is aware only of the communication via this line.

We also let the environment directly control the communication, by letting the local adversaries communicate with each other only through the environment. This is an important definitional choice that is different from [AKMZ12]. In particular, this means that the centralized simulator no longer exists and, each local adversary is replaced with a local simulator in the ideal process, where the protocol is replaced by the trusted party. The trusted party may allow different subsets of simulators to communicate by forwarding messages between them. Therefore, the communication interface provided by the trusted party to the simulators represents partition of the system to clusters. The effect of this modeling is that the simulator for an entity can no longer rely on other parties' internal information or communication in which it was not present. This way, a proof of security relies only on each entity's local information, and potentially, represents independence of clusters defined by the trusted party.

To preserve meaningfulness, we allow the local adversaries to communicate across party identities only via the environment or with ideal functionalities. Aside from these modifications in the adversarial interface, the model is identical to the UC model.

Capturing concerns. We discuss variety of security concerns that are captured by LUC security but not in other security notions:

Collusion-freeness. To provide initial evidence for the expressiveness of LUC, we consider any UC-secure protocol for multi-party secure computation (e.g. the [CLOS02] protocol). While this protocol UC realizes any ideal functionality (even ones that guarantee collusion freeness) in the presence of malicious adversaries, it allows individually corrupted parties to collude quite freely, even when the environment does not pass any information among parties. Indeed, this protocol does not LUC-realize any ideal functionality that guarantees collusion-freeness. This is so even in the presence of only semi-honest adversaries. The reason for the failure in the LUC model is the inability of the separate simulators to produce consistent views on adversaries' shared information (i.e., scheduling, committed values etc.) We note that this concern is also captured by the definitions of [LMS05,ASV08,AKL$^+$09,AKMZ12].

Anonymity. We consider several flavors of anonymity such as existence-anonymity, timing-anonymity, and sender-anonymity. Specifically, we show UC and Collusion-Preserving (CP) realizations of ideal functionalities that have these anonymity requirements by a protocol that does not have these properties. We'll then show that this realization is not LUC secure. Let us informally present the above flavors of anonymity: The first anonymity concern we present is existence-anonymity. Intuitively, we would like to have a "dropbox'' that does not let the recipient know whether a new message was

received, and thus hides information regarding the existence of the sender.
Consider the following one-time-dropbox functionality. The dropbox is a virtual box initialized with some random file. People can put files into ones dropbox. In addition, the owner can one time query the dropbox if any new file has been received; if there are any incoming files in the box, they would be delivered to the owner; else the default file would be delivered.
Indeed, whenever receiving a file from this dropbox, there is no certainty regarding the existence of a sender. Correspondingly, any protocol that LUC-realizes the dropbox functionality is guaranteed to provide anonymity regarding the existence of a sender. This is not so for standard UC security.

An additional anonymity concern that we consider is timing-anonymity. Timing-anonymity means hiding the time in which an action took place. For example consider the
following email feature: whenever sending an email, the sender can delay the sending of the email by some amount of time (say, randomly chosen from some domain).
Indeed, upon receiving an email, the receiver does not know when this email was sent. This property can be captured via an ideal functionality in a straightforward way. Again, any protocol that LUC-realizes this functionality will provide anonymity regarding the time of sending. This is not so for standard UC security.

An additional anonymity property already mentioned here is sender-anonymity. The common way to achieve this anonymity property in practice is onion routing. In the work of [CL05] the onion-routing problem is defined in the UC framework; however, they only address a potential solution to the sender-anonymity concern rather than the concern itself. In contrast, we formalize the sender-anonymity property. We also show how UC security (and even CP security) fail to capture this property. Specifically, we define an ideal functionality and show a protocol that is clearly non-anonymous but still CP-realizes the functionality according to the definition of [AKMZ12]. This protocol is not LUC secure.

Deniability. It was pointed out in [CDPW07,DKSW09] that UC security does not guarantee deniability due to issues with modeling of the PKI. While these issues were resolved in the context of global setup and deniable authentication in the generalized UC framework, it turns out that the UC formalism does not capture another deniability flavor, called *Bi-deniability* (the name is taken from [OPW11]): A protocol is *Bi-deniable* if the protocol participants can "deny" before a judge having participated in the protocol by arguing that any "evidence" of their participation in the protocol could have been fabricated without their involvement, *even if there exists an external entity that has an access to parties' log files of the communication*. In the context of authentication, the judge is provided with "evidences" of sender's participation not only by the receiver but also by this external entity. Specifically, the sender can argue that any "evidence" of participation was fabricated by this external entity, even though this external entity cannot communicate with the receiver and only has an access to the communication log files of the sender. This notion is stronger than standard deniability, in which sender's log files are ideally hidden from the judge. To motivate bi-deniability consider a corporation that is obligated to store its communication log files. The log files are collected by an external law enforcement agency. Clearly, it would be desirable to ensure that even if these files are disclosed, the corporation can always deny their authenticity.

In this work, we give a simulation-based definition of Bi-deniable authentication and prove its equivalence to LUC secure authentication. Moreover, due to the strong connection between Bi-deniability and LUC security, we obtain that Bi-deniability is preserved under composition. In addition, we show that UC framework fails to capture this flavor of deniability.

Confinement. Another important concern that seems hard to capture by the standard notions is the *information confinement* property, defined by [Lam73]. A protocol is said to enforce confinement if even misbehaving participants cannot leak secret information that they possess across predefined boundaries. [HKN05] presents a game based definition of confinement. Their definition introduces

changes in the basic UC model, but still considers a centralized adversary. We show that the definition of [HKN05] is excessively strong and protocols that clearly enforce confinement fail to admit it. The root of the problem is the centralized adversary that enables information flow to unauthorized entities.

Intuitively, separate adversaries controlling different parts of the network or different groups of parties would indeed capture this requirement more tightly. We present a formal definition of confinement and show that LUC security implies it. Similarly to Bi-deniability, we obtain composability with respect to confinement. In addition, we show the inability of UC to capture confinement. More specifically, we show that any UC functionality that enforces confinement is *super-ideal* in some well defined sense. As before, this is not so for LUC functionalities.

Game-theoretic implications. As pointed out in [ILM05] standard security does not suffice for implementation of general equilibria due to collusion. In order to overcome this problem, new notions were defined in [ILM05,ILM08,ILM11,AKMZ12]. Specifically, [ILM05,ILM08,ILM11] define notions of mechanism implementation; however, these notions are specific to the problem at hand and are not suitable for reasoning about standard security. [AKMZ12] translate their security notion to the game-theoretic setting and define a corresponding model of mediated games. In addition, they show that their constructed compiler achieves preservation of incentive-structure (more details appears in Section 2). In this work, we show how protocols modeled in the LUC framework can be viewed as games. Moreover, we show that *any protocol* that LUC-securely realizes some ideal functionality preserves the incentive structure of the realized functionality. More concretely, for any LUC-secure protocol π there exists an efficient mapping between real world strategies and ideal world strategies that can be computed by each player in a local manner and achieves indistinguishable payoffs. This implies that any Nash Equilibrium (NE) in the ideal-world game is mapped to a computational NE in the real-world game and no new equilibria are introduced.

Composition and dummy adversary. We demonstrate that LUC-security is preserved under composition. Due to the local nature of the model, this preservation applies not only to basic security concerns under composition, but rather to much more general concerns such as deniability, confinement, and game-theoretic solution concepts. The obtained game-theoretic composition implies that Nash equilibrium is preserved under non-concurrent composition.

We also extend the dummy adversary notion to the local UC framework, and show its equivalence to the general LUC-security notion.

An interesting line for future research is to try to cast the LUC framework within the Abstract Cryptography framework [MR11]. In particular, such a work might provide a unified basis for the LUC, CP and UC frameworks.

1.1.1 LUC secure protocols

We sketch the two general multiparty computation protocols that we analyze in this work.

The physical GMW protocol. The GMW version we use is the protocol from [Gol04]. Still, our construction is strongly inspired by [ILM05]. We cast the protocol in the physical world by considering a set of players sitting in a room and jointly computing a function by evaluation the gates of its circuit. In order to properly compute the function, the players use the following physical machinery: boxes with serial number, and machines for addition, multiplication, duplication, and shuffle of boxed values. In more details, Let $P_1, ..., P_n$ be a set of parties in the room and let f be the function of interest. Next:

1. *Sharing the inputs:* Each player partitions its input to random shares, one share for each player and then, it publically sends those shares, in opaque boxes, to the players.
2. *Circuit emulation:* Proceeding by the order of wires, all players jointly and publically evaluate the circuit gates.

3. *Output reconstruction:* Each player publically hands the Boxes of the output shares to the appropriate parties. Lastly, each party privately opens the boxes and computes its output.

Theorem 1 (Informal statement). *Let f be a PPT function. Then, there exists a protocol that information-theoretically LUC-realizes the SFE functionality with respect to adaptive adversaries.*

Throughout the process, everyone sees which operations are performed by each player. Still, the actual values inside the boxes remain secret.

In contrast with classic GMW protocol, here the byzantine case is not done by introducing ZK proofs; rather the primitives themselves are robust.

We achieve LUC-security for any number of corrupted parties. While the work of [ILM05] requires at least one honest party for the collusion-freeness to hold, we achieve LUC security (which implies collusion-freeness) even when all the parties are corrupted. In addition, this protocol meets the strong notion of *perfect implementation* defined by [ILM11], and therefore achieves privacy, strategy, and complexity equivalence.

The Mediated-SFE protocol. We present here a high-level description of the mediated protocol, following [AKL$^+$09].

Let $P_1, ..., P_n$, and mediator $\mathcal{M}$ be a set of parties and let π be a k-round protocol that UC-securely computes function $\mathcal{F}$. (Inspired by [AKMZ12], we think of the protocol as running directly over unauthenticated communication channels.) The protocol π is compiled to a new LUC-secure protocol for computing $\mathcal{F}$ with a semi-trusted mediator, where all the communication is done through the mediator. Specifically, for each round of the protocol π does:

1. Each party and $\mathcal{M}$ runs two-party secure computation, which outputs to $\mathcal{M}$ the next round messages of this party in π.
2. $\mathcal{M}$ sends a commitment to the relevant messages to each party P_i.
3. In the last round of π, the mediator $\mathcal{M}$ and each party run secure two-party computation, where each party obtain its output.

Theorem 2 (Informal statement). *Given a (poly-time) function $f = (f_1, ..., f_n)$ and a protocol π that UC-realizes the SFE functionality. Then there exists a protocol Π that LUC-realizes the SFE functionality with respect to adaptive adversaries.*

When $\mathcal{M}$ is honest it separates the parties of π and makes them be independent of each other. When $\mathcal{M}$ is corrupted, the independence disappears. Still, we obtain standard UC security.

We strengthen the protocol to be immune to powerful adversaries that control the scheduling, gain information via leakage in the protocol, and are able to adaptively corrupt players. In contrast to [AKMZ12], we do not assume ideally secure channels between parties and the mediator.

Organization. Section 2 presents the main differences between the CP results and this paper. Section 3 presents the LUC-security framework in details. Section 4 proves the composition theorem. Section 5 shows how LUC relates to previous security notions. Section 6 presents the game-theoretic implications and the insufficiency of standard notions to capture interesting flavors of anonymity, deniability and confinement. Sections 7 and 8 present our protocols for secure function evaluation.

2 Collusion-preserving Computation

For clarity, we briefly review the work of [AKMZ12], since it is the most related work to ours. This work defines a security model which we denote by the CP model (for Collusion Preserving computation).

The CP model is a special case of LUC and is similar to the GUC model of [CDPW07]. The differences between CP and GUC are the following: Instead of centralized adversary, the CP model considers an adversary for each party, and each adversary can corrupt only the party associated with its identity. In addition, the adversaries communicate neither with the honest parties nor with each other. In the CP model, all communication is done via functionalities called resources. The CP security definition is an extension of the GUC definition to the multiple adversary setting. Let $\mathcal{R}$ and $\mathcal{F}$ be n-party resources. Let π and ϕ be n-party protocols using resources $\mathcal{R}$ and $\mathcal{F}$ respectively. Then, we say that a protocol π CP-realizes ϕ if for any set of n adversaries $\{\mathcal{A}_i\}_{i\in[n]}$ there exists a set of n simulators $\{\mathcal{S}_i\}_{i\in[n]}$ such that no environment can distinguish between an execution of π running with $\{\mathcal{A}_i\}_{i\in[n]}$ and ϕ running with $\{\mathcal{S}_i\}_{i\in[n]}$.

The main results presented in the [AKMZ12] paper are the following:

- A composition theorem with respect to the same resource within the CP model. That is, protocols using different resources cannot be composed.
- A compiler for Secure Function Evaluation with a semi-trusted mediator as a resource. [AKMZ12] models the communication links between parties and the mediator as ideal channels, where absolutely no information regarding communication is seen by the environment (and the adversaries). In addition to the ideally secure channels, the analysis of the CP compiler heavily relies on the fact that the mediator is modeled as a resource as opposed to a party with its own adversary. This does not allow the environment to witness any communication, and therefore enables simulation in the CP model.
- A proof that the above compiler preserves the incentive-structure. That is, they translate their security notion to the game-theoretic setting and define a corresponding model of mediated games. Then they show that their security notion implies that any strategy in the ideal-world game (with ideal SFE resource) can be mapped to a strategy in the game defined by the CP compiler and vice versa.

Summary of the main differences between our modeling and results from that of [AKMZ12]:

- The LUC execution model requires all the communication to go via the environment. This gives a less idealized and more realistic modeling of communication. In our constructed compiler, the mediator is modeled as a party and all the communication in the protocol is done via the environment. We remark that this more realistic modeling would not allow the CP compiler to achieve CP security.
- The LUC composition theorem is with respect to any protocol. That is, LUC securely holds even if protocols use different functionalities.
- The LUC model can capture additional concerns (e.g., flavors of anonymity, deniability, confinement).
- The LUC model provides more general implication and composition for game theoretic modeling (i.e., any LUC secure realization preserves structure-incentive as opposed to a specific construction of CP).

3 Defining security of protocols

In this section we introduce our new Local UC (LUC) framework and discuss its relationship to basic UC security. Many of the low-level technical details, especially those that are essentially identical to those of the basic UC framework, are omitted. A full treatment of these details can be found in [Can01]. We begin with a brief review of the basic UC framework of [Can01].

3.1 Basic UC security

We first briefly review the basic computational model, using interactive Turing machines (ITMs). Although we do not modify the underlying computational model, familiarity with it is important for understanding our model.

Systems of ITMs. To capture the mechanics of computation and communication in computer networks, the UC framework employs an extension of the Interactive Turing Machine (ITM) model. A computer program (such as for a protocol, or perhaps program of the adversary) is modeled as an ITM. An execution experiment consists of a system of ITMs which are instantiated and executed, with multiple instances possibly sharing the same ITM code. (More formally, a system of ITMs is governed by a *control function* which enforces the rules of interaction among ITMs as required by the protocol execution experiment. Here we will omit the full formalisms of the control function, which can be found in [Can01].)

A particular executing ITM instance running in the network is referred to as an ITI, and we must have a means to distinguish individual ITIs from one another even if they happen to be running identical ITM code. Therefore, in addition to the program code of the ITM they instantiate, individual ITIs are parametrized by a party ID (pid) and a session ID (sid). We require that each ITI can be uniquely identified by the identity pair $id = (pid,\ sid)$, irrespective of the code it may be running. All ITIs running with the same code and session ID are said to be a part of the same protocol session, and the *party* IDs are used to distinguish among the various ITIs participating in a particular protocol *session*. The sids are unique. We also refer to a protocol session running with ITM code as an instance of protocol. ITMs are allowed to communicate with each other via the use of three kinds of I/O tapes: local input tapes, local subroutine output tapes, and communication tapes. Writes to the local input tapes of a particular ITI must include both the identity and the code of the intended target ITI, and the ITI running with the specified identity must also be running the specified code, or else an error condition occurs. If a target ITI with the specified identity does not exist, it is created ("invoked"), and given the specified code. We also require that when an ITI writes to the local subroutine output tape of another ITI, it must provide its own code. Finally, all "external" communications are passed via the communication tapes, which disclose to the recipient neither the code of the intended recipient ITI, nor the code of the sending ITI (but merely their identities). The control function determines which external-write instructions are "allowed"; "un-allowed" instructions are ignored.

The UC Protocol Execution Experiment. The UC protocol execution experiment is defined as a system of ITMs that's parametrized by three ITMs. An ITM π specifies the code of the challenge protocol for the experiment, an ITM $\mathcal{A}$ specifies the code of the adversary, and an ITM $\mathcal{Z}$ provides the code of the environment. The protocol execution experiment defines precise restrictions on the order in which ITIs are activated, and which ITIs are allowed to invoke or communicate with each other. Formally, the restrictions are enforced via the control function. The formal details of the control function can be found in [Can01], but we informally describe some of the relevant details. The experiment initially launches only an ITI running $\mathcal{Z}$. In turn, $\mathcal{Z}$ is permitted to invoke only a single ITI running $\mathcal{A}$, followed by (multiple) ITIs running the "challenge protocol" provided that these ITIs running π all share the same sid. This sid, along with the PIDs of all the ITIs running, may be chosen arbitrarily by $\mathcal{Z}$. In summary, the environment can communicate only with the ITI running the code of the adversary $\mathcal{A}$, and ITIs participating in a single session of protocol π (in a limited way). The output of the environment $\mathcal{Z}$ in this basic UC protocol execution experiment is denoted by $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

Ideal Functionalities. An ideal functionality $\mathcal{F}$ is an ITM whose code represents a desired (interactive) function to be computed by parties or other protocols which may invoke it as a subroutine (and thus, in a perfectly secure way). The pid of any ITI running $\mathcal{F}$ is set to the special value $\perp$, indicating that the ITI is an ideal functionality. $\mathcal{F}$ accepts input from other ITIs that have the same sid as $\mathcal{F}$, and may write outputs to multiple ITIs as well. Every ideal functionality $\mathcal{F}$ also induces an ideal protocol $\mathrm{IDEAL}_{\mathcal{F}}$. Parties running $\mathrm{IDEAL}_{\mathcal{F}}$ with the session ID sid act as dummy parties, simply forwarding their inputs to the input tape of an ITI running $\mathcal{F}$ with the same sid, and copying any subroutine out-

put received from $\mathcal{F}$ to the subroutine output tape of the ITI which instructed by $\mathcal{F}$. The definition of dummy adversary presented here is different from the standard definition of [Can01] and allow the ideal functionality to choose the output destination.

UC Emulation and Realizations. In the basic UC framework, a protocol π is said to UC-emulate another protocol ϕ if, for any adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$ it holds that $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. That is, no environment behaves significantly differently in the protocol execution experiment when interacting with the challenge protocol π, under any given attack, than it does when interacting with the emulated protocol ϕ, under a simulation of that same attack. We say that a protocol π UC-realizes ideal functionality $\mathcal{F}$ if it is UC-emulates $\mathrm{IDEAL}_{\mathcal{F}}$.

3.2 The Local UC model of protocol execution

The model of protocol execution is defined in terms of a system of ITMs as in [Can01]. At first, a set of party IDs is chosen. Then, we consider a pair of adversaries for all potentially communicating ITIs based on the chosen party IDs. In addition, jointly with the environment, these adversaries has a complete control over the communication between ITIs which under their custody, as opposed to the UC framework, where a centralized adversary controls all the communication in the system. Formally, all the technical difference from the UC framework is concentrated in the control function, presented below. The underlying computational model remains unchanged.

Execution of protocol π with environment $\mathcal{Z}$ and adversary $\mathcal{A}$

An execution of protocol π with an adversary $\mathcal{A}$ and environment $\mathcal{Z}$ is a run of an extended, parametrized system of ITMs as, with initial ITM $\mathcal{Z}$, and the following control function:

1. $\mathcal{Z}$ may only *pass inputs* to parties. At first, $\mathcal{Z}$ chooses a set of party IDs $\mathcal{P}$ and SID s. The first ITIs that $\mathcal{Z}$ passes input to is set to be the adversaries. That is, the identities of these ITIs are required to have a special value $id = ((i,j), \perp)$ where $i, j \in \mathcal{P}$. The code of these ITIs is set (by the control function) to be $\mathcal{A}$. Besides the adversaries, all the ITIs that $\mathcal{Z}$ passes input to are required to have the same SID, s, and party identity $\mathrm{PID} \in \mathcal{P}$. The code of all these ITIs is set (again, by the control function) to be π. Call this instance the main instance of π.
2. An adversary $\mathcal{A}_{(i,j)}$ (that is, an adversary with code $\mathcal{A}$ and id $((i,j), \perp)$) can either pass output to $\mathcal{Z}$, or deliver messages to the incoming communication tape of any party (ITI) in the system with PID=i where the sender identity of delivered messages must be PID=j. There are no restrictions on the contents of delivered messages with the following exception: Messages of the form `(Corrupt, id, p)` can be delivered only once $\mathcal{A}_{(i,j)}$ receives an input `(Corrupt, id, p)` from $\mathcal{Z}$. Here $id = (\mathrm{PID}, \mathrm{SID})$ is the identity of the recipient party, $\mathrm{PID} = i$, and p denotes potential additional parameters. These messages are used to model party corruption; see discussion in the text.
3. ITIs other than $\mathcal{Z}$ or the adversaries can send messages to an appropriate $\mathcal{A}_{(i,j)}$ according to their PID and the PID of the recipient, and also pass inputs and outputs to any other ITI other than $\mathcal{Z}$ and the adversaries as long as the designated party PID is the same as of the sender. In addition, the outputs of the main parties of π (namely, the parties with code π and with SID = s) may be channeled to $\mathcal{Z}$; see discussion in the text.

The responses to `(Corrupt)` messages are assumed to be specified within π and may vary according to the specific corruption model and the parameters included in the message.

Fig. 1: A summary of the model for protocol execution.

Let π be a protocol over a fixed set of parties. The model is parametrized by three ITMs: the protocol π to be executed, an environment $\mathcal{Z}$ and an adversary $\mathcal{A}$. That is, given π, $\mathcal{Z}$, $\mathcal{A}$, the model for

executing π is the following extended, parametrized system of PPT ITMs $(\mathcal{Z}, C^{\pi,\mathcal{A}}_{LEXEC})$ (as defined in the UC model). The initial ITM in the system is the environment $\mathcal{Z}$. The control function $C^{\pi,\mathcal{A}}_{LEXEC}$ is defined in the paragraphs below. Figure 1 presents a summary of the model. A graphical depiction appears in Figure 2

The input of the initial ITM $\mathcal{Z}$ represents some initial state of the environment in which the protocol execution takes place. In particular, it represents all the external inputs to the system, including the local inputs of all parties. As a first step, $\mathcal{Z}$ choose a set $\mathcal{P}$ of party IDs and SID s[1]. The first ITIs to be invoked by $\mathcal{Z}$ is set by the control function to be the adversaries. See discussion regarding static invocation in 3.2. An adversary with identity $id = ((i,j), \perp)$ where $i, j \in \mathcal{P}$, denoted $\mathcal{A}_{(i,j)}$, is invoke for each ordered pair $i, j \in \mathcal{P}$. The adversaries code is set to be $\mathcal{A}$. In addition, as the computation proceeds, $\mathcal{Z}$ can invoke any ITI, by passing inputs to it, subject to the restriction that all these ITIs have SID s and PID $\in \mathcal{P}$**.** The code of these ITIs is set by the control function to be π, regardless of the code specified by $\mathcal{Z}$**.** Consequently, all the ITIs invoked by $\mathcal{Z}$, except for the adversaries, are parties in a single instance of π**.** We call this instance the main instance of π in this execution. In addition, each time $\mathcal{Z}$ passes input to main parties, it may choose identity id which will appear as the identity of the sender (this id should be unique and no other party can use this SID)**.** The control function remembers all $\mathcal{Z}$'s "fake" identities. All input (as well output) requests, made by the main parties, to one of these fake destinations are written on $\mathcal{Z}$'s output tape (as presented before, input and output requests to non-existing ITIs will invoke a new ITI).[2] Other than that, $\mathcal{Z}$ may not communicate with any other ITIs. That is, $\mathcal{Z}$ cannot pass inputs to any ITI other than the adversaries or the main parties of the main instance of π, nor can any ITI other than these pass outputs to $\mathcal{Z}$.

Each adversary $\mathcal{A}_{(i,j)}$ is allowed to send messages to any ITI in the system with PID$= i$ where the sender identity of delivered messages must be PID=j. There need not be any correspondence between the messages sent by the parties and the messages delivered by the adversaries. The adversaries may not pass input to any party, nor can it pass output to any party other than $\mathcal{Z}$. It is important to notice that there is no direct communication between the adversaries and all their communication must go through the environment.

Adversaries may also *corrupt* parties. Corruption of a party (ITI) with identity *id* is modeled via a special `(Corrupt,` *id, p*`)` message delivered by $\mathcal{A}_{(i,j)}$ to that ITI, where *p* denotes potential additional parameters. The control function allows delivery of that message only if $\mathcal{A}_{(i,j)}$ previously received a special `(Corrupt,` *id, p*`)` from the environment $\mathcal{Z}$ and party ID is i.

Any ITI other than $\mathcal{Z}$ and the adversaries, namely the parties and sub-parties of the main instance of π, as well as the ITIs invoked by the adversaries, are allowed to pass inputs and outputs to any other ITI other than $\mathcal{Z}$ and the adversaries subject to the restriction that the recipient have the same PID as the sender (except when the recipient is an ideal functionality). In addition, they can send messages to the adversaries where adversary's $\text{PID}_{(i,j)}$ requires sender's PID i and recipient's PID j. (These messages may indicate an identity of an intended recipient ITI; but the adversaries is not obliged to respect these indications.)

For all input and output requests the control function will omit the code of the sender (except when the sender is a dummy party or the recipient is $\mathcal{Z}$).

The response of the party or sub-party to a `(Corrupt)` message is not defined in the general model; rather, it is left to the protocol. Here we specify one corruption model, namely that of Byzantine party corruption. We extend the known definition to fit multiple adversaries. Here, once a party or a sub-party receives a `(Corrupt)` message, it sends to that adversary its entire current local state. Also, in all future activations, a corrupted ITI merely forwards the incoming information to that adversary and follows instructions of all PID related adversaries.

[1] This is different from the UC model where the party IDs can be chosen during protocol execution and may depend on any information available.

[2] The aliasing is necessary to make sure that a protocol would not be able to distinguish whether it is being executed directly by the environment or as a subroutine of another protocol. The UC framework do not address this issue and as a result composability problems arise. Nonetheless, this can be resolved by applying our aliasing in the UC framework.

Let $\mathrm{LEXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,\, z)$ denote the random variable describing the output of the execution of the system $(\mathcal{Z},\, C^{\pi,\mathcal{A}}_{LEXEC})$, where $\mathcal{Z}$'s input is z and k is the security parameter in use.

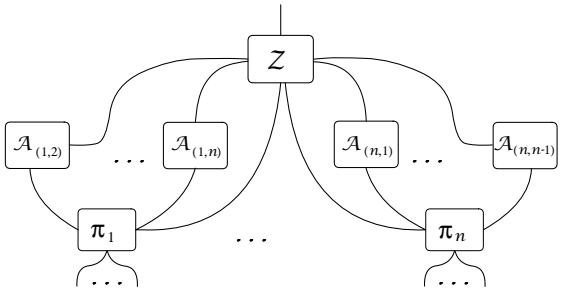

Fig. 2: The model of protocol execution. The environment $\mathcal{Z}$ writes the inputs and reads the subroutine outputs of the main parties running the protocol, while the adversaries, jointly with $\mathcal{Z}$ control the communication. In addition, $\mathcal{Z}$ may interact freely with all adversaries. The parties of π may have subroutines, to which $\mathcal{Z}$ has no direct access.

On the use of static invocation and predefined PIDs. Recall the order of activations of ITMs in an execution of a protocol: Once $\mathcal{Z}$ activated it chooses a set of PIDs and invokes all the adversaries attacking the protocol. For technical reasons, we restrict ourselves to protocols over a predefined set of PIDs in order to make the above mechanism meaningful. We remark that it is possible to formulate alternative mechanism that invokes the adversaries during the execution of the protocol and not prior to it. Such a more general mechanism do not require the PIDs of the participants to be determined in advance. Also, the results presented in this work hold with the more general formalism. However, such formalism would further complicate the model, and the extra generality does not seem essential in many cases.

3.3 Protocol emulation

This section formalizes the general notion of emulating one protocol via another protocol. In preparation for the formal definition, we first recall the notions of probability ensembles and indistinguishability, and extend the definition of *balanced environments* presented in [Can01]. We then define LUC-emulation in its general form.

Distribution ensembles and indistinguishability. A probability distribution ensemble $X = \{X(k,\, z)\}_{k\in\mathbb{N},\, z\in\{0,1\}^\star}$ is an infinite set of probability distributions, where a distribution $X(k,z)$ is associated with each $k \in \mathbb{N}$ and $z \in \{0,1\}^\star$. The ensembles considered in this work describe outputs of computations where the parameter z represents input, and the parameter k represents the security parameter.

Definition 1. *Two binary probability distribution ensembles X and Y are indistinguishable (written $X \approx Y$) if for any $c,\, d \in \mathbb{N}$ there exists $k_0 \in \mathbb{N}$ such that for all $k > k_0$ and all we have:*

$$\Pr\left[X\left(k,\, z\right) = 1\right] - \Pr\left[Y\left(k,\, z\right) = 1\right] < k^{-c}$$

.

As in [Can01], the probability distribution ensembles considered in this work represent outputs of systems of ITMs, namely outputs of environments. More precisely, we consider ensembles of the form $\mathrm{LEXEC}_{\pi,\,\mathcal{A},\,\mathcal{Z}} = \left\{\mathrm{LEXEC}_{\pi,\,\mathcal{A},\,\mathcal{Z}}\left(k,\, z\right)_{k\in\mathbb{N},\, z\in\{0,1\}^\star}\right\}$. It is stressed that definition 1 considers the

distributions $X(k,\ z)$ and $Y(k,\ z)$ only when the length of z is polynomial in k. This essentially means that we consider only environments that set the security parameter to be some polynomial fraction of the length of their input.

Balanced environments. In order to keep the notion of protocol emulation from being unnecessarily restrictive as discussed in [Can01], we extend the definition of Balanced environments and consider only environments where the amount of resources given to each adversary (namely, the length of the adversary's input) is at least some fixed polynomial fraction of the amount of resources given to the protocol. To be concrete, we consider only environments where, at any point in time during the execution, the overall length of the inputs given by $\mathcal{Z}$ to the parties of the main instance of π is at most k times the the length of input to each adversary.

The UC framework formalizes the general notion of emulating one protocol via another protocol. We extend their definition to LUC-emulation in its general form.

Definition 2. *[LUC-emulation]Let π and ϕ be PPT protocols. We say that π LUC-emulates ϕ if for any PPT adversary $\mathcal{A}$ there exists an PPT adversary $\mathcal{S}$ such that for any balanced PPT environment $\mathcal{Z}$ we have:* $\mathrm{LEXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx$ *LEXEC*$_{\phi,\mathcal{S},\mathcal{Z}}$

3.4 Realizing ideal functionalities; Hybrid protocols

Ideal functionalities. defined in the UC framework. See Section 3.1 for details.

Ideal protocols. The definition is similar to the UC framework with a difference in the input/output interface of the dummy parties. More formally, let $\mathcal{F}$ be an ideal functionality and sid be the session ID. Whenever a dummy party is activated with input v, it writes v onto the input tape of the ideal functionality $\mathcal{F}_{(sid,\ \perp)}$ (recall that this message includes the code of the dummy party). Messages delivered by the adversaries, including corruption messages, are ignored. Whenever a dummy party receives a value v from $\mathcal{F}$ on its subroutine output tape, it writes this value on the subroutine output tape of an ITI instructed by $\mathcal{F}$. In contrast, in the UC framework, dummy parties are obligated to forward any received output to the invoking ITI. This restriction is problematic since dummy parties can also be invoked by an ideal functionality, and therefore the destination of the output is undefined. Our approach can be applied in the UC framework to overcome this problem.

Let $\mathrm{LIDEAL}_{\mathcal{F},\mathcal{A},\mathcal{Z}}(k,z)$ denote the random variable $\mathrm{LEXEC}_{\mathrm{LIDEAL}_{\mathcal{F},\mathcal{A},\mathcal{Z}}(k,z)}$.

Let $\mathrm{LIDEAL}_{\mathcal{F},\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathrm{LIDEAL}_{\mathcal{F},\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N},\ z\in\{0,1\}^\star}$.

Realizing an ideal functionality. Protocols that realize an ideal functionality are defined as protocols that emulate the ideal protocol for this ideal functionality:

Definition 3. *Let $\mathcal{F}$ be an ideal functionality and let π be a protocol. We say that π LUC-realizes $\mathcal{F}$ if π LUC-emulates the ideal protocol for $\mathcal{F}$.*

Hybrid protocols. As in the UC framework, we define hybrid protocols to be protocols where, in addition to communicating via the adversary in the usual way, the parties also make calls to instances of ideal functionalities. In other words, an $\mathcal{F}$-hybrid protocol π, denoted by $\pi^{\mathcal{F}}$, is a protocol that includes subroutine calls to $\mathrm{LIDEAL}_{\mathcal{F}}$, the ideal protocol for $\mathcal{F}$.

Recall that running $\mathrm{LIDEAL}_{\mathcal{F}}$ means invoking "dummy parties" for $\mathcal{F}$, which in turn invoke an instance of $\mathcal{F}$. This somewhat "indirect" access to $\mathcal{F}$ permit the calling parties to specify the SIDs and PIDs of the dummy parties of $\mathcal{F}$. This modeling allows the adversaries to communicate freely through hybrid functionalities. This freedom makes the model to fallback to the UC model as in the input/output between adversaries scenario. Therefore, we consider only adversaries that invoke only the ideal functionalities specified by the protocol. More formally, a party controlled by an adversary invokes only the functionalities $\mathcal{F}$ that the executed protocol instructs, and specify the PIDs of the participating parties in $\mathrm{LIDEAL}_{\mathcal{F}}$ as instructed by the calling protocol.

3.5 Emulation with respect to the dummy adversary

Informally, the dummy adversary, denoted $\mathcal{D}$, only delivers to parties messages that were generated by the environment, and delivers to the environment all messages generated by the parties. The formal definition can be found in [Can01].

Let π, ϕ be protocols. We say that protocol π LUC-emulates protocol ϕ with respect to the dummy adversary $\mathcal{D}$ if there exists an adversary $\mathcal{S}_{\mathcal{D}}$ such that for any environment $\mathcal{Z}$ we have $\text{LEXEC}_{\phi,\mathcal{S}_{\mathcal{D}},\mathcal{Z}} \approx \text{LEXEC}_{\pi,\mathcal{D},\mathcal{Z}}$.

Claim. Let π, ϕ be protocols. Then π LUC-emulates ϕ according to the standard definition if and only if it LUC-emulates ϕ with respect to the dummy adversary.

Proof. The proof is similar to the proof presented in [Can01]. The construction of the adversary is identical to the construction in [Can01] and the construction of the distinguishing environment is straightforwardly extended to communicate with multiple adversaries. For completeness we present the proof below.

If π LUC-emulates ϕ according to definition 2 then it LUC-emulates ϕ with respect to dummy adversary. Let π, ϕ be protocols and let $\mathcal{S}_{\mathcal{D}}$ be the adversary guaranteed by the definition of emulation with respect to dummy adversary. We show that π LUC-emulates ϕ according to the standard definition. Given an adversary $\mathcal{A}$ we construct an adversary $\mathcal{S}$ as follows. Simulator $\mathcal{S}$ interacting with parties running ϕ with appropriate PID and runs simulated instances of $\mathcal{A}$ and $\mathcal{S}_{\mathcal{D}}$ that are invoked with the same identity as of $\mathcal{S}$. In addition:

1. $\mathcal{S}$ forwards any input from the environment to the simulated $\mathcal{A}$, and forwards any output of $\mathcal{A}$ to the environment.
2. When the simulated $\mathcal{A}$ delivers a message m to an ITI with identity id and code c, $\mathcal{S}$ activates $\mathcal{S}_{\mathcal{D}}$ with input (m, id, c). Similarly, any output generated by $\mathcal{S}_{\mathcal{D}}$ is copied to the incoming communication tape of $\mathcal{A}$.
3. Whenever $\mathcal{S}_{\mathcal{D}}$ writes a message m on some tape of some ITI, $\mathcal{S}$ writes m to that tape of that ITI. Finally, when $\mathcal{S}$ obtains a message m on its incoming communication tape, it proceeds as follows: It first writes 1^m on the input tape of $\mathcal{S}_{\mathcal{D}}$; in the next activation it writes m on the incoming communication tape of $\mathcal{S}_{\mathcal{D}}$.

Validity of $\mathcal{S}$. Analysis of the running time of $\mathcal{S}$ can be found in [Can01].

Next we assert the validity of $\mathcal{S}$. Assume for contradiction that there is an adversary $\mathcal{A}$ and environment $\mathcal{Z}$ such that $\text{LEXEC}_{\phi,\mathcal{S},\mathcal{Z}} \not\approx \text{LEXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. We construct an environment $\mathcal{Z}_{\mathcal{D}}$ such that $\text{LEXEC}_{\phi,\mathcal{S}_{\mathcal{D}},\mathcal{Z}_{\mathcal{D}}} \not\approx \text{LEXEC}_{\pi,\mathcal{D},\mathcal{Z}_{\mathcal{D}}}$. Environment $\mathcal{Z}_{\mathcal{D}}$ runs an interaction between simulated instances of $\mathcal{Z}$ and $\mathcal{A}$. Let $\mathcal{P}$ be the PID set chosen by $\mathcal{Z}$. Next, $\mathcal{Z}_{\mathcal{D}}$ declare the same set $\mathcal{P}$ (this invokes all the adversaries according to $\mathcal{P}$) In addition:

1. All the inputs generated by $\mathcal{Z}$ to the adversaries are forwarded to the internal adversaries, and all of adversaries' outputs are forwarded to $\mathcal{Z}$.
2. Whenever $\mathcal{Z}_{\mathcal{D}}$ receives an output value v from adversary with identity $((i,j),\perp)$, $\mathcal{Z}_{\mathcal{D}}$ passes v to the simulated $\mathcal{A}$ with the same identity. Similarly, whenever a simulated $\mathcal{A}$ with identity $((i,j),\perp)$ delivers a message m to some ITI, $\mathcal{Z}_{\mathcal{D}}$ instructs the external adversary with the same identity to deliver message m to that ITI.
3. All inputs from $\mathcal{Z}$ to the parties of π are forwarded to the external parties, and all the outputs coming from the external parties are forwarded to $\mathcal{Z}$ as coming from the parties of π.
4. In addition, whenever $\mathcal{Z}_{\mathcal{D}}$ passes input of length m to some party, it first passes input $1^{p(m)}$ to all external adversaries, where $p()$ is the maximum between the polynomials bounding the run times of ϕ and π.
5. Finally, $\mathcal{Z}_{\mathcal{D}}$ outputs whatever the simulated $\mathcal{Z}$ outputs.

It can be readily verified that the ensembles $\text{LEXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ and $\text{LEXEC}_{\phi,\mathcal{S}_{\mathcal{D}},\mathcal{Z}_{\mathcal{D}}}$ are identical. Similarly, ensembles $\text{LEXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{LEXEC}_{\pi,\mathcal{D},\mathcal{Z}_{\mathcal{D}}}$ are identical as well.

4 Local Universal composition

This section extends the universal composition theorem to LUC framework. First we recall the composition operation as defined in [Can01] and states the extended composition theorem. Later, we present the proof.

4.1 The universal composition operation and theorem

Universal composition. We recall the composition operator of the UC framework, called the universal composition operator uc(), is defined as follows. Given a protocol ϕ, a protocol π (that presumably makes subroutine calls to ϕ), and a protocol ρ (that presumably LUC-emulates ϕ), the composed protocol $\pi^{\rho/\phi} = \text{UC}(\pi, \rho, \phi)$ is identical to protocol π, with the following modifications.

1. Wherever π contains an instruction to pass input x to an ITI running ϕ with identity $(sid,\ pid)$, then $\pi^{\rho/\phi}$ contains instead an instruction to pass input x to an ITI running ρ with identity $(sid,\ pid)$.
2. Whenever $\pi^{\rho/\phi}$ receives an output passed from $\rho(sid,\ pid')$ (i.e., from an ITI running ρ with identity $(sid,\ pid')$), it proceeds as π proceeds when it receives an output passed from $\phi(sid,\ pid')$.

When ϕ is understood from the context we use the shorthand π instead for $\pi^{\rho/\phi}$. Note that the definition presented above is the definition in [Can01].

Subroutine Respecting protocols. We recall the definition presented in [Can01]. An instance of a protocol ρ in an execution of π^{ρ} is subroutine respecting if no sub-party of this instance passes inputs or outputs to or from an ITI which is not a party or sub-party of this instance. Furthermore, all sub-parties of this instance ignore all inputs and outputs received from parties other than the parties and sub-parties of ρ. Protocol ρ is subroutine respecting if any instance of ρ in any execution of π^{ρ}, for any π, is subroutine respecting.

Theorem 3 (Universal composition). *Let π, ρ, ϕ be PPT protocols. In addition, ρ LUC-emulates ϕ and ρ, ϕ are subroutine respecting. Then protocol $\pi^{\rho/\phi}$ LUC-emulates protocol π.*

4.2 A proof

The proof uses the equivalent formulation of emulation with respect to dummy adversary and is a straightforward extension of the proof presented in [Can01].

Construction of $\mathcal{A}_\pi$. Let π, ϕ, ρ be PID respecting PPT protocols, where ρ LUC-emulates ϕ, and and let $\pi^{\rho} = \pi^{\rho/\phi}$ be the composed protocol. We wish to construct an adversary $\mathcal{A}_\pi$ so that no $\mathcal{Z}$ will be able to tell whether it is interacting with π^{ρ} and the dummy adversary or with π and $\mathcal{A}_\pi$. That is, for any $\mathcal{Z}$, $\mathcal{A}_\pi$ should satisfy $\text{LEXEC}_{\pi^{\rho},\mathcal{D},\mathcal{Z}} \approx \text{LEXEC}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}$.

The fact that ρ LUC-emulates ϕ guarantees that there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}_\rho$ we have: $\text{LEXEC}_{\rho,\mathcal{D},\mathcal{Z}_\rho} \approx \text{LEXEC}_{\phi,\mathcal{S},\mathcal{Z}_\rho}$.

The adversary $\mathcal{A}_\pi$ is constructed out of $\mathcal{S}$ and presented in Figure 3. The construction is identical to [Can01].

Validity of $\mathcal{A}_\pi$. First, note that $\mathcal{A}_\pi$ is PPT . In fact, the polynomial $p(\cdot)$ bounding the running time of $\mathcal{A}_\pi$ can be set to be the polynomial bounding the running time of $\mathcal{Z}$.

Next, assume that there exists an environment $\mathcal{Z}$ that violates the validity of adversary $\mathcal{A}_\pi$. We construct an environment $\mathcal{Z}_\rho$ that violates the validity of the adversary $\mathcal{S}$ with respect to a single run of ρ. More specifically, fix some input value z and a value k of the security parameter, and assume that

$$\text{LEXEC}_{\pi^{\rho},\mathcal{D},\mathcal{Z}}(k,\ z) - \text{LEXEC}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(k,\ z) \geq \epsilon \tag{1}$$

We show that

$$\text{LEXEC}_{\rho,\mathcal{D},\mathcal{Z}_\rho}(k,\ z) - \text{LEXEC}_{\phi,\mathcal{S},\mathcal{Z}_\rho}(k,\ z) \geq \frac{\epsilon}{t} \tag{2}$$

Adversary $\mathcal{A}_\pi$

Adversary $\mathcal{A}_\pi$ proceeds as follows, interacting with parties running protocol π and environment $\mathcal{Z}$.

1. When activated with input $(m,\ id,\ c)$, where m is a message, $id = (sid, pid)$ is an identity, and c is a code for an ITM, do:
   (a) If the specified recipient is a party of an instance sid' of ρ, or a subsidiary thereof, then first locate the internally running instance $\mathcal{S}_{(sid')}$ that handles the protocol instance with SID $= sid$. (That is, either $sid' = sid$ or sid is the SID of a subsidiary of a party with SID $= sid'$.) If no such instance of $\mathcal{S}$ is found, then internally invoke a new instance of $\mathcal{S}$ with identity (sid). Next, activate this instance of $\mathcal{S}$ with input (m, id, c) and follow its instructions.
   (b) Else (i.e., the specified recipient is not a party or sub-party of an instance of ρ), deliver the message m to the recipient.
2. When activated with an incoming message m from some party, do:
   (a) If m was sent by some party $\phi(sid, pid)$ of protocol ϕ, or a subsidiary thereof, then internally activate the instance$\mathcal{S}_{(sid)}$ of $\mathcal{S}$ with incoming message m from $\phi(sid, pid)$, and follow its instructions. (If no such instance of $\mathcal{S}$ exists then invoke it, internally, and label it $\mathcal{S}_{(sid)}$.)
   (b) If m was sent by another party or sub-party of π, then pass output $(m,\ id)$ to $\mathcal{Z}$.
3. When an instance of $\mathcal{S}$ internally generates a request to deliver a message m to some party, then deliver m to this party. When an instance of $\mathcal{S}$ requests to pass an output v to its environment then output v to $\mathcal{Z}$, but with the the exception that $\mathcal{A}_\pi$ mimics the time bounds of a dummy adversary $\mathcal{D}$. That is, $\mathcal{A}_\pi$ stops delivering output to $\mathcal{Z}$ as soon as the output length exceeds the overall input length of $\mathcal{A}_\pi$.

Fig. 3: The adversary for protocol π.

where $t = t\,(k,\ |z|)$ is a polynomial function.

In preparation to constructing $\mathcal{Z}_\rho$, we define the following distributions, and make some observations on $\mathcal{A}_\pi$. Consider an execution of protocol π with adversary $\mathcal{A}_\pi$ and environment $\mathcal{Z}$. Let $t = t(k,\ |z|)$ be an upper bound on the number of instances of ϕ within π in this execution. For $0 \leq l \leq t$, Let the l-hybrid model for running protocol π denote the extended system of ITMs that is identical to the basic model of computation, with the exception that the control function is modified as follows. The first l instances of ϕ to be invoked are treated as usual. The parties of all other instances of ϕ are replaced with the corresponding parties of ρ. Let $\mathrm{LEXEC}^{l}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}\,(k,\ z)$ denote the output of this system of ITMs on input z and security parameter k for the environment $\mathcal{Z}$.

We also define the following variants of adversary $\mathcal{A}_\pi$ and environment $\mathcal{Z}$**.** Let $\hat{\mathcal{A}}_\pi$ denote the adversary that is identical to $\mathcal{A}_\pi$, with the following exception. $\hat{\mathcal{A}}_\pi$ expects to have its input include an additional flag, a, that determines whether to consider invoking an instance of $\mathcal{S}$ in this activation. That is, if $a = 1$ then $\hat{\mathcal{A}}_\pi$ operates as $\mathcal{A}_\pi$. If a = 0 then $\hat{\mathcal{A}}_\pi$ skips the condition in Step (1a) in 3; instead it always runs Step (1b). Let $\mathcal{Z}^{(l)}$ denote the environment that is identical to $\mathcal{Z}$, with the following exceptions. Whenever $\mathcal{Z}$ generates a message $(m,\ id,\ c)$ to be delivered to the party running ρ with identity id, then $\mathcal{Z}^{(l)}$ passes input $(m,\ id,\ c,\ a)$ to the adversary, where a is set as follows. Let $id = (sid, pid)$. If sid is the SID of one of the first l instances of ϕ, then $\mathcal{Z}^{(l)}$ sets $a = 1$. Otherwise, $a = 0$.

We observe that, when $\hat{\mathcal{A}}_\pi$ interacts with $\mathcal{Z}^{(l)}$ and parties running π in the l-hybrid model, then it internally runs at most l instances of the simulator $\mathcal{S}$. The remaining instances of $\mathcal{S}$ are replaced by interacting with the actual parties or sub-parties of the corresponding instances of ρ. Consequently, we have that the output of $\mathcal{Z}^{(t)}$ from an interaction with π and $\hat{\mathcal{A}}_\pi$ in the t-hybrid model is distributed identically to the output of $\mathcal{Z}$ from an interaction with π and $\mathcal{A}_\pi$ in the basic model, i.e.

$\mathrm{LEXEC}^{t}_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(\mathrm{t})}}(k,\,z) = \mathrm{LEXEC}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(k,\,z)$. Similarly, the output of $\mathcal{Z}^{(0)}$ from an interaction with π and $\hat{\mathcal{A}}_\pi$ in the 0-hybrid model is distributed identically to the output of $\mathcal{Z}$ from an interaction with π^ρ in the basic model of computation, i.e. $\mathrm{LEXEC}^{0}_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(0)}}(k,\,z) = \mathrm{LEXEC}_{\pi^\rho,\mathcal{D},\mathcal{Z}}(k,\,z)$. Consequently, Inequality 1 can be rewritten as:

$$\mathrm{LEXEC}^{0}_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(0)}}(k,\,z) - \mathrm{LEXEC}^{t}_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(\mathrm{t})}}(k,\,z) \geq \epsilon \tag{3}$$

We turn to constructing and analyzing environment $\mathcal{Z}_\rho$. The construction of $\mathcal{Z}_\rho$ is presented in Figure 4. We first note that $\mathcal{Z}_\rho$ is PPT. This follows from the fact that the entire execution of the system is completed in polynomial number of steps.

The rest of the proof analyzes the validity of $\mathcal{Z}_\rho$. It follows from 3 that there exists a value $l \in \{1, ..., m\}$ such that

$$\left|\mathrm{LEXEC}^{l}_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(1)}}(k,\,z) - \mathrm{LEXEC}^{l-1}_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(1)}}(k,\,z)\right| \geq \frac{\epsilon}{t} \tag{4}$$

However, for every $l \in \{1, ..., m\}$ we have

$$\mathrm{LEXEC}_{\rho,\mathcal{S},\mathcal{Z}_\rho}(k,\,(z,\,l)) = \mathrm{LEXEC}^{l-1}_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(1)}}(k,\,z) \tag{5}$$

and

$$\mathrm{LEXEC}_{\phi,\mathcal{D},\mathcal{Z}_\rho}(k,\,(z,\,l)) = \mathrm{LEXEC}^{l}_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(1)}}(k,\,z) \tag{6}$$

Equations 5 and 6 follow from inspecting the code of $\mathcal{Z}_\rho$ and $\mathcal{A}_\pi$. In particular, if $\mathcal{Z}_\rho$ interacts with parties running ϕ then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ is distributed identically to the view of $\mathcal{Z}$ when interacting with π and adversaries running $\hat{\mathcal{A}}_\pi$ in the l-hybrid model. Similarly, if $\mathcal{Z}_\rho$ interacts with parties running ρ then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ is distributed identically to the view of $\mathcal{Z}$ run when interacting with π and adversaries running $\hat{\mathcal{A}}_\pi$ in the $(l-1)$-hybrid model. Here it is important to note that, since both ρ and ϕ are subroutine respecting, the only communication between the external instance of ρ or ϕ, and the ITIs outside this instance, is the inputs and outputs of the main parties of this instance. In addition, since ITIs can input and output to ITIs with the same PID, it is ensured that no ITIs invoked with party ID not in $\mathcal{P}$.

From Equations 4, 5and 6 it follows that:

$$\left|\mathrm{LEXEC}_{\rho,\mathcal{D},\mathcal{Z}_\rho}(k,\,(z,\,l)) - \mathrm{LEXEC}_{\phi,\mathcal{S},\mathcal{Z}_\rho}(k,\,(z,\,l))\right| \geq \frac{\epsilon}{t} \tag{7}$$

as desired.

Environment $\mathcal{Z}_\rho$

Environment $\mathcal{Z}_\rho$ proceeds as follows, given a value k for the security parameter, input z_ρ, and expecting to interact with parties running a single instance of ρ. We first present a procedure called Simulate(). Next we describe the main program of $\mathcal{Z}_\rho$.

Procedure Simulate(s, l)

1. Expect the parameter s to contain a global state of a system of ITMs representing an execution of protocol π in the l-hybrid model, with adversary $\hat{\mathcal{A}}_\pi$ and environment $\mathcal{Z}$. Continue a simulated execution from state s (making the necessary random choices along the way), until one of the following events occurs. Let ϕ_l denote the lth instance of ϕ that is invoked in the simulated execution, and let sid_l denote the SID of ϕ_l.
   (a) Some simulated party passes input x to a party with identity (sid_l, pid). (That is, the recipient party participates in the lth instance of ϕ.) In this case, save the current state of the simulated system in s, pass input x the actual party with identity (sid_l, pid), and complete this activation (use the identity of the simulated party).
   (b) The simulated $\mathcal{Z}$ passes input (m, id, c, a) to some simulated adversary $\hat{\mathcal{A}}_\pi$ with identity $((i, j), \perp)$, where $id = (sid_l, pid)$ or a subroutine thereof for some pid. In this case, save the current state of the simulated system in s, write the message (m, id, c) to the appropriate external adversary (with the same identity $(i, j), \perp$) and complete this activation.
   (c) The simulated environment $\mathcal{Z}$ halts. In this case, $\mathcal{Z}_\rho$ outputs whatever $\mathcal{Z}$ outputs and halts.

Main program for $\mathcal{Z}_\rho$

1. When activated for the first time, interpret the input z_ρ as a pair $z_\rho = (z, l)$ where z is an input for $\mathcal{Z}$, and $l \in \mathbb{N}$. Initialize a variable s to hold the initial global state of a system of ITMs representing an execution of protocol π in the l-hybrid model, with adversary $\hat{\mathcal{A}}_\pi$ and environment $\mathcal{Z}$ on input z and security parameter k. Let $\mathcal{P}$ be the PID set chosen by $\mathcal{Z}$. Next, $\mathcal{Z}_\rho$ declare the same set $\mathcal{P}$ (this invokes all the adversaries according to $\mathcal{P}$) and run Simulate(s, l).
2. In any other activation, let x be the new value written on the subroutine-output tape. Next:
   (a) Update the state s. That is:
      i. If the new value, x, was written by some party $P_{(id)}$ with identity $id = (sid_l, pid)$ then write (x, id) to the subroutine-output tape of the simulated party running π with party ID pid. (we use the same pid since ϕ and ρ are PID respecting protocols).
      ii. If the new value, x, was written by some adversary with identity $((i, j), \perp)$, then update the state of the simulated copy of $\hat{\mathcal{A}}_\pi$ with identity $((i, j), \perp)$ to include an output x generated by $S_{(sid_l)}$.
   (b) Simulate an execution of the system from state s. That is, run Simulate(s, l).

Fig. 4: The environment for a single instance of ρ.

5 Relation to UC framework

In this section we show that the LUC framework is a generalization that enhances expressiveness; it captures all the old specification, and in addition allow to capture stronger properties that are not capturable in the standard UC model. We define two general transformations between functionalities in the LUC model and functionalities in the standard UC model. The generality of the transformations is the key point here. In order for the relations to be well defined we consider protocols over fixed set of parties and restrict ourselves to a compatible UC analog of the LUC environment, denoted static PIDs environment.

Static PIDs environment. In order to reason in a meaningful way regarding the relation between LUC security and UC security we cast the LUC environment into the UC framework. As mentioned before, the UC environment can choose the PIDs adaptively during an execution of a protocol, as opposed to the LUC framework where it is obligated to declare all PIDs ahead of time. To bridge this gap we consider UC environments, called static PIDs environments, which behave as follows: As a first step, the environment $\mathcal{Z}$ chooses a set of party IDs $\mathcal{P}$. Later on, when $\mathcal{A}$ is invoked, it passes $\mathcal{P}$ as part of $\mathcal{A}$'s input.

5.1 LUC security implies UC security

In order to show that LUC is a generalization of the UC model we need as a first step to formally define a UC analog of each LUC functionality, this we call splitter functionalities. Intuitively, the main difference between the models is that in UC there is a centralized adversary where in LUC we have an adversary for each PID pair. Therefore, in LUC the functionality expect to receive messages from adversaries with different PIDs, where in the UC model the functionality receives messages from only one adversary. The splitter functionality is essentially a coordinator between the internal LUC functionality and the UC world outside. As before, we denote by $\mathcal{A}_{(i,j)}$ the adversary running code $\mathcal{A}$ with identity $((i,j),\perp)$.

Definition 4 (Splitter functionality). *Let $\mathcal{F}$ be a functionality in the LUC model. The splitter functionality, denoted by $sp\mathcal{F}$, is the suitable UC analog of $\mathcal{F}$. That is, the functionality $sp\mathcal{F}$ behaves the same as $\mathcal{F}$ with the following differences in the interface:*

1. *Upon receiving an input `(m,(i,j))` from the external adversary $\mathcal{A}$, $sp\mathcal{F}$ behaves as $\mathcal{F}$ would upon receiving input `m` from adversary $\mathcal{A}_{(i,j)}$.*
2. *Whenever $\mathcal{F}$ generate output `out` to adversary $\mathcal{A}_{(i,j)}$, then $sp\mathcal{F}$ gives `(out,(i,j))` to the external adversary.*

This transformation is extended in the natural way to protocols. Let π be a LUC protocol in the $\mathcal{Q}$-hybrid model. The analog UC protocol of π, is a protocol in the $sp\mathcal{Q}$-hybrid model where all instructions in π to communication with $\mathcal{Q}$ are replaced with instructions to communicate with $sp\mathcal{Q}$.

Having defined the splitter functionalities, we are ready to show that LUC security implies UC security with its analog UC functionality. Moreover, we show that there are functionalities which can be UC realized but not LUC realized.

Theorem 4. *[General statement] Let $\mathcal{Q}$ and $\mathcal{F}$ be functionalities in the LUC model. Let $\pi^{\mathcal{Q}}$ and $\phi^{\mathcal{F}}$ be PPT protocols. If $\pi^{\mathcal{Q}}$ LUC-emulates $\phi^{\mathcal{F}}$ then $\pi^{sp\mathcal{Q}}$ UC-emulates $\phi^{sp\mathcal{F}}$ with respect to static PIDs environments.*

Sketch of Proof The idea of the proof is that LUC realization of a protocol allows the adversaries to communicate only with the hybrid ideal functionality (if any). Therefore, considering a UC realization of the analog protocol is essentially splitting the centralized adversary into a set of disjoint adversaries which we obtained by the LUC realization.

The proof uses the equivalent formulation of emulation with respect to dummy adversary Let ϕ be a $\mathcal{F}$-hybrid protocol. Let π be a $\mathcal{Q}$- hybrid protocol that LUC-realizes $\phi^{\mathcal{F}}$. We wish to construct an

adversary $\mathcal{S}_\mathcal{D}$ so that no static PIDs $\mathcal{Z}$ will be able to tell whether it is interacting with $\pi^{sp\mathcal{Q}}$ and the dummy adversary or with $\phi^{sp\mathcal{F}}$ and $\mathcal{S}_\mathcal{D}$. That is, for any static PIDs $\mathcal{Z}$, $\mathcal{S}_\mathcal{D}$ should satisfy

$$\mathrm{EXEC}_{\pi^{\mathrm{sp}\mathcal{Q}},\mathcal{D},\mathcal{Z}} \approx \mathrm{EXEC}_{\phi^{\mathrm{sp}\mathcal{F}},\mathcal{S}_\mathcal{D},\mathcal{Z}} \tag{8}$$

The general outline of the proof proceeds as follows. The fact that $\pi^\mathcal{Q}$ realizes $\phi^\mathcal{F}$ guarantees that there exists an adversary (called a simulator) $\mathcal{S}$, such that for any environment $\mathcal{Z}_\mathcal{F}$ we have:

$$\mathrm{LEXEC}_{\pi^\mathcal{Q},\mathcal{D},\mathcal{Z}_\mathcal{F}} \approx \mathrm{LEXEC}_{\phi^\mathcal{F},\mathcal{S},\mathcal{Z}_\mathcal{F}} \tag{9}$$

Adversary $\mathcal{S}_\mathcal{D}$ is constructed out of $\mathcal{S}$ and operates as follows. Recall that $\mathcal{Z}$ expects to interact with parties running $\pi^{sp\mathcal{Q}}$. To mimic the communication in $\pi^{sp\mathcal{Q}}$, $\mathcal{S}_\mathcal{D}$ runs internally all the copies of $\mathcal{S}$ according to the PIDs set $\mathcal{P}$ received from $\mathcal{Z}$. That is, $\mathcal{S}_\mathcal{D}$ runs a copy of $\mathcal{S}$ for each (i, j) where $i \neq j \in \mathcal{P}$. In addition, when output generated by any internal adversary, $\mathcal{S}_\mathcal{D}$ passes this output to $\mathcal{Z}$, together with the identity of the sender. Any input message given by $\mathcal{Z}$ is forwarded to an appropriate internal adversary according to the PIDs involved. When the simulated copy of $\mathcal{S}$ wish to give input m to $\mathcal{F}$ then $\mathcal{S}_\mathcal{D}$ gives $(m, (i, j))$ to $sp\mathcal{F}$. Similarly, any output generated by $sp\mathcal{F}$ is copied to an appropriate internal adversary.

The validity of $\mathcal{S}_\mathcal{D}$ can be inferred by the standard technique of assuming an existence of distinguishing environment $\mathcal{Z}$ and constructing a distinguishing environment $\mathcal{Z}_\mathcal{F}$ as done in 3.5. □

Corollary 1. *[Realizing functionalities] Let $\mathcal{F}$ and $\mathcal{Q}$ be functionalities in the LUC model and π be some $\mathcal{Q}$-hybrid protocol. If $\pi^\mathcal{Q}$ LUC-realizes $\mathcal{F}$ then $\pi^{sp\mathcal{Q}}$ UC-realizes $sp\mathcal{F}$ with respect to static PIDs environments.*

Here we omit the proof since it uses similar proof technique to Theorem 4.

Now we are ready to show that the converse of Theorem 1 does not hold. In fact, we show a stronger result. Not only that there exist protocols that are UC secure and not LUC secure, but also that there exist functionalities that are UC realizable and not LUC realizable. In other words:

Lemma 1. *There exist functionalities $\mathcal{F}$ in the LUC model such that $sp\mathcal{F}$ is UC-realizable by a plain protocol, but $\mathcal{F}$ cannot be LUC realized by a plain protocol.*

Sketch of Proof Recall that the LUC model does not allow the adversaries to coordinate. Intuitively, any LUC functionality which supplies the adversaries with nontrivial common knowledge among corrupted parties would not be realizable without communication. We formalize this intuition by presenting such a functionality. Consider the following 2-party functionality $\mathcal{F}$:

$\mathcal{F}$ running with parties P_1, P_2 and adversaries $\mathcal{A}_{(1,2)}$,$\mathcal{A}_{(2,1)}$. Init $I_1 = 0$

- Upon receiving an input I_1 from P_1 record I_1 and send it to $\mathcal{A}_{(1,2)}$.
- Upon receiving an input (change_input, ,I_1') from $\mathcal{A}_{(1,2)}$, if no output was yet given change the recorded value to I_1'.
- Upon receiving an input (deliver_output) from $\mathcal{A}_{(2,1)}$ send (output,I_1) to P_2

It can be verified that the splitter version of this functionality $sp\mathcal{F}$ can be securely realized by a plain protocol (i.e., without any setup) that simply instructs P_1 to send I_1 to P_2.

Now we show that $\mathcal{F}$ is not securely realizable in LUC. Assume existence of plain protocol π and consider the following environment $\mathcal{Z}$: Instead of invoking P_1 the environment honestly playing the role P_1 in π.

In the real world the output of P_2 is the value used by $\mathcal{Z}$ as P_1's input. Since $\mathcal{F}$ do not know the value used by $\mathcal{Z}$ in the communication with P_2 the output of P_2 in the ideal word will be easily distinguishable. □

5.2 Equivalence and merger functionalities

In order to reason about UC security in the LUC model we need to cast UC functionalities in the LUC framework. This transformation is done by a wrapper functionality, which we call merger functionality. the merger functionality designed to compensate for the difference between the models and enable consistency between the local adversaries.

Definition 5. *[Merger functionality]Let $\mathcal{F}$ be a functionality in the UC model. The merger variant of $\mathcal{F}$, denoted by $m\mathcal{F}$, behaves the same as $\mathcal{F}$ with the following differences in the interface:*

1. *Upon receiving an input* `in` *from some external adversary $\mathcal{A}_{(i,j)}$, $m\mathcal{F}$ behaves as $\mathcal{F}$ would upon receiving input* `in` *adversary $\mathcal{A}$.*
2. *Whenever $\mathcal{F}$ generate output* `out` *to the adversary, $m\mathcal{F}$ gives* `out` *to all external adversaries (in increasing order of PIDs).*
3. *Upon receiving input* `(Deliver,m, (`ℓ`,k))` *from some external adversary $\mathcal{A}_{(i,j)}$, $m\mathcal{F}$ outputs* `(Delivered,m, (i,j))` *to $\mathcal{A}_{(\ell,k)}$.*

This transformation is similarly extended to protocols. Let π be a UC protocol in the $\mathcal{Q}$-hybrid model. The analog LUC protocol of π, is a protocol in the $m\mathcal{Q}$-hybrid model where all instructions in π to communication with $\mathcal{Q}$ are replaced with instructions to communicate with $m\mathcal{Q}$.

First, we show the following relation between merger and splitter functionalities:

Claim. Let $\mathcal{Q}$ be a functionality in the UC model and $\mathcal{Q}'$ be the splitter functionality of $m\mathcal{Q}$. Then $\mathcal{Q}'$ and $\mathcal{Q}$ are equivalent with respect to static PIDs environment.

Sketch of Proof Here we need to show that $\mathcal{Q}'$ UC-realizes $\mathcal{Q}$ and vice versa. Let $\mathcal{S}$ be an adversary participating in $\mathrm{IDEAL}_{\mathcal{Q}'}$. We construct a simulator $\mathcal{S}^*$ with access to $\mathcal{Q}$ such that no environment can distinguish whether it is communicating with $\mathcal{S}$ and $\mathrm{IDEAL}_{\mathcal{Q}'}$ or with $\mathcal{S}^*$ and $\mathrm{IDEAL}_{\mathcal{Q}}$. The idea of the construction is that, given access to Q, the simulator $\mathcal{S}^*$ can run the splitter functionality of $m\mathcal{Q}$ by itself. More formally, upon receiving a set of PIDs $\mathcal{P}$, $\mathcal{S}^*$ runs internally $\mathcal{S}$ on input $\mathcal{P}$. In addition:

All inputs from $\mathcal{Z}$ are forwarded to $\mathcal{S}$ and all outputs of $\mathcal{S}$ are forwarded to $\mathcal{Z}$. whenever $\mathcal{S}$ gives input `(m, (i, j))` to $\mathcal{Q}'$ then $\mathcal{S}^*$ gives `m` to $\mathcal{Q}$. any output `out` received from $\mathcal{Q}$ is transferred to $\mathcal{S}$ as `(out, (i, j))` for all ordered pairs of $i \neq j$ such that $i, j \in \mathcal{P}$.

We omit the validity proof due to resemblance to previous results. The proof in the other direction is done similarly. □

Now we are ready to prove the following equivalence:

Theorem 5. *Let $\mathcal{F}, \mathcal{Q}$ be functionalities in the UC model and let π be some $\mathcal{Q}$-hybrid protocol. Then $\pi^{\mathcal{Q}}$ UC-realizes $\mathcal{F}$ with respect to static PIDs environments if and only if $\pi^{m\mathcal{Q}}$ LUC-realizes $m\mathcal{F}$.*

Sketch of Proof First, we show that UC realization of $\mathcal{F}$ implies LUC realization of $m\mathcal{F}$. Let π be a $\mathcal{Q}$-protocols such that $\pi^{\mathcal{Q}}$ UC-realizes $\mathcal{F}$. We wish to construct an adversary $\mathcal{S}$ so that no $\mathcal{Z}$ will be able to tell whether it is interacting with $\pi^{m\mathcal{Q}}$ and the dummy adversary $\mathcal{D}$ or with $m\mathcal{F}$ and $\mathcal{S}$. That is, for any $\mathcal{Z}$, $\mathcal{S}$ should satisfy

$$\mathrm{LEXEC}_{\pi^{\mathrm{m}\mathcal{Q}},\mathcal{D},\mathcal{Z}} \approx \mathrm{LEXEC}_{\mathrm{m}\mathcal{F},\mathcal{S},\mathcal{Z}} \tag{10}$$

The general outline of the proof is similar to the proof of Theorem 1. The fact that $\pi^{\mathcal{Q}}$ realizes $\mathcal{F}$ guarantees that there exists an adversary $\mathcal{S}_{\mathcal{D}}$, such that for any environment $\mathcal{Z}_{\mathcal{F}}$ we have:

$$\mathrm{EXEC}_{\pi^{\mathcal{Q}},\mathcal{D},\mathcal{Z}_{\mathcal{F}}} \approx \mathrm{EXEC}_{\mathcal{F},\mathcal{S}_{\mathcal{D}},\mathcal{Z}_{\mathcal{F}}} \tag{11}$$

The adversary $\mathcal{S}$ is constructed out of the adversary $\mathcal{S}_{\mathcal{D}}$. The main idea is that the simulator copies are able to jointly run $\mathcal{S}_{\mathcal{D}}$ by using the communication service offered by the merger functionality. More precisely, the simulators mutually run $\mathcal{S}_{\mathcal{D}}$ in the following way: Let $(\mathrm{p}_1, \mathrm{p}_2)$ be some specific ordered PID pair among the PIDs chosen by $\mathcal{Z}$ (for convenience we take the smallest pair).

The simulator $\mathcal{S}$ operates as follows: any `(Deliver,m, (ℓ,k))` instruction from $\mathcal{Z}$ is forwarded to $m\mathcal{F}$ and any output `(Delivered,m, (i,j))` from $m\mathcal{F}$ is outputted to $\mathcal{Z}$. All other messages are treated as follows. If $\mathcal{S}$ identity is not $(\mathrm{p}_1, \mathrm{p}_2)$ then it simply forwards any input from the environment to $\mathcal{S}_{(\mathrm{p}_1,\mathrm{p}_2)}$ via $m\mathcal{F}$, and pass any message from $\mathcal{S}_{(\mathrm{p}_1,\mathrm{p}_2)}$ to the environment. Any output from $m\mathcal{F}$ is ignored. The "special" copy of $\mathcal{S}$, denoted $\mathcal{S}_{(\mathrm{p}_1,\mathrm{p}_2)}$ runs internally $\mathcal{S}_{\mathcal{D}}$ and give it $\mathcal{P}$; In addition,

1. $\mathcal{S}_{(\mathrm{p}_1,\mathrm{p}_2)}$ forwards any input from the environment and any message received from other simulators to the internal $\mathcal{S}_{\mathcal{D}}$.
2. Any output of $\mathcal{S}_{\mathcal{D}}$ is forwarded as a deliver message through $m\mathcal{F}$ to the appropriate simulator according to the PIDs involved.
3. Whenever the simulated $\mathcal{S}_{\mathcal{D}}$ wish to give input `in` to $\mathcal{F}$, $\mathcal{S}_{(\mathrm{p}_1,\mathrm{p}_2)}$ gives this input to $m\mathcal{F}$. Similarly, all output of $m\mathcal{F}$ are given to $\mathcal{S}_{\mathcal{D}}$.

We omit the validity proof due to resemblance with previous results.

For the derivation in the other direction we use Theorem 1 and Claim 5.2 to conclude that $\pi^{\mathcal{Q}}$ UC-realizes $\mathcal{F}$ with respect to static PIDs environments. □

5.3 Relation among notions in presence of global setup

In this section we present the relations between the LUC framework and the UC framework; however these results do not consider global setup. In order for this to be applicable in a presence of global functionalities, we extend the Theorem 5 to the global model. However, before presenting the equivalence theorem, we need to define with respect to what global functionalities the equivalence hold.

Augmented CRS Functionality. The augmented CRS functionality $\bar{\mathcal{G}}_{\mathrm{acrs}}$, presented in [CDPW07], is a global functionality, and offers a one-time use interactive "key retrieval" service to those who choose to use it. In particular, the functionality only allows corrupt parties to retrieve their keys. Thus, we are assured that honest parties need never communicate interactively with $\bar{\mathcal{G}}_{\mathrm{acrs}}$. Here, we extend the augmented CRS functionality to the LUC framework as follows. The functionality $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$ behaves the same as $\bar{\mathcal{G}}_{\mathrm{acrs}}$ with the following difference in the adversarial interface: any retrieve or CRS request done by some party is notified to all its adversaries. As in the [CDPW07] formulation, the functionality $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$ is parametrized by two functions, Setup and $\mathrm{Extract}$. We note that $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$ is not the merger functionality of $\bar{\mathcal{G}}_{\mathrm{acrs}}$. The functionality $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}^{\mathrm{Setup},\mathrm{Extract}}$ is formally presented in Figure 5.

Functionality $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}^{\mathrm{Setup},\mathrm{Extract}}$

- **Initialization Phase:** At the first activation, compute a CRS $(PK) \leftarrow \mathrm{Setup}(MSK)$ for a randomly chosen λ-bit value MSK, and record the pair (PK, MSK).
- **Providing the public value:** Whenever activated, by a party with party identity PID requesting the CRS, return PK to the requesting party and to all PID related adversaries (in increasing order).
- **Dormant Phase:** Upon receipt of a message $(\mathrm{retrieve}, \mathrm{sid}, \mathrm{PID})$ from an adversary of corrupt party identity , return the value $SK_P \leftarrow \mathrm{Extract}(PK; P; MSK)$ to all PID related adversaries (Receipt of this message from honest parties is ignored).

Fig. 5: The Identity-Based Augmented CRS Functionality $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}^{\mathrm{Setup},\mathrm{Extract}}$

The modification of a protocol interacting with a global UC setup to a protocol interacting with a global LUC setup is done as in the merger case.

Theorem 6. *Let $\mathcal{F}$ be a functionality in the UC model and π be some $\bar{\mathcal{G}}_{\mathrm{acrs}}$-hybrid protocol. Then π UC-realizes $\mathcal{F}$ with respect to static PIDs environments if and only if $\bar{\pi}$ LUC-realizes the merger functionality $m\mathcal{F}$ in the $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$-hybrid model.*

Sketch of Proof The proof is similar to the proof presented in 5 with an additional responsibility of the constructed simulator to handle the input/output interface between the internal simulator and the global setup. This is done in a straight forward manner by having the simulator to behave as a router between the internal adversary and the global setup. That is, the simulator in both directions is the same as in proof 5, and in addition, the simulator forwards externally (to the global setup) any input addressed to this setup, and forwards to the appropriate internal simulator any output of the global setup. We show more formally how to simulate the interaction with the global functionalities.

First, we show that UC realization of $\mathcal{F}$ implies LUC realization of $m\mathcal{F}$ with global setup. Let $\mathcal{S}$ be the LUC simulator constructed in theorem 5 for the dummy adversary, where the minimal identity copy of $\mathcal{S}$ internally run the simulator promised by the UC realization (denoted by $\mathcal{S}_{\mathcal{D}}$); we refer this copy as "master" and all the other instances as "slave". We add to $\mathcal{S}$ the following behavior: the slave copies forward to the master any output received from the setup, and execute all master's instructions. Recall that $\mathcal{S}_{\mathcal{D}}$ expect to interact with $\bar{\mathcal{G}}_{\mathrm{acrs}}$. The only difference between $\bar{\mathcal{G}}_{\mathrm{acrs}}$ and $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$ is that the letter notifies all party related adversaries upon CRS retrieval. Therefore, upon receiving notification from all party related simulators, $\mathcal{S}$ forward a notification to $\mathcal{S}_{\mathcal{D}}$ as if it outputted by $\bar{\mathcal{G}}_{\mathrm{acrs}}$. Except the above, any message of $\mathcal{S}_{\mathcal{D}}$ is forwarded to the appropriate external simulator by sending a $(\mathrm{DELIVER}, ...)$ to $m\mathcal{F}$ and any received messages is forwarded internally to $\mathcal{S}_{\mathcal{D}}$.

To show that LUC realization of $m\mathcal{F}$ implies UC realization of $\mathcal{F}$ with global setup we take the simulator constructed in 5 and extend it to simulate interaction with global setup. Here, the constructed simulator runs internally all the copies of the LUC simulator according to the identity set $\mathcal{P}$ received from the static PIDs UC environment. Any output from the setup is forwarded to the appropriate internal simulator, and any input request to the setup is executed externally. We have the following exception: the notification upon CRS retrieval, which is internally forwarded to all the adversaries of the party retrieved the CRS.

The validity of the constructions in both directions can be inferred by the standard technique of assuming an existence of distinguishing environment $\mathcal{Z}$ and constructing a distinguishing environment $\mathcal{Z}_{\mathcal{F}}$ that breaks the security of the protocol that guaranteed to be secure. □

We note that Theorem 6 is general and can be reproved with respect to other global functionalities. Nonetheless, having the compiler of [CDPW07] in mind, it seems suffices to consider the augmented CRS functionality.

6 Applicability of LUC security

In this section we present the inability of known frameworks to capture various flavors of anonymity, deniability, and confinement. Moreover, we show that the LUC framework is not subject to these weaknesses. Furthermore, we (a) present a simulation-based notion of Bi-deniability and show it is tightly captured by LUC, and (b) show how LUC security can be used to capture confinement and rationality in general.

6.1 Anonymity

The timing, existence and sender anonymity were informally presented in the introduction. recall that in the introduction, these concerns are presented via devices such as dropbox, email future, and trusted coordinator; but in fact these are cryptographic channels guaranteeing anonymity in the subject matter. We present ideal functionalities, which are the formalizations of these channels, and realization by non-trivial protocols that do not provide anonymity. We remark that in absence of any formal definition, we can only show that these protocols do not satisfy our intuitive perception of anonymity. Here, we present

the inability of the UC and Collusion-Preserving notion (referred as CP) to capture the intuitive idea of anonymity. (For the formal definition of CP we refer the reader to [AKMZ12].)

6.1.1 Existence-anonymity

Here our goal is to model a sender-receiver channel, denoted by existence-anonymous channel, that has a strong anonymity guarantee regarding the existence of a sender. The existence-channel always allows the receiver to retrieve a message. However, in absence of a sender this message will be some randomly chosen message. The LUC existence-anonymous channel $\mathcal{F}_{\overline{\text{EA}}}$ is formally presented in Figure 6.

Functionality $\mathcal{F}^{D}_{\overline{\text{EA}}}$

Functionality $\mathcal{F}^{D}_{\overline{\text{EA}}}$ runs with parties S, R, adversaries $\mathcal{S}_{(S,R)}$, $\mathcal{S}_{(R,S)}$ and parametrized on message distribution D. It proceeds as follows:

- Upon receiving an input (Send, sid, m) from party S do: verify that $sid = (S, R, sid')$, else ignore the input. Next, record m, and send output (Send, sid, m) to $\mathcal{S}_{(S,R)}$. Ignore any subsequent $(\text{Send}, ...)$ inputs. Once $\mathcal{S}_{(S,R)}$ allows to forward the message mark m as approved.
- Upon receiving an input (Output, sid) from party R do:
  1. if there is an approved message m then set $\text{OUT} = m$; else set $\text{OUT} \leftarrow D$.
  2. send output (OUT, sid, m) to $\mathcal{S}_{(R,S)}$; Once $\mathcal{S}_{(R,S)}$ allows to forward the message output (OUT, sid, m) to R and halt.

Fig. 6: The existence-anonymous channel functionality $\mathcal{F}_{\overline{\text{EA}}}$

The underlying communication model is a channel called $\mathcal{F}_{\overline{\text{EB}}}$ that is similar to authentication channel with a difference in the message delivery. More specifically, a message is delivered to the recipient upon recipient's request and only if there exists a message sent to him. The LUC channel $\mathcal{F}_{\overline{\text{EB}}}$ is formally presented in Figure 7.

Functionality $\mathcal{F}_{\overline{\text{EB}}}$

Functionality $\mathcal{F}_{\overline{\text{EB}}}$ runs with parties S, R, and adversaries $\mathcal{S}_{(S,R)}$, $\mathcal{S}_{(R,S)}$. Initialize $\text{OUT} = \perp$ and proceed as follows:

- Upon receiving an input (Send, sid, m) from party S do: verify that $sid = (S, R, sid')$, else ignore the input. Next, record m, and send output (Send, sid, m) to $\mathcal{S}_{(S,R)}$. Ignore any subsequent $(\text{Send}, ...)$ inputs. Once $\mathcal{S}_{(S,R)}$ allows to forward the message mark m as approved.
- Upon receiving an input (Output, sid) from party R and there is an approved message m do:
  1. set $\text{OUT} = m$, and output (OUT, sid, m) to $\mathcal{S}_{(R,S)}$; Once $\mathcal{S}_{(R,S)}$ allows to forward the message output (OUT, sid, m) to R and halt.

Fig. 7: The basic existence-channel functionality $\mathcal{F}_{\overline{\text{EB}}}$

In order to reason about UC security we consider the analogous UC functionalities as in section 4, namely, the splitter functionalities of $\mathcal{F}_{\overline{\text{EA}}}$ and $\mathcal{F}_{\overline{\text{EB}}}$ (denoted by $\mathcal{F}_{\text{EA}}$ and $\mathcal{F}_{\text{EB}}$). The resulting UC functionality $\mathcal{F}_{\text{EA}}$ still guarantees existence-anonymity since the splitter transformation does not introduce any changes in the logic of a functionality and only replaces the multiple adversaries interface with an equivalent single adversary interface.

Claim. There exists a protocol that UC-realizes $\mathcal{F}_{\text{EA}}$ and is not a LUC-realization of the analogous functionality $\mathcal{F}_{\overline{\text{EA}}}$.

Proof. We show that $\text{IDEAL}_{\mathcal{F}_{\text{EB}}}$ UC realizes $\mathcal{F}_{\text{EA}}$, where the only important task of the simulator is to ignore recipient's output request where there is no message from the sender. More formally, let $\mathcal{D}$ be the dummy adversary that interacts with parties running $\text{IDEAL}_{\mathcal{F}_{\text{EB}}}$. We construct an adversary $\mathcal{S}$ for the ideal process for $\mathcal{F}_{\text{EA}}$ such that no environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{D}$ and the above protocol or with $\mathcal{S}$ in the ideal process. Simulator $\mathcal{S}$ behaves as $\mathcal{D}$ except when $\mathcal{Z}$ instructs it to approve output request where there is no marked message from the sender. In this case, the instruction is ignored by $\mathcal{S}$. Note that this happens also in the execution of $\text{IDEAL}_{\mathcal{F}_{\text{EB}}}$, when there is no message waiting to be approved by $\mathcal{D}$; here $\mathcal{F}_{\text{EB}}$ ignores all adversarial inputs.

In order to show that $\text{LIDEAL}_{\mathcal{F}_{\overline{\text{EB}}}}$ is not a LUC realization of $\mathcal{F}_{\overline{\text{EA}}}$ we consider two environments $\mathcal{Z}_1$ and $\mathcal{Z}_2$ such that for any simulator $\mathcal{S}$ one of two environments can distinguish between an execution of $\text{LIDEAL}_{\mathcal{F}_{\overline{\text{EB}}}}$ with the dummy adversary and $\text{LIDEAL}_{\mathcal{F}_{\overline{\text{EA}}}}$ with $\mathcal{S}$. Let $\mathcal{Z}_1$ be the environment that first gives S input (Send, sid, m) and instructs $\mathcal{S}_{(S,R)}$ to allows to forward the message to R, and then gives R input (Output, sid). Let $\mathcal{Z}_2$ be an environment that only gives R input (Output, sid).

It can be easily verified that any simulation of output request is distinguishable from the execution of $\text{LIDEAL}_{\mathcal{F}_{\overline{\text{EB}}}}$ either by $\mathcal{Z}_1$ or by $\mathcal{Z}_2$. This is so since the simulator of the recipient $\mathcal{S}_{(R,S)}$ does not have the knowledge of whether the output message sent by the sender or randomly generated by $\mathcal{F}_{\overline{\text{EA}}}$, and therefore, it cannot correctly decide whether to ignore the (Output, sid) input.

It is important to note that $\text{IDEAL}_{\mathcal{F}_{\text{EB}}}$ does not provide existence-anonymity since the recipient is guaranteed that any message received was sent by the sender. We remind the reader that there is no formal definition of existence-anonymity to be found, and therefore, we refer to the intuitive notion of it.

6.1.2 Timing-anonymity

Here, our goal is to define a channel that guarantees to the sender that no receiver, upon receiving a message from him, can tell when this message was sent. As mentioned in the introduction, we define a timing-anonymous channel, denoted by $\mathcal{F}_{\overline{\text{TA}}}$, that randomly delay a message in a way that the amount of the delay is unknown to the receiver. In particular, the message is delivered only after a certain delay. The LUC channel $\mathcal{F}_{\overline{\text{TA}}}$ is formally presented in Figure 8.

Now we formally define the underlying model. In order to capture time, we introduce a clock functionality $\mathcal{F}_{\text{clock}}$ that is observable by all participants. This clock is not directly observed by the environment, instead, it is indirectly advanced by the environment, by instructing the sender to advance the clock; this captures a setting in which the receiver's future actions are not affected by the amount of the delay. The formal description of $\mathcal{F}_{\text{clock}}$ presented in Figure 9. The second component is the authentication functionality $\mathcal{F}_{\text{auth}}$. The difference between the LUC and the UC authentication functionality is that the LUC functionality, denoted by $\mathcal{F}_{\overline{\text{auth}}}$, operates not only when a message is sent. That is, it allows the adversary associated with the sender to approve delivery even when no message was sent; in this case $\mathcal{F}_{\overline{\text{auth}}}$ outputs $\perp$ to receiver's adversary and halts. The LUC authentication functionality $\mathcal{F}_{\overline{\text{auth}}}$ is formally presented in Figure 10.

Functionality $\mathcal{F}_{\overline{\text{TA}}}$

Functionality $\mathcal{F}_{\overline{\text{TA}}}$ runs with parties S, R, and adversaries $\mathcal{S}_{(S,R)}$, $\mathcal{S}_{(R,S)}$. Let $\mathcal{T}$ be some finite set of natural numbers. The functionality proceeds as follows:

- Upon receiving an input (Send, sid, m) from party S do: verify that $sid = (S, R, sid')$, else ignore the input. Next, choose uniformly at random $N \leftarrow \mathcal{T}$, set $k = N$, and record (k, m). Ignore any subsequent $(\text{Send}, ...)$ inputs.
- Upon receiving an input $(\text{Advance}, sid)$ from party S and $k > 0$ do:
  1. update $k = k - 1$ and output $(\text{Advance}, sid)$ to adversary $\mathcal{S}_{(S,R)}$.
  2. if $k = 0$ output m to $\mathcal{S}_{(S,R)}$. Once $\mathcal{S}_{(S,R)}$ allows to forward the message output m to $\mathcal{S}_{(R,S)}$. Once also $\mathcal{S}_{(R,S)}$ allows to forward m output it to R.

1. Upon receiving $(\text{Corruptsend}, sid, m')$ from $\mathcal{S}_{(S,R)}$, if S is corrupt and m has not been delivered to $\mathcal{S}_{(R,S)}$, then change the recorded message to m'.

Fig. 8: The timing-anonymous channel functionality $\mathcal{F}_{\overline{\text{TA}}}$

Functionality $\mathcal{F}_{\overline{\text{clock}}}$

Functionality $\mathcal{F}_{\overline{\text{clock}}}$ runs with parties S, R, and adversaries $\mathcal{S}_{(S,R)}$, $\mathcal{S}_{(R,S)}$. Initialize $T = 0$. Next:

- Upon receiving an input $(\text{Advance}, sid)$ from party S do: in the first activation verify that $sid = (S, R, sid')$, else ignore the input. Next, set $T = T + 1$.
- Upon receiving an input (time, sid) from some party, output (time, sid, T).

Fig. 9: The clock functionality $\mathcal{F}_{\overline{\text{clock}}}$

We note that $\mathcal{F}_{\overline{\text{auth}}}$ provides stronger guarantees and it seems as a natural relaxation of the UC authentication functionality.

Let $\mathcal{F}_{\text{TA}}$, $\mathcal{F}_{\text{clock}}$ and $\mathcal{F}_{\text{auth}}$ be the analogous UC functionalities obtained by the splitter wrapper as presented in 4. The splitter transformation do not change the guarantees of the timing-anonymous channel, and the UC functionality $\mathcal{F}_{\text{TA}}$ still provides timing-anonymity.

Claim. There exists a protocol π_{TA} that UC-realizes $\mathcal{F}_{\text{TA}}$ in the $(\mathcal{F}_{\text{auth}}, \mathcal{F}_{\text{clock}})$-hybrid model. Moreover, π_{TA} is not a LUC-realization of the analogous functionality $\mathcal{F}_{\overline{\text{TA}}}$.

Proof. The protocol π_{TA} for UC realization of $\mathcal{F}_{\text{TA}}$ is presented in Figure 11.

First, we show that π_{TA} UC-realizes the timing-anonymous functionality $\mathcal{F}_{\text{TA}}$ in the $(\mathcal{F}_{\text{clock}}, \mathcal{F}_{\text{auth}})$-hybrid model. The main goal of the simulator is to provide, upon a request from the environment, the current time as provided by $\mathcal{F}_{\text{clock}}$ in the execution of π_{TA}. More formally, we construct a simulator $\mathcal{S}$ for simulating the protocol execution with the dummy adversary $\mathcal{D}$ and environment $\mathcal{Z}$. The simulator $\mathcal{S}$ behaves as follows. The simulator $\mathcal{S}$ counts the (Advance) notifications from $\mathcal{F}_{\text{TA}}$, and if instructed by $\mathcal{Z}$ to provide the current time, $\mathcal{S}$ outputs this counter. Other that, $\mathcal{S}$ behaves as the dummy adversary $\mathcal{D}$ by forwarding any output of $\mathcal{F}_{\text{TA}}$ to $\mathcal{Z}$ and all inputs coming from $\mathcal{Z}$ to $\mathcal{F}_{\text{TA}}$.

By considering this protocol in LUC, we note that no simulator for the receiver can tell the time provided by $\mathcal{F}_{\overline{\text{clock}}}$ in the real execution of π_{TA}; this happens since $\mathcal{S}_{(R,S)}$ is oblivious to any advancement of time in $\mathcal{F}_{\overline{\text{TA}}}$. Hence, π_{TA} is not a LUC-realization of $\mathcal{F}_{\overline{\text{TA}}}$ in the $(\mathcal{F}_{\overline{\text{auth}}}, \mathcal{F}_{\overline{\text{clock}}})$-hybrid model.

Functionality $\mathcal{F}_{\overline{\text{auth}}}$

Functionality $\mathcal{F}_{\overline{\text{auth}}}$ runs with parties S, R, and adversaries $\mathcal{S}_{(S,R)}$, $\mathcal{S}_{(R,S)}$. It proceeds as follows:

1. Upon receiving an input (Send, sid, m) from party S, do: If $sid = (S, R, sid')$ for some R, then record m and output (Send, sid, m) to $\mathcal{S}_{(S,R)}$.
2. Upon receiving "approve" from $\mathcal{S}_{(S,R)}$, if m is recorded provide(Send, sid, m) to $\mathcal{S}_{(R,S)}$, and after $\mathcal{S}_{(R,S)}$ approves, output (Send, sid, m) to R and halt. Otherwise, provide$(\text{Send}, sid, \perp)$ to $\mathcal{S}_{(R,S)}$ and halt. (Both adversaries control the channel delay.)
3. Upon receiving $(\text{Corruptsend}, sid, m')$ from $\mathcal{S}_{(S,R)}$, if S is corrupt and m was not yet delivered to $\mathcal{S}_{(R,S)}$, then output(Send, sid, m') to $\mathcal{S}_{(R,S)}$, and after $\mathcal{S}_{(R,S)}$ approves, output (Send, sid, m') to R and halt.

Fig. 10: The message authentication functionality $\mathcal{F}_{\overline{\text{auth}}}$

Protocl π_{TA}

Let $\mathcal{T}$ be some finite set of natural numbers.

- INPUT: Having received input (Send, sid, m), S chooses uniformly at random $N \leftarrow \mathcal{T}$, set $k = N$, and records (k, m).
- ADVANCE: Having received input (Advance, sid), S forward it to $\mathcal{F}_{\text{clock}}$ and updates $k = k - 1$. Once $k = 0$ send m to $\mathcal{F}_{\text{auth}}$ and halt.
- OUTPUT: Having received (OUT, sid, m) from $\mathcal{F}_{\text{auth}}$, the receiver R outputs m.

Fig. 11: The protocol π_{TA}

We note that π_{TA} does not provide timing anonymity since all participants in the protocol observe the clock. In particular, upon receiving a message, the receiver can retrieve the time by sending (time,sid) to $\mathcal{F}_{\text{clock}}$ and knows exactly when the message was sent.

6.1.3 Sender-anonymity

The sender anonymity property is presented in the introduction via a trusted mediator that masks the identity of the sender. This mediator is similar to the two-anonymous channels of [NMO08]; however, their formalism is not applicable in our setting. The channel enables two senders and a receiver to communicate anonymously in the following sense: both senders may send a message to the receiver but only one message is delivered. This sender-anonymous channel, denoted by $\mathcal{F}_{\overline{\text{SA}}}$, does not disclose the identity of the actual sender. The LUC formulation of $\mathcal{F}_{\overline{\text{SA}}}$ is presented in Figure 12.

The underlying communication channel, denoted by $\mathcal{F}_{\overline{\text{S1}}}$, is a two-sender one receiver channel that delivers only messages sent by the first sender S1. The formal description of $\mathcal{F}_{\overline{\text{S1}}}$ is presented in Figure 13.

This time too, we transform $\mathcal{F}_{\overline{\text{SA}}}$ and $\mathcal{F}_{\overline{\text{S1}}}$ to the UC framework by applying the splitter transformation, and denote by $\mathcal{F}_{\text{SA}}$ and $\mathcal{F}_{\text{S1}}$ the analogous UC functionalities. Here, the splitter transformation maintains the anonymity property of $\mathcal{F}_{\overline{\text{SA}}}$ for the same reasons as in the previous anonymity concerns.

Claim. There exists a protocol that UC-realizes $\mathcal{F}_{\text{SA}}$ and is not a LUC-realization of the analogous functionality $\mathcal{F}_{\overline{\text{SA}}}$.

Functionality $\mathcal{F}_{\overline{\text{SA}}}$

Functionality $\mathcal{F}_{\overline{\text{SA}}}$ running with parties S_1, S_2 R, and adversaries $\mathcal{S}_{(S_1,R)}$, $\mathcal{S}_{(S_2,R)}$, $\mathcal{S}_{(R,S_1)}$, $\mathcal{S}_{(R,S_2)}$. At first activation verify that $sid = (S_1, S_2, R, sid')$, else halt. Next, proceed as follows:

– Upon receiving an input (Send, sid, m_i) from party S_i do: record m_i, and send output (Send, sid, m_i) to $\mathcal{S}_{(S_i,R)}$. Ignore any subsequent $(\text{Send}, ...)$ inputs from S_i. Once $\mathcal{S}_{(S_i,R)}$ allows to forward the message output (Send, sid, m_i) to $\mathcal{S}_{(R,S_i)}$. Once approved, output (Send, sid, m_i) to R and halt.

Fig. 12: The sender-anonymous channel functionality $\mathcal{F}_{\overline{\text{SA}}}$

Functionality $\mathcal{F}_{\overline{\text{S1}}}$

Functionality $\mathcal{F}_{\overline{\text{S1}}}$ running with parties S_1, S_2 R, and adversaries $\mathcal{S}_{(S_1,R)}$, $\mathcal{S}_{(S_2,R)}$, $\mathcal{S}_{(R,S_1)}$, $\mathcal{S}_{(R,S_2)}$. Initialize variable $\text{BLOCK} = 0$. At first activation verify that $sid = (S_1, S_2, R, sid')$, else halt. Next, proceed as follows:

– Upon receiving an input (Send, sid, m_1) from party S_1 do: record m_1, and send output (Send, sid, m_1) to $\mathcal{S}_{(S_1,R)}$. Ignore any subsequent $(\text{Send}, ...)$ inputs from S_1. Once $\mathcal{S}_{(S_1,R)}$ allows to forward the message output (Send, sid, m_1) to $\mathcal{S}_{(R,S_1)}$. Once approved, if $\text{BLOCK} = 0$ output (Send, sid, m_1) to R and halt; else halt.
– Upon receiving an input (Send, sid, m_2) from party S_2 do: record m_2, and send output (Send, sid, m_2) to $\mathcal{S}_{(S_2,R)}$. Ignore any subsequent $(\text{Send}, ...)$ inputs from S_2. Once $\mathcal{S}_{(S_2,R)}$ allows to forward the message output (Send, sid, m_2) to $\mathcal{S}_{(R,S_2)}$. Once approved set $\text{BLOCK} = 1$.

Fig. 13: The basic sender-channel functionality $\mathcal{F}_{\overline{\text{S1}}}$

Proof. We show that $\text{IDEAL}_{\mathcal{F}_{\text{S1}}}$ is a UC realization of the sender-anonymous functionality $\mathcal{F}_{\text{SA}}$. In order to prove UC realization by $\text{IDEAL}_{\mathcal{F}_{\text{S1}}}$ we construct a simulator $\mathcal{S}$. Let $\mathcal{D}$ be the dummy adversary that interacts with $\text{IDEAL}_{\mathcal{F}_{\text{S1}}}$. We construct simulator $\mathcal{S}$ for the ideal process for $\mathcal{F}_{\text{SA}}$. Recall that the simulator interacts only with the ideal functionality $\mathcal{F}_{\text{SA}}$ and with the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ initializes $\text{BLOCK} = 0$. Next, it proceeds as follows:

1. The simulation of the senders is done by simply behaving as a dummy adversary. Moreover, all outputs from $\mathcal{F}_{\text{SA}}$ are forwarded to $\mathcal{Z}$.
2. Upon receiving an $(\text{Approve}, sid, m)$ instruction from $\mathcal{Z}$, it proceeds as follow: if m was send by S_2 then set $\text{BLOCK} = 1$. if m was send by S_1 and $\text{BLOCK} = 0$ forward $(\text{Approve}, sid, m)$ to $\mathcal{F}_{\text{SA}}$; else ignore.

The validity of $\mathcal{S}$ can be easily verified.

In the context of sender anonymity, the weakness of the centralized adversary modeling is the awareness of the adversary to all the incoming communication of the receiver, and not surprisingly, this is the key ingredient of the UC simulation above. In the LUC framework, each of the receiver's adversaries is exposed only to a partial communication. Indeed, when considering $\text{LIDEAL}_{\mathcal{F}_{\overline{\text{S1}}}}$ in the LUC setting we note that it is impossible to come up with a simulator $\mathcal{S}_{(R,S_1)}$ that induces an indistinguishable view. In particular, upon receiving an $(\text{Approve}, sid, m)$ instruction, $\mathcal{S}_{(R,S_1)}$ does not know whether to ignore (in case there was already "approved" message send by S_2) or forward to it to $\mathcal{F}_{\overline{\text{SA}}}$ (in case it is the first message to be approved) and any decision will not fool the environment. Therefore $\text{LIDEAL}_{\mathcal{F}_{\overline{\text{S1}}}}$ do not LUC-realize $\mathcal{F}_{\overline{\text{SA}}}$.

We note that the channel $\mathcal{F}_{\mathrm{S1}}$ guarantees that any message received via this channel was sent by S_1, and therefore, $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{S1}}}$ do not provide sender-anonymity.

Sender-anonymity in the CP framework. We note that in the context of sender-anonymity, the CP model suffers from the same weakness as the UC model. That is, the above non sender-anonymous protocol is a CP-realization of the sender-anonymous channel $\mathcal{F}_{\mathrm{SA}}$.

we show the CP security of the above protocol. As a first step, we map $\mathcal{F}_{\overline{\mathrm{SA}}}$ and $\mathcal{F}_{\overline{\mathrm{S1}}}$ to the the CP framework by applying the splitter transformation only to the adversarial interface of the receiver. As before, the splitter transformation maintains the anonymity property of $\mathcal{F}_{\overline{\mathrm{SA}}}$. We denote by $\mathcal{F}_{\widehat{\mathrm{SA}}}$ and $\mathcal{F}_{\widehat{\mathrm{S1}}}$ the analogous CP functionalities.

Claim. $\mathcal{F}_{\widehat{\mathrm{S1}}}$ *CP-realizes* $\mathcal{F}_{\widehat{\mathrm{SA}}}$.

The construction of the simulators done as in claim 6.1.3, where the simulators associated with the receivers behave as described in step (1) and the simulator associated with the receiver behaves as described in step (2).

This weakness pertain to the CP model for the same reasons as in the UC model. Namely, the adversary of the receiver is exposed to all of the receiver's incoming communication.

Discussion. It seems that the UC framework does not provide enough means of information separation in the *ideal process*. In the context of anonymity, where parties have contradictory interests, the separation is essential even if all the parties are honest. In particular, any sensitive information (for example, the existence of the sender) should be available only to an entity with the same interests. Indeed, this information is extensively used by the UC simulator and the absence of it prevents from the above protocols to LUC-realize the anonymous channels. Moreover, if we wish to adequately capture anonymity, it seems essential not to grant any adversary in the network with a knowledge that we wish to withhold from parties it is in charge of their communication. This should hold independently of the integrity of these parties.

6.2 Bi-Deniability

In the introduction we informally presented Bi-deniability. Here, we formalize this notion and show that UC security does not capture this flavor of deniability. In fact, this is true also for GUC. Moreover, we define Bi-deniability separately and show equivalence between Bi-deniable authentication and LUC secure authentication. We remark that, although we focus on authentication, Bi-deniability is a general notion that applies to other tasks such as commitments and encryption.

6.2.1 Our definition

Bi-deniability aims to capture the ability of a participant in a two party protocol to deny participation in a protocol execution even if its communication had been externally exposed. The actual definition has some similarities to the definition presented in [DKSW09].

The relevant entities are the following: we have a sender S who is potentially communicating with a receiver R, a judge $\mathcal{J}$ who will eventually rule whether or not the transmission was attempted, two informants $\mathcal{I}_S$, $\mathcal{I}_R$ who witness the communication (represented as log files owned by $\mathcal{I}_S$, $\mathcal{I}_R$) between S and R and are trying to convince the judge, and two misinformants $\mathcal{M}_S$, $\mathcal{M}_R$ who did not witness any communication but still want to convince the judge that one occurred.

The idea of the Bi-deniability definition is that no party should be accused of participating in a protocol, if any evidence presented to the judge (by the informants) based on witnessing the protocol execution can be also presented (by the misinformants) without any communication whatsoever. This idea is formalized via indistinguishability of experiments as follows:

The system in which the experiment takes place may include some trusted incorruptible global setup (to which the judge has access), some means of communication between the sender and receiver, and a direct, private channel between the judge and each (mis)informant. The judge cannot directly communicate with the sender or the receiver; instead it obtains all its information from the informants (or misinformants) and any global setup available. It also cannot view the log files but rather it is informed by the informants (or misinformant) via the direct communication channels.

The means of communication between the sender and receiver determines the communication witnessed by the informants (for example, in the case of ideally secure channels, no communication is witnessed). More formally, the communication observed by the informants is fully determined via the adversarial interface of the communication means between S and R. Moreover, this adversarial interface also determines the capabilities of the informants in the network. In particular, it fully determines the communication interface between the informants. For example, the ideally secure channels do not enable any communication between the informants.

The informants $\mathcal{I}_S$, $\mathcal{I}_R$ and the misinformants $\mathcal{M}_S$, $\mathcal{M}_R$ can communicate with S and R respectively and adaptively corrupt it. By corrupting a party the corrupter learns the entire state of the party. Additionally, once either S or R is corrupt, the judge learns about the corruption. In addition, the corrupter can totally control the actions of this party going forward. Lastly, $\mathcal{M}_S$ is allowed to send a single `(signal)` message to $\mathcal{M}_R$.

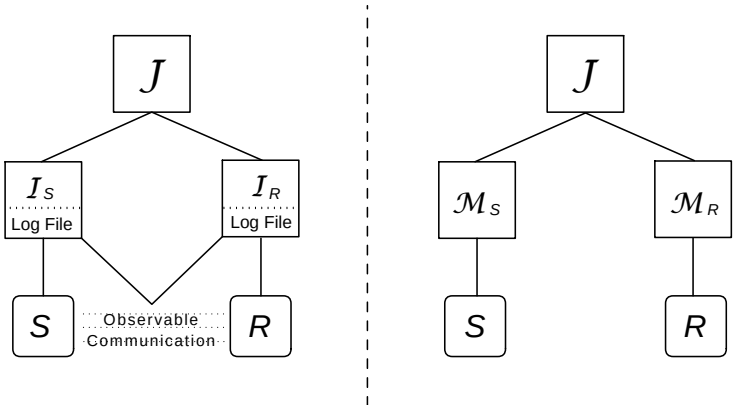

Fig. 14: The Bi-deniability experiments. In the informant experiment (left Figure) the parties run π, while the informants observe this communication. In the misinformant experiment (right Figure) S and R do not communicate. The sender's misinformant is allowed to send a signal (only once) to the receiver's misinformant. In both experiments the judge with the informants (or misinformants) can communicate freely. In addition, the informants (or misinformants) can communicate with the appropriate party and corrupt it.

Now we are ready to present the definition. Let π be some 2-party protocol.

Informant-Experiment. The inputs to parties are given by the judge, and any output produced by the parties is given to the judge. S and R run π in the presence of the informants $\mathcal{I}_S$, $\mathcal{I}_R$. The informants report to $\mathcal{J}$ regarding any observed communication and execute all $\mathcal{J}$'s instruction. The output distribution of the judge $\mathcal{J}$ in this basic informant-experiment is denoted by $\mathrm{EXP}_{\pi,\mathcal{I}_S,\mathcal{I}_R,\mathcal{J}}$.

Misinformant-Experiment. The inputs to parties are also given to the misinformants. S and R do not communicate except with $\mathcal{M}_S$, $\mathcal{M}_R$. The misinformant $\mathcal{M}_S$ can send single `(signal)` message to $\mathcal{M}_R$; in addition, they can freely communicate with $\mathcal{J}$. Any message received by the parties from their misinformant is outputted to the judge. The output distribution of the judge $\mathcal{J}$ in this basic misinformant-experiment is denoted by $\mathrm{EXP}_{\mathcal{M}_S,\mathcal{M}_R,\mathcal{J}}$.

A graphical depiction appears in Figure 14.

Definition 6 (Bi-deniability). *Let π be some PPT protocol and let the informants $\mathcal{I}_S, \mathcal{I}_R$ be as defined above. We say that π is Bi-deniable if there exist PPT misinformants $\mathcal{M}_R$ and $\mathcal{M}_S$ such that for any PPT judge $\mathcal{J}$ we have:*

$$\mathrm{EXP}_{\pi,\mathcal{I}_S,\mathcal{I}_R,\mathcal{J}} \approx \mathrm{EXP}_{\mathcal{M}_S,\mathcal{M}_R,\mathcal{J}}$$

6.2.2 Discussion

We discuss some aspects of Definition 6 and mention some related definitional approaches.

Signaling misinformants. Recall that the authentication functionality (presented in Figure 10) enables one bit of information flow between the adversaries, even when no message was sent. To capture this guarantee, we allow in the Bi-deniability definition to send one time a signal to the receiver's misinformant. Alternatively, we could consider more ideal formulation of the authentication functionality. Namely, an authentication functionality that instead of waiting for the adversaries to approve delivery, it immediately outputs the message. For such a functionality, that do not enable any information flow between the adversaries, the Bi-deniability definition should change accordingly. That is, any communication between the misinformants should be disallowed.

Dummy informants. An important observation is that Bi-deniability significantly different from a general security against any adversary. In opposed to the latter, we do not argue about immunity of a protocol against arbitrary attackers but rather we wish to argue about the ability to deny participation when the actual, true communication is exposed. We note that the ability to deny manipulated communication is a fundamentally different task. Intuitively, it defines the ability to acquit oneself when being accused of uncommitted crime. Here, our goal is to capture the former ability, and therefore, we consider "dummy" informants that report the actual communication as being witnessed by them.

6.2.3 Bi-deniable authentication

In this section we show that the ideal message authentication functionality, $\mathcal{F}_{\overline{\text{auth}}}$ that presented in Figure 10, provides strong deniability guarantees. More formally, we show that the LUC ideal protocol for authentication is Bi-deniable.

Claim. Let $\mathcal{F}_{\overline{\text{auth}}}$ be the LUC authentication functionality. Then the protocol $\mathrm{LIDEAL}_{\mathcal{F}_{\overline{\text{auth}}}}$ is Bi-deniable.

Proof. In order to show Bi-deniability of $\mathrm{LIDEAL}_{\mathcal{F}_{\overline{\text{auth}}}}$ we construct the misinformants $\mathcal{M}_S$ and $\mathcal{M}_R$ such that no $\mathcal{J}$ will be able to tell whether it is interacting with π and the dummy informants $\mathcal{I}_S$, $\mathcal{I}_R$ or with $\mathcal{M}_S$ and $\mathcal{M}_R$. The misinformants construction is presented below:

- having the message m, $\mathcal{M}_S$ forwards it to the judge as if it was received from $\mathcal{F}_{\overline{\text{auth}}}$. Once $\mathcal{J}$ approves to forward the message, $\mathcal{M}_S$ sends `(signal)` message to $\mathcal{M}_R$.
- Upon receiving `(signal)` message from $\mathcal{M}_S$, the misinformant $\mathcal{M}_R$ forwards m to the judge as if it was received from $\mathcal{F}_{\overline{\text{auth}}}$. Once $\mathcal{J}$ approves to forward the message, $\mathcal{M}_S$ gives m to R.

Clearly, the view of $\mathcal{J}$ is identical in both experiments since the communication observed by the informants in $\mathrm{LIDEAL}_{\mathcal{F}_{\overline{\text{auth}}}}$ is only the authenticated message m (which the misinformants also receive). Moreover, using the `(signal)` message the misinformants perfectly simulate the scheduling done in $\mathrm{LIDEAL}_{\mathcal{F}_{\overline{\text{auth}}}}$.

6.2.4 On LUC security and Bi-deniability

In this section we prove that Bi-deniable authentication is equivalent to LUC realization of the authentication functionality $\mathcal{F}_{\overline{\text{auth}}}$ (presented in Figure 10). Although the informant-experiment is identical to the examined protocol execution with dummy adversaries, the "ideal-execution" has many differences. The main difference is the "ideal world experiment" in which the ideal protocol is executed compared to

the misinformant experiment in which no protocol is executed. Intuitively, producing indistinguishable transcript when no communication whatsoever is provided is significantly harder than producing this transcript based on communication of a tightly related protocol. Nonetheless, we show that this gap can be bridged.

Theorem 7. *Let π be some protocol. Then π LUC-realizes $\mathcal{F}_{\overline{\text{auth}}}$ if and only if π is Bi-deniable.*

Proof. First, we show that LUC realization of $\mathcal{F}_{\overline{\text{auth}}}$ implies Bi-deniability. Let π be a protocol that LUC-realizes $\mathcal{F}_{\overline{\text{auth}}}$. We wish to construct misinformants $\mathcal{M}_S$ and $\mathcal{M}_R$ such that no $\mathcal{J}$ will be able to tell whether it is interacting with π and the dummy informants $\mathcal{I}_S$, $\mathcal{I}_R$ or with $\mathcal{M}_S$ and $\mathcal{M}_R$. That is, for any $\mathcal{J}$, $\mathcal{M}_S$ and $\mathcal{M}_R$ should satisfy

$$\text{EXP}_{\pi,\mathcal{I}_S,\mathcal{I}_R,\mathcal{J}} \approx \text{EXP}_{\mathcal{M}_S,\mathcal{M}_R,\mathcal{J}} \tag{12}$$

The general outline of the proof proceeds as follows. The fact that π realizes $\mathcal{F}_{\overline{\text{auth}}}$ guarantees that there exists an adversary (called a simulator) $\mathcal{S}_{\mathcal{D}}$, such that for any environment $\mathcal{Z}$ we have:

$$\text{LEXEC}_{\pi,\mathcal{D},\mathcal{Z}} \approx \text{LEXEC}_{\mathcal{F}_{\overline{\text{auth}}},\mathcal{S}_{\mathcal{D}},\mathcal{Z}} \tag{13}$$

The misinformants $\mathcal{M}_S$,$\mathcal{M}_R$ are constructed out of $\mathcal{S}_{\mathcal{D}}$. Recall that $\mathcal{J}$ expects to interact with informants running π where $\mathcal{S}_{\mathcal{D}}$ expects to interact with $\mathcal{F}_{\overline{\text{auth}}}$ and environment $\mathcal{Z}$. The main idea in the proof is that $\mathcal{S}_{\mathcal{D}}$ will generate all the outgoing communication to $\mathcal{J}$ as long as the misinformants will provide the services given to $\mathcal{S}_{\mathcal{D}}$ by $\mathcal{F}_{\overline{\text{auth}}}$. More precisely, to mimic the communication in π, each misinformant run internally a copy of $\mathcal{S}_{\mathcal{D}}$. In addition, when output is generated by the internal adversary, the misinformant passes this output to $\mathcal{J}$. Any input message given by $\mathcal{J}$ is forwarded the internal adversary. To mimic for $\mathcal{S}_{\mathcal{D}}$ the communication with $\mathcal{F}_{\overline{\text{auth}}}$, upon receiving a message m, the misinformant forwards it to $\mathcal{S}_{\mathcal{D}}$ as if it was given by $\mathcal{F}_{\overline{\text{auth}}}$. Once the internal adversary of $\mathcal{M}_S$ wish to approve the message m then $\mathcal{M}_S$ sends `(signal)` message $\mathcal{M}_R$. Similarly, once the internal adversary of $\mathcal{M}_R$ wish to approve the message m then $\mathcal{M}_R$ gives m to R.

We note that the view of the internal simulator $\mathcal{S}_{\mathcal{D}}$ in the misinformant experiment is identical to its view in the ideal execution with $\mathcal{F}_{\overline{\text{auth}}}$. This is true since each misinformant provides its internal adversary with an interface identical to $\mathcal{F}_{\overline{\text{auth}}}$. In addition, the informant-experiment is identical to the LUC execution of π and therefore the communication interface of $\mathcal{Z}$ is successfully provided by $\mathcal{J}$. The validity of the misinformants can be inferred by the standard technique of assuming an existence of distinguishing judge $\mathcal{J}$ and constructing a distinguishing environment $\mathcal{Z}$ as done in 3.5. In particular, if $\mathcal{Z}$ interacts with parties running π then the view of the simulated $\mathcal{J}$ within $\mathcal{Z}$ is distributed identically to the view of $\mathcal{J}$ when interacting with π and the informants. Similarly, if $\mathcal{Z}$ interacts with parties running $\text{LIDEAL}_{\mathcal{F}_{\overline{\text{auth}}}}$ then the view of the simulated $\mathcal{J}$ within $\mathcal{Z}$ is distributed identically to the view of $\mathcal{J}$ run when interacting with the misinformants.

For the derivation in the other direction we are given a Bi-deniable protocol π. We wish to construct an adversary $\mathcal{S}$ so that no $\mathcal{Z}$ will be able to tell whether it is interacting with π and the dummy adversary $\mathcal{D}$ or with $\mathcal{F}_{\overline{\text{auth}}}$ and $\mathcal{S}$. That is, for any $\mathcal{Z}$, $\mathcal{S}$ should satisfy

$$\text{LEXEC}_{\pi,\mathcal{D},\mathcal{Z}} \approx \text{LEXEC}_{\mathcal{F}_{\overline{\text{auth}}},\mathcal{S},\mathcal{Z}} \tag{14}$$

The fact that π is Bi-deniable guarantees that there exists misinformants $\mathcal{M}_S$ and $\mathcal{M}_R$, such that for any judge $\mathcal{J}$ we have:

$$\text{EXP}_{\pi,\mathcal{I}_S,\mathcal{I}_R,\mathcal{J}} \approx \text{EXP}_{\mathcal{M}_S,\mathcal{M}_R,\mathcal{J}} \tag{15}$$

The adversary $\mathcal{S}$ is constructed out of the misinformants. For simplicity, we denote by $\mathcal{S}_{(S,R)}$ the simulator of the sender and by $\mathcal{S}_{(R,S)}$ the simulator of the receiver. The environment $\mathcal{Z}$ expects to interact with parties running π. To mimic the communication in π, the simulators $\mathcal{S}_{(S,R)}$ and $\mathcal{S}_{(R,S)}$ run internally a copy of $\mathcal{M}_S$ and $\mathcal{M}_R$ respectively. In addition, when output is generated by the internal misinformant, the simulator passes this output to $\mathcal{Z}$. Any input message given by $\mathcal{Z}$ is forwarded to the internal misinformant. When a message m is given by $\mathcal{F}_{\overline{\text{auth}}}$, the simulator forwards it to the internal misinformant.

Once $\mathcal{M}_S$ wishes to send (signal) message to $\mathcal{M}_R$ then $\mathcal{S}_{(S,R)}$ send (Approve) to $\mathcal{F}_{\overline{\text{auth}}}$. Similarly, once the internal $\mathcal{M}_R$ wishes to give m to R , then $\mathcal{S}_{(R,S)}$ send (Approve) to $\mathcal{F}_{\overline{\text{auth}}}$.

We omit the validity proof due to resemblance with previous results.

We note that the proof above cannot be applied to show that UC security implies Bi-deniability. That is, we cannot construct uncoordinated misinformants out of a single UC simulator that depends on communication of both parties. To establish this observation, we show in section 6.2.5 that the UC ideal authentication functionality $\mathcal{F}_{\text{auth}}$ is not Bi-deniable.

From Theorem 6.2.4 and the LUC composability follows that Bi-deniability is a composable notion. That is:

Corollary 2. *Let ρ be some protocol and let π^ρ be a Bi-deniable protocol. Then, for any protocol ϕ that LUC-realizes ρ, π^ϕ is Bi-deniable.*

6.2.5 On UC security and Bi-deniability

Here our goal is to emphasize the inability of UC to capture Bi-deniability by showing that $\mathcal{F}_{\text{auth}}$ is not Bi-deniable.

Claim. The protocol $\text{IDEAL}_{\mathcal{F}_{\text{auth}}}$ *is not Bi-deniable.*

Proof. In order to argue about non Bi-deniability of the GUC ideal authentication protocol $\text{IDEAL}_{\mathcal{F}_{\text{auth}}}$ we consider the equivalent authentication functionality, the merger functionality of $\mathcal{F}_{\text{auth}}$. Recall that a merger functionality enables the adversaries to communicate freely via (Deliver,...) messages. In order to show that $\text{LIDEAL}_{\mathcal{F}_{\text{auth}}}$ is not Bi-deniable we construct a PPT judge $\mathcal{J}$ such that for all PPT misinformants $\mathcal{M}_R$ and $\mathcal{M}_S$ we have:

$$\text{EXP}_{\text{LIDEAL}_{\mathcal{F}_{\text{auth}}},\mathcal{I}_S,\mathcal{I}_R,\mathcal{J}} \not\approx \text{EXP}_{\mathcal{M}_S,\mathcal{M}_R,\mathcal{J}}$$

The judge $\mathcal{J}$ proceeds as follows: when activated for the first time, $\mathcal{J}$ randomly choose $r \leftarrow \{0,1\}^\lambda$, and instructs $\mathcal{I}_R$ to send $(\text{Deliver}, sid, r)$ to $\mathcal{I}_S$. Upon receiving output $(\text{Deliver}, sid, r)$ from $\mathcal{I}_S$, it outputs 1.

It can be verified that no misinformant $\mathcal{M}_S$ can make the judge output 1 with probability greater than negligible in λ. This is so, since $\mathcal{M}_S$ does not know r. However, when $\mathcal{J}$ interact with the informants and $\text{LIDEAL}_{\mathcal{F}_{\text{auth}}}$, it always outputs 1.

6.3 Confinement

Recall that a protocol is said to enforce confinement if it prevents leakage of secret information to unauthorized processes in the network. This guarantee should hold even if all parties are faulty. In this section we revisit the question of how to capture the classic confinement property in a simulation based, composable framework. We first recall the intuitive notion. Then we show the undesirably strong requirements posed by confinement definitions based on centralized adversary. Next, we present a definition of confinement and prove that any LUC secure realization enforces confinement as long as the realized task does. Lastly, we show that any UC functionality that enforces confinement is "super-ideal", in a well-defined sense, and thus hard to realize.

6.3.1 Confinement with a centralized adversary

In the work of [HKN05] a definition of confinement is presented. Their definition considers the UC execution model with the following modifications: the UC environment is split into two environments $\mathcal{E}_\mathcal{H}$ and $\mathcal{E}_\mathcal{L}$, where $\mathcal{E}_\mathcal{H}$ interacts with the high-level processes and $\mathcal{E}_\mathcal{L}$ with the low-level processes. All

processes have an I/O interface with the appropriate environment according to their classification. In addition, the high-level environment $\mathcal{E}_{\mathcal{H}}$ cannot give inputs either to the adversary or to the low-level environment $\mathcal{E}_{\mathcal{L}}$. [HKN05] define confinement as the following game: a random bit b is chosen by $\mathcal{E}_{\mathcal{H}}$, the parties run the protocol π, and eventually $\mathcal{E}_{\mathcal{L}}$ outputs its guess for b. We say that π enforce confinement for partition $\mathcal{H} : \mathcal{L}$ of the parties in π, if for any environments $\mathcal{E}_{\mathcal{H}}$, $\mathcal{E}_{\mathcal{L}}$ and adversary as above, $\mathcal{E}_{\mathcal{L}}$ succeeds in the confinement game with probability $\approx \frac{1}{2}$.

This definition enforces very strong requirements on the examined protocols, and as a consequence, many protocols that "obviously enforce confinement" do not satisfy this definition. We show an example for such a protocol, for three parties: Let P_1, P_2 be authorized processes, and P_3 be unauthorized process. P_1 holds a secret bit b and sends it to P_2. Whenever P_2 receives a message from P_1, it sends "hello" to P_3.

This protocol ensures that as long as P_2 is honest, the secret will never leak to unauthorized P_3, and therefore, it enforces confinement with respect to honest P_2. However, this protocol does not satisfy the confinement definition of [HKN05]. In particular, consider the case where P_1 is corrupted and the bit b chosen by $\mathcal{E}_{\mathcal{H}}$ to be P_1's input. In this scenario, b is known to the adversary that will output it to $\mathcal{E}_{\mathcal{L}}$. This results in success probability '1', and hence, fails to satisfy this definition. Moreover, no protocol, in which a party holding some secret information can be corrupted, would satisfy this definition. We remark that this weakness is not unique to the [HKN05] definition, and any definition based on centralized adversary is subject to this weakness.

6.3.2 Our definition

Our definition follows the idea of [HKN05]. Like there, we consider split environments. More precisely, the definition consists of the following entities: two environments $\mathcal{E}_{\mathcal{H}}$ and $\mathcal{E}_{\mathcal{L}}$, where $\mathcal{E}_{\mathcal{H}}$ knows some secret that $\mathcal{E}_{\mathcal{L}}$ is trying to learn, a set of parties $\mathcal{P} = \{P_1, .., P_n\}$ and a pair of adversaries $\mathcal{A}_{(i,j)}$ and $\mathcal{A}_{(j,i)}$ for each pair of potentially communicating parties P_i and P_j that are helping $\mathcal{E}_{\mathcal{H}}$ to leak the secret to $\mathcal{E}_{\mathcal{L}}$. We define the identity of adversary $\mathcal{A}_{(i,j)}$ to be $((i,j), \perp)$ and the code to be $\mathcal{A}$.

The experiment executed is the following: Let π be some protocol over a fixed set of parties $\mathcal{P}$ and let $\mathcal{H} : \mathcal{L}$ be some partition of the parties participating in π. The first ITI to be activated is $\mathcal{E}_{\mathcal{H}}$ and it is given a random bit b, which $\mathcal{E}_{\mathcal{L}}$ is trying to learn. The environments $\mathcal{E}_{\mathcal{H}}$ and $\mathcal{E}_{\mathcal{L}}$ control the I/O interface of the parties in $\mathcal{H}$ and $\mathcal{L}$ respectively. $\mathcal{E}_{\mathcal{H}}$ cannot communicate either with adversaries or with $\mathcal{E}_{\mathcal{L}}$; however, any $\mathcal{A}_{(i,j)}$ for $i \in \mathcal{H}$ can give outputs to it. On the other hand, $\mathcal{E}_{\mathcal{L}}$ is only prohibited from communicating with adversaries $\mathcal{A}_{(i,j)}$ for $i \in \mathcal{H}$ (i.e., it can send messages to $\mathcal{E}_{\mathcal{H}}$).

The system includes some communication means between the parties in π. The parties run π in the presence of the adversaries. The adversarial interface of this communication means defines the communication capabilities between the adversaries and the communication observed by them. Adversary $\mathcal{A}_{(i,j)}$ can communicate with party P_i and adaptively corrupt it. Upon corruption, the corrupter learns the entire state of the party. Whenever a party corrupted, the appropriate environment (according to the identity of the corrupted party) is informed about it. In addition, a corrupted party is fully controlled by its adversaries. A graphical depiction appears in Figure 15.

Let $\mathrm{CEXP}^{\mathcal{H}:\mathcal{L}}_{\pi,\mathcal{A},\mathcal{E}_{\mathcal{H}},\mathcal{E}_{\mathcal{L}}}$ denote the output distribution of $\mathcal{E}_{\mathcal{L}}$ in the above experiment.

Definition 7 (Confinement). *Let π be a PPT protocol and let $\mathcal{H} : \mathcal{L}$ be some partition of the parties in π. We say that π enforces $(\mathcal{H} : \mathcal{L})$-confinement if for any PPT adversary $\mathcal{A}$ and for any balanced PPT environments $\mathcal{E}_{\mathcal{H}}$ and $\mathcal{E}_{\mathcal{L}}$ we have: $\mathrm{CEXP}^{\mathcal{H}:\mathcal{L}}_{\pi,\mathcal{A},\mathcal{E}_{\mathcal{H}},\mathcal{E}_{\mathcal{L}}} \approx U_1$, where U_1 is the uniform distribution over $\{0,1\}$.*

6.3.3 LUC preserves confinement

Here we prove that LUC security preserves confinement. That is, we show that any LUC realization of a protocol that enforces confinement also enforce confinement.

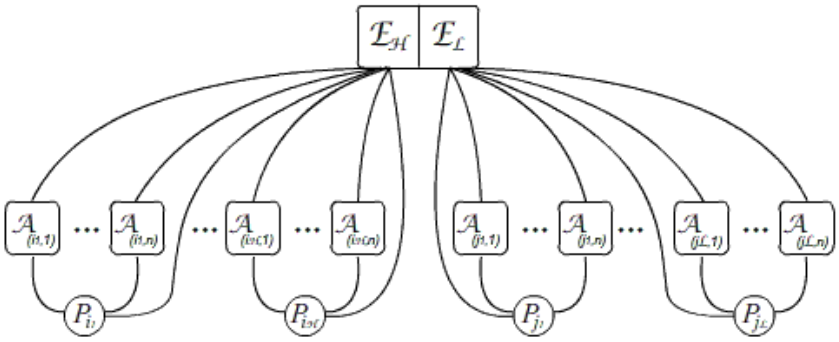

Fig. 15: The confinement experiment for partition $\mathcal{H} : \mathcal{L}$ of the participating parties in π. In the experiment the parties run π, while the adversaries, jointly with $\mathcal{E}_{\mathcal{H}}$ and $\mathcal{E}_{\mathcal{L}}$ control the communication. The environments $\mathcal{E}_{\mathcal{H}}$ and $\mathcal{E}_{\mathcal{L}}$ write inputs and read the subroutine outputs of parties under the above constrains. $\mathcal{E}_{\mathcal{L}}$ can also give inputs to $\mathcal{E}_{\mathcal{H}}$. In addition, $\mathcal{E}_{\mathcal{L}}$ can interact freely with all adversaries associated with $\mathcal{L}$, and all the adversaries associated with $\mathcal{H}$ can give outputs to $\mathcal{E}_{\mathcal{H}}$. Lastly, the adversaries can communicate with the appropriate party and corrupt it.

Theorem 8. *Let π, ϕ be protocols such that π LUC emulates ϕ. Then π enforce $(\mathcal{H} : \mathcal{L})$-confinement for all partitions $\mathcal{H} : \mathcal{L}$ of the parties in π for which ϕ enforce $(\mathcal{H} : \mathcal{L})$-confinement.*

Proof. Intuitively, the difference between confinement and LUC security is that the former considers split environments which, is a subset of all environments considered in in the security definition. Therefore, when we consider secure realization of a protocol that enforces confinement, we find that it also preserves the confinement property. More formally, let $\mathcal{H} : \mathcal{L}$ be a partition of parties in π such that ϕ enforce $(\mathcal{H} : \mathcal{L})$-confinement. We show that π enforce $(\mathcal{H} : \mathcal{L})$-confinement.

Assume for contradiction that there is an adversary $\mathcal{A}$ and environments $\mathcal{E}_{\mathcal{H}}$ and $\mathcal{E}_{\mathcal{L}}$ such that

$$\mathrm{CEXP}^{\mathcal{H}:\mathcal{L}}_{\pi,\mathcal{A},\mathcal{E}_{\mathcal{H}},\mathcal{E}_{\mathcal{L}}} \not\approx U_1 \tag{16}$$

Let $\mathcal{S}$ be the adversary guaranteed by the definition of realization with respect to adversary $\mathcal{A}$. We construct an environment $\mathcal{Z}_\pi$ such that

$$\mathrm{LEXEC}_{\phi,\mathcal{S},\mathcal{Z}_\pi} \not\approx \mathrm{LEXEC}_{\pi,\mathcal{A},\mathcal{Z}_\pi} \tag{17}$$

Environment $\mathcal{Z}_\pi$ runs an interaction between simulated instances of $\mathcal{E}_{\mathcal{H}}$ and $\mathcal{E}_{\mathcal{L}}$ and $\mathcal{A}$. When activated for the first time, it interprets the input z as a pair $z = (z,\, b)$ where z is an input for $\mathcal{Z}_\pi$, and $b \in \{0, 1\}$. Next, give $\mathcal{E}_{\mathcal{H}}$ the bit b as external input. In addition:

1. All the inputs generated by $\mathcal{E}_{\mathcal{L}}$ to the adversaries are forwarded to the external adversaries. Similarly, whenever $\mathcal{Z}_\pi$ receives an output value v from adversary with identity $id = ((i, j), \perp)$, $\mathcal{Z}_\pi$ passes v to the simulated environment according to $\mathcal{H} : \mathcal{L}$.
2. All inputs from $\mathcal{E}_{\mathcal{H}}$ and $\mathcal{E}_{\mathcal{L}}$ to the parties of π are forwarded to the external parties, and all the outputs coming from the external parties are forwarded to $\mathcal{E}_{\mathcal{H}}$ and $\mathcal{E}_{\mathcal{L}}$ as coming from the parties of π according to $\mathcal{H} : \mathcal{L}$.
3. In addition, whenever $\mathcal{Z}_\pi$ passes input of length m to some party, it first passes input $1^{p(m)}$ to all external adversaries, where $p()$ is the maximum between the polynomials bounding the run times of ϕ and π.
4. Finally, $\mathcal{Z}_\pi$ outputs whatever the simulated $\mathcal{E}_{\mathcal{L}}$ outputs.

It can be readily verified, by inspecting the code of $\mathcal{Z}_\pi$, that the ensembles $\mathrm{CEXP}^{\mathcal{H}:\mathcal{L}}_{\pi,\mathcal{A},\mathcal{E}_\mathcal{H},\mathcal{E}_\mathcal{L}}$ and $\mathrm{LEXEC}_{\pi,\mathcal{A},\mathcal{Z}_\pi}$ are identical. Similarly, ensembles $\mathrm{CEXP}^{\mathcal{H}:\mathcal{L}}_{\phi,\mathcal{S},\mathcal{E}_\mathcal{H},\mathcal{E}_\mathcal{L}}$ and $\mathrm{LEXEC}_{\phi,\mathcal{S},\mathcal{Z}_\pi}$ are identical as well. By the definition of confinement, it guarantees that

$$\mathrm{CEXP}^{\mathcal{H}:\mathcal{L}}_{\phi,\mathcal{S},\mathcal{E}_\mathcal{H},\mathcal{E}_\mathcal{L}} \approx U_1 \tag{18}$$

By equations 16 and 18 follows that

$$\mathrm{CEXP}^{\mathcal{H}:\mathcal{L}}_{\phi,\mathcal{S},\mathcal{E}_\mathcal{H},\mathcal{E}_\mathcal{L}} \not\approx \mathrm{CEXP}^{\mathcal{H}:\mathcal{L}}_{\pi,\mathcal{A},\mathcal{E}_\mathcal{H},\mathcal{E}_\mathcal{L}} \tag{19}$$

and therefore it follows that $\mathrm{LEXEC}_{\phi,\mathcal{S},\mathcal{Z}_\pi} \not\approx \mathrm{LEXEC}_{\pi,\mathcal{A},\mathcal{Z}_\pi}$ as desired.

We note that this proof technique cannot be applied to show that any UC realization of a UC functionality that enforces confinement will also enforce confinement. That is so, since upon receiving an output from the external adversary, the constructed environment does not have the knowledge to correctly determine which internal environment this output should be given.

From Theorem 8 and the LUC composability follows that confinement is preserved under composition. That is:

Corollary 3. *Let ρ be some protocol and let π^ρ be a protocol that enforces $(\mathcal{H}:\mathcal{L})$-confinement. Then, for any protocol ϕ that LUC-realizes ρ, π^ϕ enforces $(\mathcal{H}:\mathcal{L})$-confinement.*

6.3.4 Confinement with respect to super-ideal functionalities

Here, we show that any UC functionality that enforces confinement is "super-ideal". That is, such functionalities do not provide the adversary with any information, even when a party is corrupted. We call such functionalities super-ideal since such functionalities essentially mandate communication channels which offer absolute physical security that hides even whether communication took place at all. We also show that this is not the case for LUC functionalities that enforce confinement.

Definition 8 (super-ideal functionality). *Let $\mathcal{F}$ be a UC functionality, $m\mathcal{F}$ be the equivalent LUC functionality, and $\mathcal{P}$ be a set of party identities. We say that $\mathcal{F}$ is super-ideal with respect to party P_i if for any PID j, any PPT adversary $\mathcal{A}$, and for any two input vectors $\overrightarrow{x_0}$ and $\overrightarrow{x_1}$ that differ only in ith entry the distributions $\mathcal{SI}_j$ and U_1 are indistinguishable, where U_1 is the uniform distribution over $\{0,1\}$ and $\mathcal{SI}_j$ is the output distribution of the following game:*

1. *The challenger selects a bit $b \in \{0,1\}$ uniformly at random, and sets parties inputs according to $\overrightarrow{x_b}$.*
2. *The parties and the adversaries (with code $\mathcal{A}$) execute $\mathrm{LIDEAL}_{m\mathcal{F}}$.*
3. *Finally, the adversary with identity $((i,j)\perp)$ outputs a guess for the value of b.*

Claim. Let $\mathcal{F}$ be a UC functionality and let $\mathcal{H}:\mathcal{L}$ be some partition for which $\mathcal{F}$ enforces $(\mathcal{H}:\mathcal{L})$-confinement. Then, $\mathcal{F}$ is super-ideal with respect to all parties in $\mathcal{H}$.

We note that Claim 6.3.4 does not hold for general LUC functionalities. In contrast to UC, where the centralized adversary can pass to low-level process any information available to it, a general LUC functionality can prevent the high-level adversaries to pass information to low-level adversaries. Therefore, LUC functionality can enforce confinement, even if the high-level adversaries exposed to the secret information.

Proof. Let $\mathcal{P}$ be a set of party identities and let $\mathcal{H}:\mathcal{L}$ be some partition of $\mathcal{P}$ for which $\mathcal{F}$ enforce $(\mathcal{H}:\mathcal{L})$-confinement. Assume for the sake of contradiction that there exists a party $P_i \in \mathcal{H}$ for which $\mathcal{F}$ is not super-ideal. Therefore, there exists a PID j, an adversary $\mathcal{A}$ and two input vectors $\overrightarrow{x_0}$ and $\overrightarrow{x_1}$ such that $\mathcal{SI}_j \not\approx U_1$. We construct an adversary $\mathcal{A}_\mathcal{F}$ and environments $\mathcal{E}_\mathcal{H}$,$\mathcal{E}_\mathcal{L}$ such that

$$\mathrm{CEXP}^{\mathcal{H}:\mathcal{L}}_{\mathrm{IDEAL}_\mathcal{F},\mathcal{A}_\mathcal{F},\mathcal{E}_\mathcal{H},\mathcal{E}_\mathcal{L}} \not\approx U_1 \tag{20}$$

– The environment $\mathcal{E}_{\mathcal{H}}$ behaves as follows: when activated for the first time with input bit b, if $b = 0$ it gives inputs according to $\overrightarrow{x_0}$; otherwise according to $\overrightarrow{x_1}$.
– The environment $\mathcal{E}_{\mathcal{L}}$ behaves as follows:
  • when activated for the first time it gives inputs according to $\overrightarrow{x_0}$ (recall that the vectors are differ only in entry $i \in \mathcal{H}$).
  • when it receives from one of the adversaries a bit b' , it outputs b' and halts.
– The adversary $\mathcal{A}_{\mathcal{F}}$ runs $\mathcal{A}$ internally and do whatever $\mathcal{A}$ does. Whenever $\mathcal{A}$ halts and outputs its decision, $\mathcal{A}_{\mathcal{F}}$ with identity $((i,j)\perp)$ does the following: if the output of $\mathcal{A}$ indicates that $\overrightarrow{x_0}$ was used, then it sets $b' = 0$; otherwise it sets $b' = 1$. Next, it sends b' to $\mathcal{E}_{\mathcal{L}}$ via the deliver service of $m\mathcal{F}$.

It can be verified that the success probability of $\mathcal{E}_{\mathcal{L}}$ is $\frac{1}{2} + \epsilon$ for non negligible ϵ, where ϵ is the advantage of $\mathcal{A}$ in the above game.

We remark that Claim 6.3.4 does not hold with respect to LUC functionalities. We establish this observation by presenting a non super-ideal functionality that enforces confinement. Let $\mathcal{F}$ be the following three party functionality:

– on input message m from party P_1 the functionality computes $c = \mathrm{ENC}_{pk}(m, r)$ and outputs c to P_2.
– on input message c from party P_2 the functionality outputs c to P_3.
– upon corruption of party P_i, the functionality reveals to all of P_i's adversaries its inputs.

Let P_1 and P_3 be high-level processes that can communicate only via the channel $\mathcal{F}$ (i.e through the low-level process P_2). The functionality $\mathcal{F}$ enforces $(\mathcal{H} : \mathcal{L})$-confinement for $\mathcal{H} = \{P_1, P_3\}$ and $\mathcal{L} = \{P_2\}$ since it does not allow high-level processes to communicate with low-level process, and it hides the high-level processes input. Moreover, the functionality $\mathcal{F}$ is not super-ideal with respect to $\mathcal{H}$ since it disclose P_1's input message.

6.4 Incentive-structure Preservation

Recall that game-theoretic models propose that all players are motivated by their own individual incentives and usually trying to maximize their payoff. We view protocols as games to be played by rational players, in a sense that payoffs are now functions of the players' transcripts. Later in this section, we discuss how to formally represent protocols as games.

Informally, local adversaries enable cryptographically model rational players that are driven by their own incentives. In particular, these incentives determine for each player with whom he wishes to cooperate and in what cost, and this can be modeled using local adversaries. In contrast, the centralized adversary models, where the centralized adversary controllers the parties all together, cannot be used to capture players incentives. Indeed, we show that LUC preserves the incentive structure when protocols are viewed as games played by rational players.

6.4.1 On LUC and preservation of Incentive-structure

First, we need to define how to represent protocols as games. Similar to [GLR10], we consider n-party protocols as extensive form games of incomplete information. The possible messages of players in a protocol correspond to the available actions in the game tree, and the prescribed instructions correspond to a strategy in the game.

The protocol is parametrized by a security parameter $k \in \mathbb{N}$. The set of possible messages in the protocol, as well as its prescribed instructions, typically depend on this k. Assigning for each k and each party a payoff for every outcome, a protocol naturally induces a sequence $\Gamma^{(k)} = (H^{(k)}, P^{(k)}, A^{(k)}, u^{(k)})$ of extensive games, where:

- $H^{(k)}$ is the set of possible transcripts of the protocol (sequences of messages exchanged between the parties). A history $h \in H^{(k)}$ is terminal if the prescribed instructions of the protocol instruct the player whose turn it is to play next to halt on input h. The set of terminal histories is denoted $Z^{(k)}$.
- $P^{(k)}$:$(H^{(k)} \backslash Z^{(k)}) \rightarrow \{1, 2, ..., n\}$ is a function that assigns a "next" player to every non-terminal history.
- $A^{(k)}$is a function that assigns to every non-terminal history $h \in H^{(k)} \backslash Z^{(k)}$ a set $A^{(k)}(h) = \{m : (h \circ m) \in H^{(k)}\}$ of possible protocol messages to player $P^{(k)}(h)$
- $u^{(k)} = (u_1^{(k)}, u_2^{(k)}, ..., u_n^{(k)})$ is a vector of payoff functions $u_i^{(k)} : Z^{(k)} \rightarrow \mathbb{R}$.

A sequence $\Gamma^{(k)} = (H^{(k)}, P^{(k)}, A^{(k)}, u^{(k)})_{k \in \mathbb{N}}$ of games defined as above is referred to as a computational game. We note that for each protocol and any security parameter $H^{(k)}, P^{(k)}$ and $A^{(k)}$ are uniquely defined.

We extend the above notation to ideal protocols. Let $\mathcal{F}$ be an ideal functionality, then the ideal protocol $\mathrm{LIDEAL}_{\mathcal{F}}$ corresponds to a game with a communication device $\mathcal{F}$ as follows:

- players receive their inputs.
- players communicate with $\mathcal{F}$ by giving it some input, which are not necessary the inputs received.
- each player receive its personal output from $\mathcal{F}$.

In this section we consider a non-concurrent version of LUC, denoted by (NC)LUC security. That is, we consider a NC environment as defined in [Can01] and we require the protocols to be LUC-secure in the $\mathcal{F}_{\mathrm{mt}}$-hybrid model with respect to NC environments. The functionality $\mathcal{F}_{\mathrm{mt}}$ presented in figure 16.

Functionality $\mathcal{F}_{\mathrm{mt}}$

Functionality $\mathcal{F}_{\mathrm{mt}}$ proceeds as follows:

At the first activation, verify that $sid = (\mathcal{P}; sid')$, where $\mathcal{P}$ is an ordered set of m identities; else halt. Denote the parties identities $P_1, ..., P_m$, adversaries identities $\left\{\mathcal{S}_{(i,j)}\right\}_{i \neq j, i \in \mathcal{P} \bigcup \{\mathcal{M}\}, j \in \mathcal{P} \bigcup \{\mathcal{M}\}}$.

- Upon receiving an input $(\mathrm{Send}, sid, m, P_j)$ from party P_i send a public delayed output $(\mathrm{Send}, sid, m, P_i)$ to Pj.

Fig. 16: The message transmission functionality $\mathcal{F}_{\mathrm{mt}}$.

For simplicity, we consider LUC ideal functionalities (that represent communication device) for Secure Function Evaluation.

In order to reason about strategies of PPT players in the extensive form game we consider *behavioral strategies*. Informally, behavior strategy specifies the probability with which each action would be taken conditional upon that history happened.

Definition 9 (Behavioral strategy). *Behavioral strategies of players in an extensive form game are collections $\sigma_i = (\sigma_i(h))_{h:P(h)=i}$ of independent probability measures, where $\sigma_i(h)$ is a probability measure over $A(h)$. Let $\sigma = (\sigma_1, ..., \sigma_n)$ denote a strategy profile.*

Now that we defined how to view protocols as games, we translate the emulation of protocols notion to the game theoretic setting. Let $\Gamma_{\mathcal{F}} = (H, P, A, u)$ be a game with communication device $\mathcal{F}$, and let π be a protocol that (NC)LUC-realizes $\mathcal{F}$. We define the real-world extensive form game $\Gamma_{\pi} = (H', P', A', u')$, where players execute π instead of $\mathcal{F}$, as follows: since π uniquely defines H', P', A' we only need to define a payoff function u'. For each terminal history $h' \in H$' consider the corresponding history $h \in H$. That is, h is the history that induces the simulated view h' in $\mathrm{LIDEAL}_{\mathcal{F}}$. Moreover,

it can be efficiently computed by the simulator guaranteed from the security of π. For each terminal history $h' \in H$', we define $u'_i(h') = u_i(h)$.

We note that the payoff function of a real-world game Γ_π is uniquely defined by the simulators ,which guaranteed from the LUC security of π, and the payoff function of $\Gamma_\mathcal{F}$.

Theorem 9. *Let $\Gamma_\mathcal{F} = (H, P, A, u)$ be an extensive form game, where $\mathcal{F}$ is a communication device in $\Gamma_\mathcal{F}$. Then for any π that is a (NC)LUC-secure realization of $\mathcal{F}$ holds:*
(1) for any strategy profile $\sigma = (\sigma_1, ..., \sigma_n)$ in the real-world extensive form game $\Gamma_\pi = (H', P', A', u')$ there exists a strategy profile $\bar{\sigma} = (\bar{\sigma}_1, ..., \bar{\sigma}_n)$ in $\Gamma_\mathcal{F}$ that achieves the same expected payoff.
(2) for any strategy profile $\bar{\sigma} = (\bar{\sigma}_1, ..., \bar{\sigma}_n)$ in $\Gamma_\mathcal{F}$ there exists a strategy profile $\sigma = (\sigma_1, ..., \sigma_n)$ in the real-world extensive form game $\Gamma_\pi = (H', P', A', u')$ that achieves the same expected payoff.

Proof. Let $\Gamma_\mathcal{F} = (H, P, A, u)$, protocol π ,and $\Gamma_\pi = (H', P', A', u')$ be as above. The proof of item (1) is done in a way of contradiction. That is, there exists a strategy profile $\sigma = (\sigma_1, ..., \sigma_i, ..., \sigma_n)$ in Γ_π such that for all strategies profiles $\bar{\sigma} = (\bar{\sigma}_1, ..., \bar{\sigma}_n)$ in $\Gamma_\mathcal{F}$ the expected payoff induced by σ is distinguishable with probability ϵ from the expected payoff induced by $\bar{\sigma}$. Let A be the distinguisher and let $\mathcal{S}_\mathcal{D}$ be the simulator guaranteed for the dummy adversary $\mathcal{D}$. Now we construct an environment $\mathcal{Z}_\pi$that manages to distinguish between $\mathrm{LEXEC}_{\pi,\mathcal{D},\mathcal{Z}_\pi}$ and $\mathrm{LEXEC}_{\mathrm{LIDEAL}_\mathcal{F},\mathcal{S}_\mathcal{D},\mathcal{Z}_\pi}$.

The environment $\mathcal{Z}_\pi$ constructed as follows: it instructs the adversaries to corrupt all the parties and to play according to σ. Let $\hat{h}$ denote the terminal history as observed by $\mathcal{Z}_\pi$. Next, $\mathcal{Z}_\pi$ computes the payoff of each player $p_i = u'_i(\hat{h})$, forward the payoff vector to A and outputs whatever A outputs.

It can be verified that the advantage of $\mathcal{Z}_\pi$ in distinguishing $\mathrm{LEXEC}_{\pi,\mathcal{D},\mathcal{Z}_\pi}$ and $\mathrm{LEXEC}_{\mathrm{LIDEAL}_\mathcal{F},\mathcal{S}_\mathcal{D},\mathcal{Z}_\pi}$ is the same as the advantage of A in distinguishing payoffs drawn from the payoff distribution induced by Γ_π from $\Gamma_\mathcal{F}$. Therefore, $\mathrm{LEXEC}_{\pi,\mathcal{D},\mathcal{Z}_\pi} \not\approx \mathrm{LEXEC}_{\mathrm{LIDEAL}_\mathcal{F},\mathcal{S}_\mathcal{D},\mathcal{Z}_\pi}$.

For the proof of item (2) it is enough to show that u' defines a surjective strategy mapping from Γ_π to $\Gamma_\mathcal{F}$. We note in the non-concurrent model each strategy depend only on the initial input given by the environment and the output received from the SFE functionality. For each terminal history h of party P_i in $\Gamma_\mathcal{F}$ consider the following terminal h' history for P_i in Γ_π: run the protocol π on input x' that was given to $\mathcal{F}$ in h. Whenever obtaining an output y of π set the output as described in h as if P_i received y from $\mathcal{F}$. By definition of u' we have that $u'_i(h') = u_i(h)$, as desired.

We remark that Theorem 9 implies that any Nash equilibrium (NE) in $\Gamma_\mathcal{F}$ is mapped to a computational NE in Γ_π and vice versa. In addition, since the non-concurrent model achieves sequential composition, Theorem 9 implies game-theoretic non-concurrent composition.

7 A solution via physical computation

Informally, our goal here is to construct a LUC secure realization of an arbitrary (poly-time) functionality $\mathcal{F}$. The idea behind this solution is to implement the GMW protocol using physical implementations of cryptographic primitives. This approach enables us to eliminate the use of randomness, which can be used by parties in order to collude. We first introduce the physical components of our protocol, and then describe the protocol in detail. This solution is strongly inspired by the [ILM05] solution.

7.1 Building blocks

In this section we informally present the communication network in which the physical GMW games will be played.

Machinery. We have the following physical components:

- boxes
- machine to create boxes with content
- multiplication and addition machines
- duplication machine

– opening machine
– machine to shuffle boxes in a super-box.

We will represent $'0$' by an empty box and '1' by a box containing a piece of paper. The multiplication, duplication, and addition machines will receive the boxes and create a new box with the appropriate content. (If the new box should represent '1', the machine will use a fresh piece of paper.)

Conceptually, we envisage a group of players, seating far apart around a large table in a room equipped with cameras, communicating via identical boxes (and super-boxes) and addition, multiplication, duplication and shuffle machines. Player can also communicate with entities outside the room using their cell-phones. Informally, a player can privately or publicly choose a message or privately toss a coin (all participants can see that he tossed a coin, but nobody can see the result of the toss), and put a piece of paper according to the outcome into a new, empty box. So long as it is not opened, the box totally hides and guarantees the integrity of its content. Only the owner of a box can open it using an opening machine (in which case all the players are aware that he is opening it and he will be the only one to read its content). An opening of a box can be done only if the opening machine is on. A player owns a box if it is physically close to him. By definition, the player originally locking a new box owns it. After that, ownership of a box can be transferred to another player by passing it to him. Furthermore, each box has a unique serial number, which only its owner can see. Other participants sitting around the table cannot see the serial number on the box.

A player can perform multiplication and addition of the values inside boxes by inserting two boxes into an appropriate machine, obtaining a single box containing the product or sum (respectively) of the values in the inserted boxes. The addition machine may accept either two boxes or m boxes as inputs (where m is the number of players). In addition, a player can duplicate a value inside a box by inserting the box into the duplication machine and obtaining an additional new box containing the same value.

Furthermore, a player can also publicly put four of his boxes into a new super-box SB, in which case none of them can be opened before SB is opened. All super-boxes are again identical to each other, but are larger than (ordinary) boxes. The rules of ownership for super-box are the same as for boxes. There is only one possible way for a player i to open a super-box SB of his: all players observe that i has opened SB, and the player can choose only one sub-box inside which can be manipulated (e.g., opened or transferred). The remaining sub-boxes stay inside the locked superbox, which is placed in the middle of the table where none of the players can touch it.

Boxes and super-boxes always stay above the table and their transfers are always tracked by the players. The players can thus "mentally assign" to each box or super-box an identifier, j, insensitive to any possible change of ownership. The only exception is when a player i publicly puts his super-box into a shuffle machine: when it is taken out, the sub-boxes contents will remain unchanged and private, but their order is randomly permuted, in a way that is unpredictable to all players.

The cameras can identify actions and are in charge of the electricity in the room and can turn on and off each machine and lights. See motivation in paragraph 7.2.1

Lastly, at each moment a player may leave the room, thereby causing the game to abort.

7.1.1 Committed oblivious transfer

Now we describe how to implement one-out-of-four oblivious transfer in the physical model described above. It is similar to the committed OT of [CGT95].

Suppose Alice would like Bob to obtain a commitment to one out of four values, in such way that Bob will not learn anything about the remaining three values and Alice won't know which value was chosen by Bob.

In order to achieve these properties, Alice will generate four boxes containing the values for the OT using the multiplication/addition machines and put those boxes into a new super-box. This super-box is shuffled using the shuffle machine. Afterwards Alice removes the super-box from the shuffle machine and passes it to Bob, who in turn opens the super-box and chooses exactly one sub-box. Bob chooses a box according to the serial number of the sub-boxes and the value in Bob's input box.

This procedure is observable by all participants, who can verify that Alice and Bob abide by the rules of the model: for example, they verify that Bob take exactly one box and places the super-box with the remaining sub-boxes locked in the middle of the table. However, the participants cannot see neither the serial number of the chosen sub-box nor the serial number of the remaining sub-boxes.

By the construction, it is clear that Bob learns only one value: if Bob deviates and greedily opens more than one box, the other participants notice that and abort the protocol. Since the super-box was shuffled and passed to Bob, Alice and all other participants cannot see the serial number of Bob's chosen box, and therefore cannot know which of the four sub-boxes Bob chose.

The committed OT implementation described above will be invoked many times during the function evaluation.

It is important to note that the use of the shuffle machine in the implementation is vital to guaranteeing OT properties: without the shuffling, Alice will know which box Bob chose by the relative order of the chosen box in the super-box. For instance, if Bob chooses the third box in the super-box, Alice will know that it's the third box she inserted. The shuffling permutes the sub-boxes into an unpredictable order, therefore seeing which box was chosen (by its relative position in the super-box) won't reveal any information to Alice.

7.1.2 Ideal functionalities for the physical devices

We present ideal functionalities that capture the physical properties of the devices in this model, as described above. Note that all actions in the physical world are public, and only the values (inputs and outputs) are private.

In order to capture the notions of *validity* (opened/unopened boxes) and *ownership* (who possesses a given box) in the physical world, we define the following ideal functionality, $\mathcal{F}_{\mathrm{db}}$:

Functionality $\mathcal{F}_{\mathrm{db}}$, described in Figure 17, is a registrar of commitments ownership. It models boxes with serial numbers containing a value inside and possessed by some parties. Each record in the database represents an unopened box in the physical world. The existence of access control enables us to model oblivious transfer.

We would like to capture the situation of players sitting in a room equipped with cameras, where each player can perform actions at any time (we assume time is discrete). Each action is observed and examined not only by the players but also by the cameras which controls the electricity in the room. In order to achieve the desired, we define the following ideal functionality $\mathcal{F}_{\mathrm{room}}$:

Functionality $\mathcal{F}_{\mathrm{room}}$, described in Figure 18, proceeds as follows:

- The ability of players to perform actions at any time instant is modeled by having $\mathcal{F}_{\mathrm{room}}$ collect all the messages that the parties wish to send (recall that according to the model each party can send only one message in each activation which models the fact that each player can perform for at most one action in a time instant).
- The ability of players to perform actions concurrently to other players' actions is modeled by having $\mathcal{F}_{\mathrm{room}}$ send simultaneously all the valid messages collected.
- The ability of leaving the room and turn off the electricity (as long as the opening machine is off) is modeled by having $\mathcal{F}_{\mathrm{room}}$ halt whenever receiving and abort message.
- The cameras which are able to detect inappropriate behavior and turn off the electricity (as long as the opening machine is off) are modeled by having $\mathcal{F}_{\mathrm{room}}$ halt whenever there is illegal action that performed (delivered to designated functionality).
- The turning on of the opening machine after all players performed some specific action (and the ability to open boxes only after turning it on) is modeled by having $\mathcal{F}_{\mathrm{room}}$ discarding all the opening messages until all the players perform the (Added_m_values) action and setting the machine_on variable to '1' afterwards.
- The machine_on variable indicates whether the opening machine is on or off.

Functionality $\mathcal{F}_{\mathrm{db}}$

Functionality $\mathcal{F}_{\mathrm{db}}$ running with functionalities $\mathcal{F}_{\mathrm{ccon}}$, $\mathcal{F}_{\mathrm{add}}$, $\mathcal{F}_{\mathrm{mult}}$, $\mathcal{F}_{\mathrm{COT}}$, $\mathcal{F}_{\mathrm{duplicate}}$, $\mathcal{F}_{\mathrm{local\_open}}$, $\mathcal{F}_{\mathrm{valid}}$ and a memory M, proceeds as follows: At the first activation, verify that $sid = (\mathcal{T}; sid')$, where $\mathcal{T}$ is a set of functionality identities; else halt. Also, initialize variable $last\_cid = 0$

1. upon receiving a tuple $(\mathrm{owner}, cid, P_i)$ from another ideal functionality, proceed as follows:
   (a) if the tuple (cid, P_i, b) is recorded then send the message (valid,, cid, P_i, b) to that ideal functionality.
   (b) otherwise, send (invalid_record).
2. upon receiving a tuple $(\mathrm{record}, cid, P_i, b)$ from another ideal functionality, proceed as follows:
   (a) if the tuple (cid, P_j, b') is recorded for some party P_j then send the message (invalid_cid) to that ideal functionality.
   (b) otherwise, add (cid, P_i, b) to M and send the message (recorded) to that ideal functionality.
3. upon receiving a tuple $(\mathrm{delete}, cid, P_i)$ from another ideal functionality, proceed as follows:
   (a) if the tuple (cid, P_i, b) is recorded then delete (cid, P_i, b) from M and send the message (deleted) to that ideal functionality.
   (b) otherwise, send the the message (invalid_record) to that ideal functionality
4. upon receiving a tuple $(\mathrm{delete\_read\_only}, cid, P_i)$ from another ideal functionality, proceed as follows:
   (a) if the tuple $(\mathrm{read\_only}, cid, P_i, b)$ is recorded then delete $(\mathrm{read\_only}, cid, P_i, b)$ from M and send the message (deleted) to that ideal functionality.
   (b) otherwise, send the the message (invalid_record) to that ideal functionality
5. upon receiving a tuple $(\mathrm{change\_owner}, cid, P_i, P_j)$ from another ideal functionality, proceed as follows:
   (a) if the tuple (cid, P_i, b) is recorded then change the record in M to (cid, P_j, b) and send the message (changed) to that ideal functionality.
   (b) otherwise, send the the message (invalid_record) to that ideal functionality.
6. upon receiving a tuple $(\mathrm{restrict\_access}, cid, P_i)$ from another ideal functionality, proceed as follows:
   (a) if the tuple (cid, P_i, b) is recorded then change the record in M to $(\mathrm{read\_only}, cid, P_i, b)$ and send the message (changed) to that ideal functionality.
   (b) otherwise, send the the message (invalid_record) to that ideal functionality
7. upon receiving a tuple $\left(\mathrm{duplicate},\ cid,\ \hat{cid},\ P_i\right)$ from another ideal functionality, proceed as follows:
   (a) if the tuple (cid, P_i, b) is recorded then add $\left(\hat{cid}, P_i, b\right)$ to M and send the message (duplicated) to that ideal functionality
   (b) otherwise, send the the message (invalid_record) to that ideal functionality
8. upon receiving a message (Generate_ID) from another ideal functionality, proceed as follows:
   (a) compute $last\_cid = last\_cid + 1$.
   (b) send ID ($last\_cid$) to that ideal functionality.

Fig. 17: The database functionality

Functionality $\mathcal{F}_{\text{room}}$

Functionality $\mathcal{F}_{\text{room}}$ running with functionalities $\mathcal{F}_{\text{ccon}}$, $\mathcal{F}_{\text{add}}$, $\mathcal{F}_{\text{mult}}$, $\mathcal{F}_{\text{COT}}$, $\mathcal{F}_{\text{duplicate}}$, $\mathcal{F}_{\text{local\_open}}$, $\mathcal{F}_{\text{valid}}$ proceeds as follows. At the first activation, verify that $sid = (\mathcal{P}, \mathcal{T}; sid')$, where $\mathcal{P}$ is an ordered set of m identities , and $\mathcal{T}$ is a set of functionality identities; else halt. Denote the parties identities $P_1, ..., P_m$ and the adversaries identities $\left\{\mathcal{A}_{(i,j)}\right\}_{i \neq j, i \in \mathcal{P}, j \in \mathcal{P}}$. Also, initialize variable $machine\_on = 0$ and record $P = \emptyset$. Next:

1. For $i = 1, ..., m$ do:
   (a) Activate the party P_i and collect the message s (If any) it wish to send.
      i. verify that the message has the correct structure and the recipients are the functionalities (in case $s \neq \perp$), otherwise discard the message and goto step (a).
      ii. if the message is (Local_open) message and $machine\_on = 0$ then discard this message and goto step (a) with i++.
      iii. if $machine\_on = 1$ and the message is not (Local_open) then discard the message and goto step (a) with i++.
      iv. in case $s \neq \perp$ check the validity of the message by sending s to the functionality $\mathcal{F}_{\text{valid}}$; if is invalid then discard the message and goto step (a) with i++.
      v. If the message is (Added_m_values) then update P to contain $\{P_i\}$.
2. Deliver all the non discarded messages (which are not abort messages) collected in the previous step to their destination.
3. Inform all parties and adversaries on any abort received (include aborting parties identities). also, if $machine\_on = 0$ and some party aborted then halt.
4. If at least one of the delivered messages (which describes an action that has been executed) is not according to the publicly observable legal behavior and $machine\_on = 0$ then halt.
5. If P contains all parties identity then set $machine\_on = 1$.
6. Upon receiving input (Corrupt, P_i, p) from P_i related adversary $\mathcal{A}_{(i,j)}$ do:
   (a) Record P_i as corrupted party and send (Corrupted, P_i) to P_i.

Fig. 18: The room functionality. (the code above is more high level, and the precise code can be derived.)

– The turning off all other machines after all players performed some specific action is modeled by having $\mathcal{F}_{\text{room}}$ discarding any message besides opening after setting the machine_on to '1'.

In order to capture the set of available actions to each player at given time we define the following ideal functionality, $\mathcal{F}_{\text{valid}}$:

Functionality $\mathcal{F}_{\text{valid}}$

Functionality $\mathcal{F}_{\text{valid}}$ running with functionalities $\mathcal{F}_{\text{room}}$, $\mathcal{F}_{\text{db}}$ and proceeds as follows: At the first activation, verify that $sid = (\mathcal{T}; sid')$, where $\mathcal{T}$ is a set of functionality identities; else halt.

1. upon receiving a message m from $\mathcal{F}_{\text{room}}$ proceeds as follows:
   (a) if the message is (Pass, cid, P_i, P_j) do:
      i. if the tuple (cid, P_i, b) is recorded then send (valid) to $\mathcal{F}_{\text{room}}$.
      ii. otherwise, send (invalid) to $\mathcal{F}_{\text{room}}$.
   (b) if the message is (Add, P_i, cid, cid') do:
      i. if the tuples (cid, P_i, b) and (cid', P_i, b') are recorded then send (valid) to $\mathcal{F}_{\text{room}}$.
      ii. otherwise, send (invalid) to $\mathcal{F}_{\text{room}}$.
   (c) if the message is (Add_m_values, $cid_1, \ldots, cid_m$, P_i) do:
      i. if the tuples $(cid_1, P_i, b_1), \ldots, (cid_m, P_i, b_m)$ are recorded then send (valid) to $\mathcal{F}_{\text{room}}$.
      ii. otherwise, send (invalid) to $\mathcal{F}_{\text{room}}$.
   (d) if the message is (Multiply,P_i, cid, cid') do:
      i. if the tuples (cid, P_i, b) and (cid', P_i, b') are recorded then send (valid) to $\mathcal{F}_{\text{room}}$.
      ii. otherwise, send (invalid) to $\mathcal{F}_{\text{room}}$.
   (e) if the message is (Duplicate, P_i, cid) do:
      i. if the tuple (cid, P_i, b) is recorded then send (valid) to $\mathcal{F}_{\text{room}}$.
      ii. otherwise, send (invalid) to $\mathcal{F}_{\text{room}}$.
   (f) if the message is (Shuffle, cid_{00},cid_{01}, cid_{10}, cid_{11}, P_i, P_j) do:
      i. if the tuples (cid_{00}, P_i, b_{00}) , (cid_{01}, P_i, b_{01}),(cid_{10}, P_i, b_{10}), and (cid_{11}, P_i, b_{11}) are recorded then send (valid) to $\mathcal{F}_{\text{room}}$.
      ii. otherwise, send (invalid) to $\mathcal{F}_{\text{room}}$.
   (g) if the message is (Choose, cid, cid', P_j) do:
      i. if the tuples (cid, P_j, b) , (cid', P_j, b') and $\left(\text{read\_only}, cid_{bb'}, P_j, \widehat{b}\right)$ are recorded then send (valid) to $\mathcal{F}_{\text{room}}$.
      ii. otherwise, send (invalid) to $\mathcal{F}_{\text{room}}$.
   (h) if the message is (Local_open, cid, P_i) do:
      i. if the tuple (cid, P_i, b) is recorded then send (valid) to $\mathcal{F}_{\text{room}}$.
      ii. otherwise, send (invalid) to $\mathcal{F}_{\text{room}}$.
   (i) otherwise, send (valid) to $\mathcal{F}_{\text{room}}$.

Fig. 19: The functionality for verifying validity of action

Functionality $\mathcal{F}_{\text{valid}}$, described in Figure 19, is a filter of messages generated by the parties. It is a utility subroutine of $\mathcal{F}_{\text{db}}$ and models the validity of an action of a player at current time instant and ensures that all the needed components of the desired action are available; in particular it checks that all the boxes associated with this action are owned by the applicant. Here, cid is the commitment ID, used to distinguish among the different commitments that take place within a single instance of protocol**.** Presenting $\mathcal{F}_{\text{valid}}$ separately from $\mathcal{F}_{\text{db}}$ is done for the sake of simplicity.

Additionally, we model the actions available in the physical world. We need to capture the ability to create boxes with chosen or random content either privately or publicly. Furthermore, we need to capture the ability to pass a box to another player.

Functionality $\mathcal{F}_{\text{ccon}}$

Functionality $\mathcal{F}_{\text{ccon}}$ running with functionalities $\mathcal{F}_{\text{db}}$, $\mathcal{F}_{\text{room}}$ and proceeds as follows: At the first activation, verify that $sid = (\mathcal{P}, \mathcal{T}; sid')$, where $\mathcal{P}$ is an ordered set of m identities, and $\mathcal{T}$ is a set of functionality identities; else halt. Denote the parties identities $P_1, ..., P_m$ and the adversaries identities $\{\mathcal{A}_{(i,j)}\}_{i \neq j, i \in \mathcal{P}, j \in \mathcal{P}}$. Next:

1. Upon receiving input (Corrupt, P_i, p) from P_i related adversary $\mathcal{A}_{(i,j)}$ do:
   (a) record P_i as corrupted party and send (Corrupted, P_i) to P_i.
   (b) send to all P_i related adversaries all inputs given by P_i.
2. Upon receiving a value (Content, P_i, b) from $\mathcal{F}_{\text{room}}$ where $b \in \{0, 1\}$ proceed as follows:
   (a) generate a unique cid (for instance $cid = last\_cid + 1$)
   (b) record the tuple (cid, P_i, b) .
   (c) send the message (Receipt, cid, P_i) to $P_1, ..., P_m$ and all the adversaries.
3. Upon receiving a value (Rand_content, P_i) from $\mathcal{F}_{\text{room}}$, proceed as follows:
   (a) generate a unique cid
   (b) choose a random bit $b \in \{0, 1\}$, and record the tuple (cid, P_i, b).
   (c) send the message (Receipt_rand, cid, P_i) to $P_1, ..., P_m$ and all the adversaries.
4. Upon receiving a value (Public_content, P_i, b) from $\mathcal{F}_{\text{room}}$, proceed as follows:
   (a) generate a unique cid
   (b) record the tuple (cid, P_i, b).
   (c) send the message (Receipt_public, cid, P_i, b) to $P_1, ..., P_m$ and all the adversaries.
5. Upon receiving a value (Pass, cid, P_i, P_j) from $\mathcal{F}_{\text{room}}$, proceed as follows:
   (a) change record to (cid, P_j, b) where b is the recorded value.
   (b) send (Passed, cid, P_i, P_j) to $P_1, ..., P_m$ and all the adversaries.

Fig. 20: The functionality for encapsulating chosen or random content

Functionality $\mathcal{F}_{\text{ccon}}$, described in Figure 20, proceeds as follows:

- The "commitment to a chosen content" mode is modeled by having $\mathcal{F}_{\text{ccon}}$ receive a value (Content,P_i, b) from $\mathcal{F}_{\text{room}}$ (which in turn received it from P_i, the committer). Here $b \in \{0, 1\}$ is the value committed to. In response, $\mathcal{F}_{\text{ccon}}$ generates an unique commitment ID cid, lets all the parties know that P_i has committed to some value and that this value is associated with commitment ID cid. This is done by sending the message (Receipt, cid, P_i) to all the parties.
  The "publicly commit to a chosen content " mode is modeled similarly. This models the ability of players to choose a private/public value and publicly place it in a box.
- The "commitment to a random content" mode is modeled by having $\mathcal{F}_{\text{ccon}}$ receive a value (Rand_content, P_i) from $\mathcal{F}_{\text{room}}$. In response, $\mathcal{F}_{\text{ccon}}$ generates an unique commitment ID cid, randomly chooses a bit $b \in \{0, 1\}$, and informs all the parties that P_i has committed to some value. This is done in the same way as for commitments with chosen content.
  This models the ability of players to obtain a private random value using a designated machine and publicly place it in a box.

– The "commitment ownership passing" mode is modeled by having $\mathcal{F}_{\text{ccon}}$ receive a value (Pass, cid, P_i, P_j) from $\mathcal{F}_{\text{room}}$ (which in turn received it from P_i, the current owner). Here, in addition, the message includes the identity of the receiver. In response, $\mathcal{F}_{\text{ccon}}$ sends the message (Passed, cid, P_i, P_j) to all the parties.
This models the ability of players to publicly pass a box to other players.

Next, we model the addition, multiplication and duplication machines. This is done in a straightforward manner.

Functionality $\mathcal{F}_{\text{add}}$

Functionality $\mathcal{F}_{\text{add}}$ running functionalities $\mathcal{F}_{\text{db}}$, $\mathcal{F}_{\text{room}}$ proceeds as follows. At the first activation, verify that $sid = (\mathcal{P}, \mathcal{T}; sid')$, where $\mathcal{P}$ is an ordered set of m identities, and $\mathcal{T}$ is a set of functionality identities; else halt. Denote the parties identities $P_1, ..., P_m$ and the adversaries identities $\{\mathcal{A}_{(i,j)}\}_{i \neq j, i \in \mathcal{P}, j \in \mathcal{P}}$. Next:

1. Upon receiving input (Corrupt, P_i, p) from P_i related adversary $\mathcal{A}_{(i,j)}$ do:
   (a) record P_i as corrupted party and send (Corrupted, P_i) to P_i.
2. Upon receiving a value (Add, P_i, cid, cid') from $\mathcal{F}_{\text{room}}$, proceed as follows:
   (a) generate a unique $\hat{cid}$
   (b) record the tuple $\left(\hat{cid}, P_i, b + b'\right)$ where b,b' are the values associated with cid, cid' respectively.
   (c) delete the records (cid, P_i, b) and (cid', P_i, b') .
   (d) send (Added, cid, cid',$\hat{cid}$, P_i) to $P_1, ..., P_m$ and all the adversaries.
3. Upon receiving a value (Add_m_values, $cid_1, \ldots, cid_m$, P_i) from $\mathcal{F}_{\text{room}}$, proceed as follows:
   (a) compute $result = \sum_{j=1}^{m} b_j$ where $b_j$is the value associated with cid_j.
   (b) generate a unique $\hat{cid}$
   (c) record the tuple $\left(\hat{cid}, P_i, result\right)$
   (d) delete the records $(cid_1, P_i, b_1), \ldots, (cid_m, P_i, b_m)$
   (e) send (Added_m_values, $cid_1, \ldots, cid_m$,$\hat{cid}$, P_i) to $P_1, \ldots, P_m$ and all the adversaries.

Fig. 21: The functionality computing addition

Functionality $\mathcal{F}_{\text{add}}$, described in Figure 21, proceeds as follows:

– The "add two committed contexts" mode is modeled by having $\mathcal{F}_{\text{add}}$ receive a message (Add, P_i, cid, cid') from $\mathcal{F}_{\text{room}}$. Here cid,cid' are the commitment ID's of the addends, and $\hat{cid}$ is the commitment ID of the result. In response, $\mathcal{F}_{\text{add}}$ generates an unique $\hat{cid}$ (commitment ID of the result), records the result and delete the records that have been used. Then $\mathcal{F}_{\text{add}}$ lets all parties know that P_i has added two values associated with commitment IDs cid and cid', and that the resulting value is associated with commitment ID $\hat{cid}$. The "add m committed contexts" mode is similar.

Functionalities $\mathcal{F}_{\text{mult}}$ and $\mathcal{F}_{\text{duplicate}}$, described in Figures 22 and 23 proceed in the same way as $\mathcal{F}_{\text{add}}$, *mutatis mutandis*.

Functionality $\mathcal{F}_{\text{mult}}$

Functionality $\mathcal{F}_{\text{mult}}$ running with functionalities $\mathcal{F}_{\text{db}}$,$\mathcal{F}_{\text{room}}$ and proceeds as follows. At the first activation, At the first activation, verify that $sid = (\mathcal{P}, \mathcal{T}; sid')$, where $\mathcal{P}$ is an ordered set of m identities, and $\mathcal{T}$ is a set of functionality identities; else halt. Denote the parties identities $P_1, ..., P_m$ and the adversaries identities $\left\{\mathcal{A}_{(i,j)}\right\}_{i\neq j, i\in\mathcal{P}, j\in\mathcal{P}}$. Next:

.

1. Upon receiving input (Corrupt, P_i, p) from P_i related adversary $\mathcal{A}_{(i,j)}$ do:
   (a) record P_i as corrupted party and send (Corrupted, P_i) to P_i.
2. upon receiving a value (Multiply, P_i, cid, cid') from $\mathcal{F}_{\text{room}}$, do:
   (a) generate a unique $\hat{cid}$
   (b) record the tuple $\left(\hat{cid}, P_i, b \cdot b'\right)$
   (c) delete the records (cid, P_i, b) and (cid', P_i, b') .
   (d) send (Multiplied,cid, cid', $\hat{cid}$, P_i) to $P_1, ..., P_m$ and all the adversaries.

Fig. 22: The functionality computing multiplication

Functionality $\mathcal{F}_{\text{duplicate}}$

Functionality $\mathcal{F}_{\text{duplicate}}$ running with functionalities $\mathcal{F}_{\text{db}}$, $\mathcal{F}_{\text{room}}$ and proceeds as follows.At the first activation, At the first activation, verify that $sid = (\mathcal{P}, \mathcal{T}; sid')$, where $\mathcal{P}$ is an ordered set of m identities, and $\mathcal{T}$ is a set of functionality identities; else halt. Denote the parties identities $P_1, ..., P_m$ and the adversaries identities $\left\{\mathcal{A}_{(i,j)}\right\}_{i\neq j, i\in\mathcal{P}, j\in\mathcal{P}}$. Next:

.

1. Upon receiving input (Corrupt, P_i, p) from P_i related adversary $\mathcal{A}_{(i,j)}$ do:
   (a) record P_i as corrupted party and send (Corrupted, P_i) to P_i.
2. Upon receiving a value (Duplicate, P_i, cid) from $\mathcal{F}_{\text{room}}$, proceed as follows:
   (a) generate a unique $\hat{cid}$
   (b) record the tuple $\left(\hat{cid}, P_i, b\right)$
   (c) send (Duplicated, cid, $\hat{cid}$, P_i) to $P_1, ..., P_m$ and all the adversaries.

Fig. 23: The functionality for duplicating records

Functionality $\mathcal{F}_{\mathrm{COT}}$

Functionality $\mathcal{F}_{\mathrm{COT}}$ running with functionalities $\mathcal{F}_{\mathrm{db}}$, $\mathcal{F}_{\mathrm{room}}$ and proceeds as follows. At the first activation, At the first activation, verify that $sid = (\mathcal{P}, \mathcal{T}; sid')$, where $\mathcal{P}$ is an ordered set of m identities, and $\mathcal{T}$ is a set of functionality identities; else halt. Denote the parties identities $P_1, ..., P_m$ and the adversaries identities $\left\{\mathcal{A}_{(i,j)}\right\}_{i\neq j, i\in\mathcal{P}, j\in\mathcal{P}}$. Next:

.

1. Upon receiving input (Corrupt, P_i, p) from P_i related adversary $\mathcal{A}_{(i,j)}$ do:
   (a) record P_i as corrupted party and send (Corrupted, P_i) to P_i.
2. Upon receiving a value (Shuffle, cid_{00},cid_{01}, cid_{10}, cid_{11}, P_i, P_j) from $\mathcal{F}_{\mathrm{room}}$, proceed as follows:
   (a) change records to $(\mathrm{read\_only}, cid_{00}, P_j, b_{00})$, $(\mathrm{read\_only}, cid_{01}, P_j, b_{01})$, $(\mathrm{read\_only}, cid_{10}, P_j, b_{10})$, and $(\mathrm{read\_only}, cid_{11}, P_j, b_{11})$.
   (b) send (Passed_read_only, cid_{00}, cid_{01}, cid_{10}, cid_{11}, P_i, P_j) to $P_1, ..., P_m$ and all the adversaries.
3. Upon receiving a value (Choose,cid', cid'',P_j) from $\mathcal{F}_{\mathrm{room}}$ proceed as follows:
   (a) generate a unique $\hat{cid}$
   (b) record the tuple $\left(\hat{cid}, P_j, \widehat{b}\right)$ ($\widehat{b}$ is the value inside box $cid_{cid'cid''}$)
   (c) delete the records $(\mathrm{read\_only}, cid_{11}, P_j, b_{11})$, $(\mathrm{read\_only}, cid_{01}, P_j, b_{01})$, $(\mathrm{read\_only}, cid_{00}, P_j, b_{00})$, and $(\mathrm{read\_only}, cid_{10}, P_j, b_{10})$
   (d) send the message (Chosen,cid', cid'', $\hat{cid}$, P_j) to $P_1, ..., P_m$ and all the adversaries.

Fig. 24: The functionality for shuffling

Next, we model Oblivious Transfer (OT), which was described in 7.1.1 and presented in Figure 24. Functionality $\mathcal{F}_{\mathrm{COT}}$**,** which is used to models OT, proceeds as follows.

- The permuting operation is modeled by having $\mathcal{F}_{\mathrm{COT}}$ receive a value (Shuffle, cid_{00},cid_{01}, cid_{10}, cid_{11}, P_i, P_j) from $\mathcal{F}_{\mathrm{room}}$. Here, P_j is the receiver. In response, $\mathcal{F}_{\mathrm{COT}}$ changes the owner of the commitment to P_j and the permission to "read-only". Then, it lets all parties know that P_j is the new owner of the "read-only" commitments.
- The choosing operation is modeled by having $\mathcal{F}_{\mathrm{COT}}$ receive a value (Choose, cid', cid'',P_j) from $\mathcal{F}_{\mathrm{room}}$. In response, $\mathcal{F}_{\mathrm{COT}}$ generates an unique $\hat{cid}$ (new commitment ID for the chosen commitment), records the commitment associated with the values inside cid', cid'' with it new ID, deletes the "read-only" commitments and informs the parties that P_j has chosen a commitment that associated with commitment ID $\hat{cid}$.
  This ensures that all players have no clue regarding the serial number of the chosen box (which is uniquely determined by the values inside cid' and cid'') but they can verify that only one box was chosen.
- Destroying the boxes after choosing one of them is modeled by having $\mathcal{F}_{\mathrm{COT}}$ delete all records associated with the OT after the receiving party chose his box. Deleting all records ensures that no additional information regarding sender's input to the OT will be revealed.

Finally, we would like to capture the ability of players to publicly open a box they possess via the opening machine, while keeping its content private. This is done in a straightforward manner.

Functionality $\mathcal{F}_{\mathrm{local\_open}}$, described in Figure 25, proceeds as follows. $\mathcal{F}_{\mathrm{local\_open}}$ is initiated by receiving a message (Local_open, cid, P_i) from $\mathcal{F}_{\mathrm{room}}$. In response, $\mathcal{F}_{\mathrm{local\_open}}$ hands the value (Open, cid, P_i, b) to P_i and lets all parties know that P_i has opened a commitment he owns with the serial number cid. In addition, the opened record is destroyed.

Functionality $\mathcal{F}_{\text{local\_open}}$

Functionality $\mathcal{F}_{\text{local\_open}}$ running with functionalities $\mathcal{F}_{\text{db}}$, $\mathcal{F}_{\text{room}}$ and proceeds as follows. At the first activation, At the first activation, verify that $sid = (\mathcal{P}, \mathcal{T}; sid')$, where $\mathcal{P}$ is an ordered set of m identities, and $\mathcal{T}$ is a set of functionality identities; else halt. Denote the parties identities $P_1, ..., P_m$ and the adversaries identities $\left\{\mathcal{A}_{(i,j)}\right\}_{i \neq j, i \in \mathcal{P}, j \in \mathcal{P}}$. Next:

.

1. Upon receiving input (Corrupt, P_i, p) from P_i related adversary $\mathcal{A}_{(i,j)}$ do:
   (a) record P_i as corrupted party and send (Corrupted, P_i) to P_i.
   (b) send all P_i related adversaries any output that was given to P_i.
2. Upon receiving a value (Local_open, cid, P_i) from $\mathcal{F}_{\text{room}}$,do:
   (a) if P_i corrupted then send the message (Open, cid, P_i, b) to all P_i related adversaries; otherwise, sent it to P_i.
   (b) delete the record (cid, P_i, b).
   (c) send (Opened, cid, P_i) to $P_1, ..., P_m$ and all the adversaries.

Fig. 25: The functionality that locally opens messages

We clarify that all the ideal functionalities presented communicate among themselves via secure channels. In addition, all record, delete, duplicate and ownership requests are done by sending an appropriate request to $\mathcal{F}_{\text{db}}$. The relations between functionalities are presented in Figure 26.

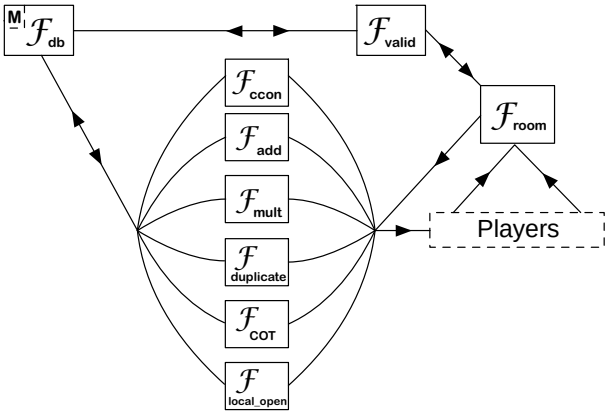

Fig. 26: The hierarchy of the ideal functionalities. A bidirectional arrows represent bidirectional communication where one directional arrow represent communication in the arrow direction.

7.1.3 An ideal functionality for secure function evaluation

We would like an ideal functionality that would capture multi-party computation in the physical model presented[3] above. This ideal functionality proceeds in rounds, and at each round may leak information to the adversarial parties.

[3] For simplicity we restrict ourselves to PPT functions instead of functionalities

We would like to capture an ideal process for securely evaluating a function in a physical computation model. We would like to allow parties to act simultaneously at all times, therefore the functionality proceeds in rounds. At each round all parties may act if they wish to. Since each action can depend only on actions that precede it, we can linearize and discretize the actions through the use of rounds.

Moreover, we would like all actions to be transparent to all participants, that is: all parties should be aware of the actions taken so far. Our goal is to maintain the inputs' and outputs' privacy while making the action public. To that end, we have the functionality inform all parties and adversaries about any defection. This ideal functionality formally presented in Figure 27.

.

7.2 The Physical-GMW protocol

We now present a construction for m parties to compute any function f that is expressed by an arithmetic circuit which consists of OR and AND gates. We assume that there are m parties $P_1, ..., P_m$, and each input consists of n bits. We further assume that each party holds the values of some of the input wires to the circuit, and in addition there are random input wires.

Our construction is similar to the GMW protocol in the honest-but-curious setting, where the goal is to propagate, via private computation, shares of the input wires of the circuit to shares of all wires of the circuit, so that finally we obtain shares of the output wires of the circuit. In contrast, our construction will be resilient to Byzantine adversaries. In all steps of the protocol party P_i starts his computation only after party P_{i-1} had finished and all the messages generated by the parties $P_1, ..., P_m$ are send through $\mathcal{F}_{\text{room}}$.

The key points of the construction are that at any moment there exists only one legal action which should be performed by some specific party and verified by all the others. Thus, this protocol has exactly one valid transcript (which is unavoidable in order to prevent corrupted parties from signaling). In addition, no party can deviate by opening commitments it owns since the opening functionality is activated only in the last round of the protocol. Moreover, in this round the only commitments available to parties are the ones that associated with their outputs and the only functionality available is $\mathcal{F}_{\text{local\_open}}$. Therefore, no party can discover its output while preventing output from others. In other words, the protocol guarantees complete fairness.

7.2.1 The protocol Π^f_{GMW}

Let f be some PPT function and let C_f be the analogous arithmetic circuit. Let $\mathcal{P} = P_1, ..., P_m$ denote the parties identities and x_i denote the input of party P_i.

Step 0: Insuring the only faults are "abort faults" Since all the commitment ID's (cid's) are generated by $\mathcal{F}_{\text{ccon}}$, $\mathcal{F}_{\text{add}}$, $\mathcal{F}_{\text{mult}}$ and $\mathcal{F}_{\text{duplicate}}$ sequentially and each ideal functionality sends all the commitment IDs involved in the operation it performs ($\mathcal{F}_{\text{ccon}}$ in "pass", $\mathcal{F}_{\text{add}}$ in "Add" etc.) to all parties, each party can verify that all the parties are using the right boxes for the operations ("pass", "add", etc) by reconstructing the sequence of ID generation. If some party P_j doesn't follow the protocol then all parties will detect that and abort (this cheating will be included in parties transcript). Moreover, all operations, the ID's of the parties involved in them, and the commitment ID's associated with them are fully public. Hence, each party can verify that the protocol is followed properly.

Step 1: Sharing the inputs Each party P_i (in increasing order of i) splits each of its input bits. That is, for every input bit x^i_j for $j = 1, ..., n$ and $k \neq i$ (in increasing order of k) the party P_i generates a commitment to a random bit r^k_j using $\mathcal{F}_{\text{ccon}}$. Then, party P_i sets his own share of his j-th input wire by generating a commitment to $x^i_j + \sum_{k \neq i} r^k_j$ by adding two values repeatedly using $\mathcal{F}_{\text{add}}$ (the party will duplicate the shares before using $\mathcal{F}_{\text{add}}$). Then, each party P_i (in increasing order of i) shares each of its

Functionality $\mathcal{F}_{\text{psfe}}^{f}$

Functionality $\mathcal{F}_{\text{psfe}}^{f}$ proceeds as follows, given a function $f : (\{0,1\}^* \cup \{\perp\})^m \times R \to (\{0,1\}^*)^m$. At the first activation, verify that $sid = (\mathcal{P}; sid')$, where $\mathcal{P}$ is an ordered set of m identities; else halt. Denote the parties identities $P_1, ..., P_m$, adversaries identities $\{\mathcal{S}_{(i,j)}\}_{i \neq j, i \in \mathcal{P}, j \in \mathcal{P}}$ and let pid' be the first PID in $\mathcal{P}$.Also, initialize variables $x_1, ..., x_m, y_1, ..., y_m$ to default value $\perp$, $currpid = pid'$ and output a `continue` message to all $\{\mathcal{S}_{(pid',j)}\}_{pid' \neq j, j \in \mathcal{P}}$ (referred as $currpid$ related adversaries).

Next:

1. INPUT: Upon receiving input (Input,v) from party P_i then set $x_i = v$ and output (Input, P_i) to all P_i related adversaries.
2. RESPONSE: Upon receiving message `(response)` from all $currpid$ related adversaries do:
   (a) if $currpid$ is not the last PID in $\mathcal{P}$ then:
      i. record responses
      ii. advance $currpid$ and output a `continue` message to all $currpid$ related adversaries.
   (b) else do as follows:
      i. set $currpid = pid'$
      ii. if all `response=okay` send an `advance` message to all adversaries and a `continue` message to all $currpid$ related adversaries.
      iii. if at least one `response=abort (at least one party aborted the game)`
         A. output the identities of the aborting parties to all parties and adversaries.
      iv. if at least one `response=cheat (at least one party deviated)` do:
         A. output the identities of the cheating parties followed by the description of the defections to all parties and adversaries.
      v. if at least one `response=output` do:
         A. if x_i has been set for all parties in $\mathcal{P}$, and $y_1, ..., y_n$ have not yet been set, then choose $r \leftarrow R$ and set $(y_1, .., y_n) \leftarrow f(x_1, ..., x_n, r)$; else if x_i has not been set for all parties, ignore .
         B. for each output request: if the party is corrupted then output y_i to all PID related adversaries; else output y_i to the party.
         C. output the identities of the output requesting parties to all other parties and adversaries.
      vi. if (iii) or (iv) happened and none of the parties received output then halt; otherwise, output `continue` message to all $currpid$ related adversaries.
3. CORRUPT: Upon receiving input (Corrupt, P_i, p) from P_i related adversary do:
   (a) record P_i as corrupted party and send (Corrupted, P_i) to P_i.
   (b) give x_i, output that was given to P_i (if any) and the identity of the corrupter to all P_i related adversaries.
4. DELIVER: Upon receive input (Deliver, $\mathcal{S}_{(\ell,k)}$, m) from $\mathcal{S}_{(i,j)}$ and P_i is corrupted then:
   (a) if $i = \ell$ then output (Deliver, $\mathcal{S}_{(i,j)}$, $\mathcal{S}_{(\ell,k)}$, m) to $\mathcal{S}_{(\ell,k)}$.
5. SET INPUT: Upon receiving input `(set_input,` P_i`,` v'`)` from P_i related adversary do:
   (a) if P_i is corrupted and none of the parties received output then set $x_i \leftarrow v'$; otherwise ignore.

Fig. 27: The functionality for secure function evaluation

input bits with all other parties (in order): P_i sends to party P_k its share of the j-th input wire of party P_i using $\mathcal{F}_{\text{ccon}}$. Additionally, each party P_i (in increasing order of i) generates a commitment to a random bit using $\mathcal{F}_{\text{ccon}}$ for each of his random input wires. To guarantee total order, P_i will start only after P_{i-1} finishes.

During this process, each party verifies, as described in **Step 0**, that the proceedings are done correctly.

Step 2: Circuit emulation Proceeding by the order of wires in C_f, the parties use their commitments to the two input wires to a gate in order to privately compute shares for the output-wire of the gate. Suppose that the parties hold the commitments that correspond to the two input wires of some gate: that is, for $i = 1, ..., m$, party P_i holds the commitments to the shares a_i, b_i, where $a_1, ..., a_m$ are the shares of the first wire, and $b_1, .., b_m$ are the shares of the second wire.

Since the boxes are destroyed after use, each party (in increasing order) duplicates the shares it is about to use the exact number of times those shares will be needed in the future (and only if it didn't duplicate those shares before). Next, we consider the usual two cases:

Evaluation of OR gate:

Each party (in increasing order) sets its output wire to be $a_i + b_i$ by using $\mathcal{F}_{\text{add}}$. In other words, each party commits to the XOR of a_i and b_i using the functionality $\mathcal{F}_{\text{add}}$. That way, the shared output will be given by

$$\sum_{i=1}^{m}(a_i + b_i) = a + b$$

and that is the desired true output.

Evaluation of AND gate:

The true value is given by

$$\left(\sum_{i=1}^{m} a_i\right) \cdot \left(\sum_{i=1}^{m} b_i\right) = \sum_{i=1}^{m} a_i b_i + \sum_{i=1}^{n} \sum_{i<j\leq m} (a_i b_j + a_j b_i)$$

Thus, if every party P_i will have his share set to

$$w_i = a_i b_i + \sum_{i<j\leq m} (a_i b_j + a_j b_i)$$

then the true output will be given by $\sum w_i$.

Since each party P_i owns commitments to a_i, b_i it is left with the task of finding out the value of $a_i b_j + a_j b_i$ for every $i < j$. For every i, j such that $i < j$, it is P_j's responsibility to help the party P_i learn $a_i b_j + a_j b_i$. The solution is to use $\mathcal{F}_{\text{COT}}$: For each $j > i$ in increasing order of j the party P_j will generate "read only" commitments to $a_i b_j + a_j b_i$ for every possible combination of (a_i, b_i) by publicly committing twice to 0 and then twice to 1 using $\mathcal{F}_{\text{ccon}}$ and then using $\mathcal{F}_{\text{add}}$ and $\mathcal{F}_{\text{mult}}$. Then, using $\mathcal{F}_{\text{shuffle}}$, it will transfer the commitment to the correct combination to party P_i by passing all four commitments to P_i and allowing P_i to choose the one that corresponds to his shares. That way, the i-th party will obtain a commitment to the j-th share it needs in order to compute its own share using the suitable ideal functionalities.

Again, as in **Step 1** each party P_k verifies that P_i was using the right values (commitments) in the evaluation (in other word, in case of the OR gate evaluation, P_i indeed set w_i to be $a_i + b_i$ and not $a_{i+1} + b_{i-1}$for example), and that the calculations are done in the right order. And again, this is done as described in **Step 0**.

Step 3: Output reconstruction Once the shares of C_f output wires are computed, each party passes its commitment to the share of each such wire to the party with which the wire is associated (in increasing order). This is done using $\mathcal{F}_{\text{ccon}}$. Once again the each party verifies that the right commitment was sent to the right party (as described in **Step 0**).

Then, in an order described before, each party adds all m shares using $\mathcal{F}_{\text{add}}$. Lastly, all parties open their box containing the sums and discover the function output using $\mathcal{F}_{\text{local\_open}}$ (again, this is done in an order described above but here parties do not verify the behavior of others and do not abort due to inappropriate behavior).

Again, each party P_k verifies that P_i was using the right values (commitments) in the evaluation and in the correct order (this is done as described in **Step 0**).

This completes the description of Π^f_{GMW} and now we are ready to prove the following theorem:

Theorem 10. *Let f be a PPT function. Then, protocol Π^f_{GMW} LUC-realizes $\mathcal{F}^f_{\text{psfe}}$ in the presence of adaptive adversaries where the corruptions are done PID-wise. Moreover, Π^f_{GMW} guarantees complete fairness .*

On the use of cameras. The reader should note that opposed to [ILM05] our protocol can handle up to n corruptions and not $n-1$. This implies that signaling is prevented even if all the parties are corrupted. We do not establish security on the existence of honest party to detect inappropriate behavior. Instead, we introduce the camera entity which takes the responsibility of honest parties to perform detection. This entity enforces honest behavior not only by detection but also by controlling the activation of machines. Although, it might considered as unusual assumption, the existence of such external observer is essential to obtain the ability to reason about game theoretic solution concepts which require absence of coalitions.

7.2.2 Proof sketch

The proof is strongly relies on parties common knowledge of the unique legal behavior enforced by the protocol and on the ability to verify each step in the execution. Moreover, no matter when a party was corrupted we know that up to this point all parties honestly followed the protocol.

We prove this theorem with respect to the dummy adversary $\mathcal{D}$ by showing that there exists an simulator $\mathcal{S}_{\mathcal{D}}$ such that no environment $\mathcal{Z}$ can distinguish between $\text{LEXEC}_{\text{LIDEAL}_{\mathcal{F}^f_{\text{psfe}}},\mathcal{S}_{\mathcal{D}},\mathcal{Z}}$ and $\text{LEXEC}_{\Pi^f_{\text{GMW}},\mathcal{D},\mathcal{Z}}$. In our case, the output ensembles will be identically distributed.

High-level description of the simulator $\mathcal{S}_{\mathcal{D}}$.
The simulation is done in a round-by-round manner. Since the ideal functionality $\mathcal{F}^f_{\text{psfe}}$ enforces round discipline, no simulator will proceed to round k unless all the simulators successfully passed round $k-1$ (where "successfully passed" is that all the parties neither cheated nor aborted in round $k-1$). We call the output given by $\mathcal{F}^f_{\text{psfe}}$ regarding the aborting and cheating of parties (i.e., advance/abort/cheat messages from $\mathcal{F}^f_{\text{psfe}}$) at current round as "round leakage".

Simulation of honest party P_i. Since all the communication is done via ideal functionalities which reports to all entities, the simulator need only to send `response=okay` messages and `response=output` message (when it is P_i's turn to receive output from $\mathcal{F}_{\text{local\_open}}$) to $\mathcal{F}^f_{\text{psfe}}$ unless the party is corrupted at some later time. In addition, $\mathcal{S}_{\mathcal{D}}$ generates an appropriate messages as in Π^f_{GMW} and forward them to $\mathcal{Z}$ based on the output message from $\mathcal{F}^f_{\text{psfe}}$ regarding current round leakage. This can be easily done since at each round there is only one action associated with an `advance` message, where any other type of the messages contain all the information needed. If the party is later corrupted, the simulator receives party's input and any output it received from $\mathcal{F}^f_{\text{psfe}}$ and can construct its internal state (see internal state construction).

Internal state construction. When party P_i become corrupted, the simulator need to generate an internal state which reflects a honest participation of the party in Π^f_{GMW} up to the corruption point. Upon receiving an input (Corrupt, P_i, p) $\mathcal{S}_{\mathcal{D}}$ forward it to $\mathcal{F}^f_{\text{psfe}}$. Recall that the only private information

available to P_i is its input (to which P_i committed in step 1) and the output received from $\mathcal{F}_{\text{local\_open}}$. However, $\mathcal{S}_{\mathcal{D}}$ receives this information from $\mathcal{F}_{\text{psfe}}^{f}$ and can easily construct the internal state based on the well-known legal transcript.

Corrupted party simulation. Here we analyze two possible scenarios: In case $\mathcal{S}_{\mathcal{D}}$ is not the corrupter of P_i then it forwards all inputs from $\mathcal{Z}$ to the corrupter. This is done using the Deliver service provided by $\mathcal{F}_{\text{psfe}}^{f}$. all outputs received from $\mathcal{F}_{\text{psfe}}^{f}$ are treated as in the "honest party simulation".

When $\mathcal{S}_{\mathcal{D}}$ is the corrupter of P_i, it honestly plays the role of $\mathcal{F}_{\text{room}}$. More formally, upon receiving a `continue` message the simulation of round k proceed as follows:

When received round k instruction, $\mathcal{S}_{\mathcal{D}}$ informs the ideal functionality $\mathcal{F}_{\text{psfe}}^{f}$ about this by sending an appropriate message (a `cheat` message would be accompanied by a deviating message that $\mathcal{S}_{\mathcal{D}}$ generates by honestly simulating the response of the suitable ideal functionality that the instruction was aimed for). Note that if the instructed action is unavailable for the party at current round according to $\mathcal{F}_{\text{room}}$ and $\mathcal{F}_{\text{valid}}$ (which $\mathcal{S}_{\mathcal{D}}$ honestly simulates) then $\mathcal{S}_{\mathcal{D}}$ sends `okay` to $\mathcal{F}_{\text{psfe}}^{f}$**.** Based on the leakage of $\mathcal{F}_{\text{psfe}}^{f}$ in this round $\mathcal{S}_{\mathcal{D}}$ simulates the responses of all functionalities used at this round (as done in "honest party simulation").

In addition, during the simulation of **Step 1** (of the protocol), $\mathcal{S}_{\mathcal{D}}$ plays the role of $\mathcal{F}_{\text{ccon}}$. In this step, $\mathcal{Z}$ should commitment to chosen content. In the process, $\mathcal{S}_{\mathcal{D}}$ will discover the value x_i' that $\mathcal{Z}$ uses as an input to the protocol and will update P_i's input sending `(set_input,`P_i`,` x_i'`)` to $\mathcal{F}_{\text{psfe}}^{f}$. All of this holds only if $\mathcal{Z}$ plays the protocol correctly (neither aborts nor cheats) until this phase.

Output distribution indistinguishability.
All the parties' actions are public since all actions are done via the ideal functionalities (capturing the corresponding physical actions), which inform all the parties regarding the illegal actions other parties do. By the protocol's construction, there is only one valid run (transcript) of the protocol. Therefore, as mentioned previously, any party at any time can verify (even during the execution of the protocol) that the protocol executed correctly: that is, no cheating can go undetected.

On the one hand, if there occurred cheating/aborting at some round k in the real world, then in the ideal world, once the simulation reaches round k, we will discover the defection via the leakage provided by the ideal functionality $\mathcal{F}_{\text{psfe}}^{f}$ (as described in the simulation) and generate output according to the output of the suitable ideal functionalities in the real world. Note that no cheating can occur in the opening the output wires phase since the only available functionality is $\mathcal{F}_{\text{local\_open}}$ and the only available commitment is the output. Furthermore, causing the protocol to abort at this phase is impossible. Due to the leakage and the round-by-round simulation, the environment view is *identical* to its view in the real world. Moreover, the views of all internal adversaries are consistent with each other.

On the other hand, if there was no deviation, the simulators play honestly, making the view of the environment in the simulation *identical* to the view of the real-world. The protocol has only one legal transcript, which all the simulator know. The transcripts produced by the simulators would be the legal transcripts, which are consistent with each other, and the union of these transcripts represents the only legal execution of the protocol. In particular, the output of the function would be same as in the real world.

8 A solution via semi-trusted mediator

This section presents general constructions for LUC-realizing any multi-party ideal functionality in the presence of adaptive adversaries. The high-level construction is basically that of Alwen et. al. [AKL$^+$09]. The idea is to add a shared subroutine, called the *mediator*, which the parties partially trust: if the mediator is honest, the resulting LUC-security guarantees isolation between players. Moreover, if the mediator is dishonest it can harm only the separation guarantee; but neither the security nor the correctness is damaged. However, there are some differences (that also differentiate our feasibility result from [AKMZ12]). First, Alwen et. al. consider static adversaries whereas we consider adaptive adversaries. The second difference between the Alwen et. al. construction and ours is that while Alwen et. al. consider super-ideal

channels, where the adversary is not aware if a communication occurred, we consider more realistic setting where any communication leaks information to the adversaries and controlled by them. We note that although the protocols are similar, our proof of security is significantly different. In particular, we have to deal with the fact that the revealed communication and the adversarial capabilities might be used as additional means of immunity testing.

We begin by presenting the components of our construction. Following this we present the protocol for LUC-realizing any multi-party functionality in presence of static adversaries. Then we extend the protocol to the adaptive case.

8.1 Functionalities

The first step is to define the set of functionalities for which our feasibility results apply. As in the work of [CLOS02,CDPW07] we put some restrictions on the ideal functionalities to be realized . We consider the following class of ideal functionalities:

Well-formed functionalities. Informally, a well-formed functionality allows the adversary to delay the generation of outputs to uncorrupted parties and receive the length of their inputs. In addition, it allows the adversary to control the inputs and outputs of all corrupted parties. The formal definition is presented in [CLOS02].

Aborting functionalities. A functionality $\mathcal{F}$ is aborting if the ideal protocol $\mathrm{LIDEAL}_{\mathcal{F}}$ allows the adversary to abort the execution at any time by giving special input $\perp$ to $\mathcal{F}$. Moreover, the adversary allowed deciding which of the honest parties outputs $\perp$.

Non-reactive functionalities. For sake of simplicity we restrict ourselves to non-reactive functionalities. In other words, we consider functionalities for secure function evaluation.

8.2 Tools and assumptions

We review the functionalities and the underlying protocols used within our construction.

8.2.1 The LUC functionalities

The functionalities presented below are standard, well known functionalities that are extended to a multiple adversary setting. We start by formally defining in LUC the widely use term of delayed-output.

Delayed-output. A delayed-output to some party is an output that its delivery is adversarially scheduled. More formally, in a centralized adversary frameworks a functionality said to generate a delayed output m to party P if: Whenever the output m is ready to be outputted to P by the functionality, it sends (OUTPUT, P, p) to the adversary, where p is optional parameters. Later, whenever receiving an input of the form $(\mathrm{APPROVE}, P, p)$ from the adversary, the functionality writes m to the subroutine output tape of P. A public delayed output reveal the output m to the adversary, where private delayed output not. We extend this definition to the LUC framework for two-party functionalities as follows: Let P_1, P_2 denote the party-identities, and $\mathcal{A}_{(1,2)}$, $\mathcal{A}_{(2,1)}$ denote the appropriate adversaries. Functionality $\mathcal{F}$ generates delayed output m to P_i by sending $(\mathrm{OUTPUT}, P_i, p)$ first to $\mathcal{A}_{(i-1,i)}$ and upon receiving its confirmation $\mathcal{F}$ sends $(\mathrm{OUTPUT}, P_i, p)$ to $\mathcal{A}_{(i,i-1)}$. Once $\mathcal{A}_{(i,i-1)}$ approves the output delivery, $\mathcal{F}$ writes m to the subroutine output tape of P_i.

Secure Message Transmission functionality. Informally, in secure message transmission the adversary has no access to the content of the transmitted message and only learns that a message was send. In addition, the transmission is ideally authenticated. The original formulation of the secure message transmission functionality $\mathcal{F}_{\mathrm{smt}}$ can be found in [Can01]. We extend $\mathcal{F}_{\mathrm{smt}}$ to the LUC framework by defining the analog merger functionality $\mathcal{F}_{\overline{\mathrm{smt}}}$ as in Definition 5. The functionality is formally presented in Figure 28.

Functionality $\mathcal{F}_{\overline{\text{smt}}}$

Functionality $\mathcal{F}_{\overline{\text{smt}}}$ running with parties S, R and the adversaries $\mathcal{S}_{(S,R)}$, $\mathcal{S}_{(R,S)}$, proceeds as follows:

- Upon receiving an input (Send, sid, m) from party S do: verify that $sid = (S, R, sid')$, else ignore the input. Next, record m, and send a private delayed output (Send, sid, m) to R. Once m is recorded, ignore any subsequent $(\text{Send}, ...)$ inputs.
- Upon receiving an input `(Corrupt, sid, T)` ,where $T \in \{S, R\}$, from an appropriate adversary do: if $T = S$ then disclose m to $\mathcal{S}_{(S,R)}$ and $\mathcal{S}_{(R,S)}$. Furthermore, if $\mathcal{S}_{(S,R)}$ now provides an input m' and the output was not yet written on R's tape, then change recorded message to m'. In any case, output `(Corrupted, sid, T)` to the newly corrupted party.
- Upon receiving an input (Deliver, $\mathcal{S}_{(j,i)}$, m) from $\mathcal{S}_{(i,j)}$ do: verify that $\mathcal{S}_{(i,j)}$ and $\mathcal{S}_{(j,i)} \in \{\mathcal{S}_{(S,R)}, \mathcal{S}_{(R,S)}\}$, else ignore the input. output (Deliver, $\mathcal{S}_{(j,i)}$,$\mathcal{S}_{(i,j)}$, m) to $\mathcal{S}_{(j,i)}$.

Fig. 28: The secure message transmission functionality $\mathcal{F}_{\overline{\text{smt}}}$

Augmented CRS functionality. Recall the LUC augmented CRS functionality $\bar{\mathcal{G}}_{\overline{\text{acrs}}}$ presented in Figure 5. Similarly to the UC functionality $\bar{\mathcal{G}}_{\text{acrs}}$, it holds a single short "master public key", and guarantees that the parties who follow their protocols never obtain their secret keys. In addition, any retrieve or CRS request done by some party is notified to all its adversaries. It is important to note that $\bar{\mathcal{G}}_{\overline{\text{acrs}}}$ do not enable the adversaries to exploit its services for signaling.

8.2.2 The underlying protocol

Now, we are ready to define a general class of protocols and show that any well-formed aborting functionality $\mathcal{F}$ can be UC-realized by some protocol in this class. Our feasibility result is applied with respect to this class of protocols.

$\mathcal{G}$-Setup oblivious protocols. A protocol π is $\mathcal{G}$-setup oblivious if it consists of a setup-phase, in which all the interaction between parties and $\mathcal{G}$ happens. Moreover, this phase precedes any other communication in the protocol, and any incoming communication is ignored until the end of the setup-phase.

Claim. For any adaptively well-formed aborting ideal functionality $\mathcal{F}$, there exists a $\bar{\mathcal{G}}_{\text{acrs}}$-setup oblivious protocol π that UC-realizes $\mathcal{F}$ in the $\bar{\mathcal{G}}_{\text{acrs}}$-hybrid model in the presence of static adversaries over unauthenticated channels, where all corruptions are PID-wise.

The claim follows by composing the [CDPW07] protocol for realizing any well-formed aborting functionality over authenticated channels with the authentication protocol in [Wal08]. The obtained protocol in the $\bar{\mathcal{G}}_{\text{acrs}}$-hybrid model can be modified to a new protocol where all the interaction with $\bar{\mathcal{G}}_{\text{acrs}}$ performed offline.

A protocol π that UC-realizes $\mathcal{F}$. Let π be an n-party protocol as in Claim 8.2.2. We assume the following about π, all of which can be ensured using standard techniques:

- The protocol π consists of a sequence of rounds. It has a fixed, known number of rounds and every party learns its output in the final round of the protocol.
- All messages in π include the round number it associated with. In any given round each party send fixed length messages to all other parties (at once) and these messages do not depend on messages associated with the current round.

8.3 Oblivious computation of π

The general structure of the compiler, as described in the introduction, has the mediator send to each party P *commitments* (using a commitment scheme C) to its protocol messages of π. Thus, P cannot

compute its π-messages directly, but rather it is done by executing a two-party protocol with the mediator. Specifically, we define a two-party functionality $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ that computes the next π-message of P and enables P to obtain its π-output. This functionality is stateless but verifies that P and the mediator agree on protocol history with respect to P's view; it also updates P's view. In case of failure a special message $\perp$ outputted. More formally, $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ expect to receive, both from P and the mediator, commitments to all previous messages received by P in π, and to its inputs and randomness. It also receives decommitment information to each of the commitment from the appropriate entity. Then, $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ verifies input correctness and consistency; this verification guarantees that no corrupted party/mediator is able to learn information or bias the output of honest parties. The formal definition of the functionality $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ can be found in Figure 29.

8.4 An ideal functionality for secure function evaluation

Protocol π is UC-secure and therefore it guarantees correctness in its original setting. However, when introducing another entity (i.e., a mediator) the correctness is no longer guaranteed, and it depends on the mediator integrity. When the mediator is honest correctness should be maintained. In the case of corrupted mediator, this is no longer true. A corrupted mediator can prevent honest parties from receive their output (and receive $\perp$ instead). However, the mediator cannot learn any information on honest parties' inputs and outputs or to cause a honest party to output an incorrect value.

To capture these capabilities, we present an ideal functionality which initially receives a bit that indicates the integrity of the mediator and behave according to it. In the honest mediator case, in contrast to the SFE functionality presented in Figure 27, the ideal functionality here, denoted by $\mathcal{F}^{f}_{\mathrm{msfe}}$**,** does not provide the corrupted parties with a leakage regarding the defection of players. Additionally, letting the scheduling be completely controlled by the environment (by introducing the (Compute,...) input) enables $\mathcal{F}^{f}_{\mathrm{msfe}}$ to provide only limited means of synchronization between parties. (See discussion regarding these SFE functionalities in 8.4.) More formally, $\mathcal{F}^{f}_{\mathrm{msfe}}$ receives also (Compute,...) inputs and forward them to the appropriate adversary for approval. $\mathcal{F}^{f}_{\mathrm{msfe}}$ do not allow the adversaries to communicate, and it only notifies the adversaries whenever there are approved (Compute,...) inputs of all parties. Excluding this difference in the adversarial interface, $\mathcal{F}^{f}_{\mathrm{msfe}}$ behaves the same as the standard SFE functionality. In the corrupted mediator case, $\mathcal{F}^{f}_{\mathrm{msfe}}$ behaves similarly to the UC functionality $\mathcal{F}^{f}_{\mathrm{sfe}}$ (presented in [Can01]); however it additionally receives a (Compute,...) input. This ideal functionality is formally presented in Figure 30.

Mediated vs. Physical Solution. The difference between the secure function evaluation functionality $\mathcal{F}^{f}_{\mathrm{msfe}}$ and functionality $\mathcal{F}^{f}_{\mathrm{psfe}}$ that presented in the GMW section is the synchronization provided to parties by the functionality. Recall that $\mathcal{F}^{f}_{\mathrm{psfe}}$ operates in round robin manner, and therefore provides strong synchronization, where $\mathcal{F}^{f}_{\mathrm{msfe}}$ provides only minimal synchronization. This difference follows from $\mathcal{F}^{f}_{\mathrm{psfe}}$ being designed for a specific physical setting as described in Section 7. Therefore, the protocol realizing this functionality can operate only in this setting. In contrast, functionality $\mathcal{F}^{f}_{\mathrm{msfe}}$, and the protocol realizing it can be used in any setting. Moreover, $\mathcal{F}^{f}_{\mathrm{msfe}}$ can be modified to define different clusterings of isolated parties and the realizing protocol will preserve this isolation.

8.5 A compiler for secure multi-party computation

We present a general compiler $\mathrm{MComp}()$ that translates a given protocol π to a more "robust" one that introduces new entity called mediator $\mathcal{M}$. The compiled protocol has a strong isolation guarantee and preserves the original properties even when the mediator is malicious. Informally, the effect of the compiler on the mediator's capabilities is that the mediator must exhibit honest behavior, or else its cheating will be detected. The compiler expects π to be a fully instantiated protocol, namely a protocol that uses

Functionality $\mathcal{F}^{\pi}_{\text{oblComp}}$

Functionality $\mathcal{F}^{\pi}_{\text{OblComp}}$ proceeds as follows, given a protocol π. At the first activation, verify that $sid = (\mathcal{P}, \mathcal{M}; sid')$, where $\mathcal{P}$ is an ordered set of m identities; else halt. Denote the parties identities P_j, $\mathcal{M}$ and the adversaries $\mathcal{S}_{(j,\mathcal{M})}$, $\mathcal{S}_{(\mathcal{M},j)}$. Also, initialize variables OUT_j and $\overrightarrow{\text{msg}}$ to $\perp$. Next:

- P_j inputs a pair of commitments $\text{com}^{\text{input}}$ and com^{rand}; a vector of commitments $\overrightarrow{C}$; and a round number rid. In addition, P_j inputs strings $\text{dec}^{\text{input}}$ and dec^{rand}, and string CRS.
- $\mathcal{M}$ inputs a pair of commitments $\text{com}_j^{\text{input}}$ and $\text{com}_j^{\text{rand}}$; a vector of commitments $\overrightarrow{C_j}$; a commitment $C_j^{\text{rid}'-1}$; and a round number rid'. In addition, $\mathcal{M}$ inputs a vector $\overrightarrow{\text{dec}_j}$, $\text{dec}_j^{\text{rid}'-1}$ and a string CRS'.

1. Upon receiving an input $(\text{Compute} - \text{party}, sid, \text{com}^{\text{input}}, \text{com}^{\text{rand}}, \overrightarrow{C}, \text{rid}, \text{dec}^{\text{input}}, \text{dec}^{\text{rand}}, CRS)$ from party P_j record the input, and send a public delayed output (Receipt_party, sid, P_j) to $\mathcal{M}$. Once it is recorded, ignore any subsequent $(\text{Compute} - \text{party}, ...)$ inputs.
2. Upon receiving an input $(\text{Compute}, sid, \text{com}_j^{\text{input}}, \text{com}_j^{\text{rand}}, \overrightarrow{C_j}, C_j^{\text{rid}'-1}, \text{rid}', \overrightarrow{\text{dec}_j}, \text{dec}_j^{\text{rid}'-1}, CRS')$ from $\mathcal{M}$ and $(\text{Compute} - \text{party}, sid, ...)$ already outputted to $\mathcal{M}$, then record the input. Once it is recorded, ignore any subsequent $(\text{Compute}, ...)$ inputs.
3. Upon receiving `(Corrupt,sid,T)`,where $T \in \{P_j, \mathcal{M}\}$, from an appropriate adversary, do: give both adversaries this party inputs. Furthermore, if this adversary now provides some input value and none of the parties received output, then change the record to this value. In any case, output `(Corrupted,sid,T)` to the newly corrupted party.
4. Upon receiving an input (Deliver, $\mathcal{S}_{(j,i)}$, m) from $\mathcal{S}_{(i,j)}$ do: verify that $\mathcal{S}_{(j,i)}$ and $\mathcal{S}_{(i,j)} \in \{\mathcal{S}_{(j,\mathcal{M})}, \mathcal{S}_{(\mathcal{M},j)}\}$, else ignore the input. output (Deliver, $\mathcal{S}_{(j,i)}$, $\mathcal{S}_{(i,j)}$, m) to $\mathcal{S}_{(j,i)}$.
5. Upon receiving an input `(next_message,sid)` from $\mathcal{S}_{(\mathcal{M},j)}$ and the values above are recorded, run Compute() to obtain $\overrightarrow{\text{msg}}$ and output $(\text{message}, sid, \overrightarrow{\text{msg}})$ to $\mathcal{M}$.
6. Upon receiving an input `(update,sid)` from $\mathcal{S}_{(j,\mathcal{M})}$ and the values above are recorded, run Compute() to obtain OUT_j and output $(\text{update}, sid, C_j^{\text{rid}'-1}, \text{OUT}_j)$ to P_j.

Procedure Compute()

- If $(\text{com}^{\text{input}}, \text{com}^{\text{rand}}, \overrightarrow{C}, \text{rid}, CRS) \neq (\text{com}_j^{\text{input}}, \text{com}_j^{\text{rand}}, \overrightarrow{C_j}, \text{rid}', CRS')$ then set $\overrightarrow{\text{msg}} = \text{OUT}_j = \perp$.
- If $\text{dec}^{\text{input}}\left(\text{resp.}, \text{dec}^{\text{rand}}\right)$ is not a valid decommitment to $\text{com}_j^{\text{input}}\left(\text{resp.}, \text{com}_j^{\text{rand}}\right)$, or $\overrightarrow{\text{dec}_j}$ does not contain valid decommitments to all the commitments in $\overrightarrow{C_j}$ then set $\overrightarrow{\text{msg}} = \text{OUT}_j = \perp$.
- If $\text{rid}' > 1$ and $\text{dec}_j^{\text{rid}'-1}$ is not a valid decommitment to $C_j^{\text{rid}'-1}$ then set $\overrightarrow{\text{msg}} = \text{OUT}_j = \perp$.
- Let $\overrightarrow{\text{msg}_1}, \ldots, \overrightarrow{\text{msg}_{\text{rid}'-1}}$ be the committed values in $\overrightarrow{C_j}$ and $C_j^{\text{rid}'-1}$. If any of these contain dummy values, set $\overrightarrow{\text{msg}} = \text{OUT}_j = \perp$.
- Let x_j and r_j be the committed values in $\text{com}_j^{\text{input}}$and $\text{com}_j^{\text{rand}}$ respectively and $\overrightarrow{\text{msg}} = \text{OUT}_j = \perp$. if $\text{rid}' < r + 1$ then compute and store in $\overrightarrow{\text{msg}}$ the next messages that party P_j would send in protocol π when running with input x_j, random tape r_j, common reference string CRS, and after receiving messages $\overrightarrow{\text{msg}_1}, \ldots, \overrightarrow{\text{msg}_{\text{rid}'-1}}$. else compute and store in OUT_j the output of party P_j.
- Return $\left\langle \overrightarrow{\text{msg}}, \text{OUT}_j \right\rangle$.

Fig. 29: The functionality computing the next round messages of party P_j in π

Functionality $\mathcal{F}_{\text{msfe}}^{f}$

Functionality $\mathcal{F}_{\text{msfe}}^{f}$ proceeds as follows, given a function $f : (\{0,1\}^* \cup \{\perp\})^m \times R \to (\{0,1\}^*)^m$. At the first activation, verify that $sid = (\mathcal{P}; sid')$, where $\mathcal{P}$ is an ordered set of m identities; else halt. Denote the parties identities $P_1, ..., P_m$, adversaries identities $\{\mathcal{S}_{(i,j)}\}_{i \neq j, i \in \mathcal{P} \bigcup \{\mathcal{M}\}, j \in \mathcal{P} \bigcup \{\mathcal{M}\}}$. Also, initialize variables $x_1, ..., x_m$, $y_1, ..., y_m$ to default value $\perp$. Next:

– If $mediator = 1$ (this indicates a honest mediator) then $\mathcal{F}_{\text{msfe}}^{f}$ proceeds as follows:

1. INPUT&COMPUTE: Upon receiving an input message from party P_i do:
   (a) if input is (Input,sid,v) then set $x_i = v$ and output (Input,sid,P_i) to $\mathcal{S}_{(i,\mathcal{M})}$.
   (b) if input is (Compute,sid) then notify the adversaries $\mathcal{S}_{(i,\mathcal{M})}$, $\mathcal{S}_{(\mathcal{M},i)}$ and record the (Compute,...) request.
2. INPUT RESPONSE: Upon receiving an `okay` message from all $\mathcal{M}$ related adversaries and all inputs are set then output a `continue` message to all $\mathcal{M}$ related adversaries.
3. COMPUTE RESPONSE: Upon receiving input `(response, sid)` from $\mathcal{S}_{(\mathcal{M},i)}$ do:
   (a) mark the appropriate (Compute,sid) request as approved in current round (if P_i is corrupted mark as if recorded).
   (b) if all parties in $\mathcal{P}$ have at least one (Compute,...) approved in current round then output a `continue` message to all $\mathcal{M}$ related adversaries and delete all approved requests.
4. OUTPUT: Upon receiving an `(output, P_i, p)` message from $\mathcal{S}_{(i,\mathcal{M})}$ do:
   (a) if x_i has been set for all parties in $\mathcal{P}$, and $y_1, ..., y_n$ have not yet been set, then choose $r \leftarrow R$ and set $(y_1, .., y_n) \leftarrow f(x_1, ..., x_n, r)$; else if x_i has not been set for all parties, ignore.
   (b) if P_i is corrupted then output y_i to $\mathcal{S}_{(i,\mathcal{M})}$; else output y_i to P_i.
5. CORRUPT: Upon receiving input (Corrupt, P_i, p) from P_i related adversary do:
   (a) record P_i as corrupted party and output (Corrupted, P_i) to P_i.
   (b) give x_i and an output that was given to P_i (if any) to all P_i related adversaries.
6. DELIVER: Upon receive input (Deliver, $\mathcal{S}_{(\ell,k)}$, m) from $\mathcal{S}_{(i,j)}$ do:
   (a) if $i = \ell$ and P_i is corrupted or $j = \ell = \mathcal{M}$ and $k = j$ then output (Deliver, $\mathcal{S}_{(i,j)}$, $\mathcal{S}_{(\ell,k)}$, m) to $\mathcal{S}_{(\ell,k)}$.
7. SET INPUT: Upon receiving input `(set_input, P_i, v')` from $\mathcal{S}_{(i,\mathcal{M})}$ do:
   (a) if P_i is corrupted and none of the parties received output then set $x_i \leftarrow v'$; otherwise ignore.
8. ABORT: Upon receiving input `(abort, P_i)` from $\mathcal{S}_{(i,\mathcal{M})}$ do:
   (a) if P_i is corrupted and none of the parties received output then set $(y_1, .., y_n) \leftarrow (\perp, ..., \perp)$; otherwise ignore.

– If $mediator = 0$ (this indicates a corrupted mediator) then $\mathcal{F}_{\text{msfe}}^{f}$ proceeds as follows:

1. INPUT&COMPUTE: Upon receiving an input message from party P_i do as in the honest mediator case but notify all adversaries.
2. OUTPUT: Upon receiving an `(output, P_i, p)` message from $\mathcal{S}_{(i,\mathcal{M})}$ do:
   (a) if x_i has been set for all parties $P_i \in \mathcal{P}$, and $y_1, ..., y_n$ have not yet been set, then choose $r \leftarrow R$ and set $(y_1, .., y_n) \leftarrow f(x_1, ..., x_n, r)$; else if x_i has not been set for all parties, ignore.
   (b) if P_i is corrupted then output y_i to all adversaries.
   (c) else: if $p = \perp$ then set $y_i = \perp$. In any case, output y_i to P_i.
3. CORRUPT: Upon receiving input (Corrupt, P_i, p) from P_i related adversary do as for the honest mediator case but notify all adversaries.
4. DELIVER: Upon receive input (Deliver, $\mathcal{S}_{(\ell,k)}$, m) from $\mathcal{S}_{(i,j)}$ then output (Deliver, $\mathcal{S}_{(i,j)}$, $\mathcal{S}_{(\ell,k)}$, m) to $\mathcal{S}_{(\ell,k)}$ (this capture the ability of corrupted parties to communicate using corrupted mediator)
5. SET INPUT: Upon receiving input `(set_input, P_i, v')` from $\mathcal{S}_{(i,\mathcal{M})}$ do as in the honest mediator case.

Fig. 30: The Secure Function Evaluation functionality for evaluating an n-party function f

no ideal functionalities other than $\bar{\mathcal{G}}_{\overline{\text{acrs}}}$. As discussed above, the constructed protocol $\text{MComp}(\pi)$ is presented as an $(\mathcal{F}_{\overline{\text{smt}}}, \mathcal{F}^{\pi}_{\text{OblComp}}, \bar{\mathcal{G}}_{\overline{\text{acrs}}})$-hybrid protocol. Next, when these functionalities are realized we obtain a protocol in the $\bar{\mathcal{G}}_{\overline{\text{acrs}}}$-hybrid model.

The protocol consists of three stages. In the first stage the parties and the mediator communicate with $\bar{\mathcal{G}}_{\overline{\text{acrs}}}$ and obtain the CRS string. In the second stage, the parties commit to their inputs and random coins for a protocol π that securely computes $\mathcal{F}$. In the third stage, the parties emulate π, round-by-round, as follows. Upon party P_j requests, $\mathcal{M}$ engages in P_j's next π-message computation. at the end of this computation, $\mathcal{M}$ obtains the messages that P_j would send in the current round of π, and P_j obtains a commitment to the messages it would receive in the previous round of π. In last round emulation P_j also obtains its output. However, the mediator would not engage in a computation with the same party more than once in a given round, but rather records the request and process it in the next round of the protocol. Whenever $\mathcal{M}$ collects all current round messages it proceeds to the emulation of the next round. Everything the mediator sends to the parties will be "wrapped" inside a commitment. When all parties behave honestly, these will all be commitments to legitimate protocol messages of π. If some party P_j aborts (or deviates from the protocol), $\mathcal{M}$ will not be able to generate a valid commitment of this sort. Nevertheless, we do not want other party to learn that P_j aborted the protocol; Therefore, we allow $\mathcal{M}$ to send special "dummy commitments" to $\perp$. It is important to note that no corrupted party can collude with the corrupted mediator to perform a malleability attack on the protocol. More formally, the input commitments received by the corrupted mediator do not enable to compute the protocol on a related value. This follows from the ideal $\mathcal{F}^{\pi}_{\text{OblComp}}$ (that do not reveal any information on the committed values to the mediator) and from the UC security of the protocol realizing $\mathcal{F}^{\pi}_{\text{OblComp}}$. Therefore, a standard statistically binding non-interactive commitment scheme suffices. Let C be such a commitment scheme, where $C(m; r)$ denotes a commitment to m using random coins r. The decommitment of $\text{com} = C(m; r)$ is $\text{dec} = (m; r)$. Recall that that the interface of $\mathcal{F}^{\pi}_{\text{OblComp}}$ involves the commitment scheme C. Formal description of the protocol appears in Figures 31 and 32.

We are now ready to prove the security of the compiled protocol $\text{MComp}(\pi)$:

Theorem 11. *Given a (poly-time) function $f = (f_1, ..., f_n)$ and a protocol π that is $\bar{\mathcal{G}}_{\text{acrs}}$-setup oblivious UC-realization of the well-formed aborting functionality $\mathcal{F}^{f}_{\text{sfe}}$. Then the compiled protocol $\text{MComp}(\pi)$ LUC-realize $\mathcal{F}^{f}_{\text{msfe}}$ in the $(\mathcal{F}_{\overline{\text{smt}}}, \mathcal{F}^{\pi}_{\text{OblComp}}, \bar{\mathcal{G}}_{\overline{\text{acrs}}})$-hybrid model in the presence of static adversaries, where all corruptions are PID-wise.*

8.6 Proof of security

Let $f = (f_1, ..., f_n)$ be a (poly-time) function and let π be a r-round as in 8.2.2 that UC-realize $\mathcal{F}^{f}_{\text{sfe}}$ in the $\bar{\mathcal{G}}_{\text{acrs}}$-hybrid model. Let $\text{MComp}(\pi)$ be the compiled protocol in the $(\mathcal{F}_{\overline{\text{smt}}}, \mathcal{F}^{\pi}_{\text{OblComp}}, \bar{\mathcal{G}}_{\overline{\text{acrs}}})$-hybrid model, and let $\mathcal{A}$ be an adversary interacting with $\text{MComp}(\pi)$. We construct a simulator $\mathcal{S}$ that interacts with $\mathcal{F}^{f}_{\text{msfe}}$ in the ideal model, such that no environment $\mathcal{Z}$ can tell whether it is interacting with $\text{MComp}(\pi)$ and $\mathcal{A}$ or with $\mathcal{F}^{f}_{\text{msfe}}$ and $\mathcal{S}$. As usual, we restrict attention to the case where $\mathcal{A}$ is the dummy adversary. We prove LUC realization for the honest and malicious mediator scenario separately. For convenience, we denote a simulator with identity $((i, j), \perp)$ as $\mathcal{S}_{(i,j)}$.

8.6.1 Honest mediator

Recall that the messages in $\text{MComp}(\pi)$ are adversely scheduled. Therefore, the simulator is not only in charge of simulating the protocol messages but also simulate the scheduling done in the execution of $\text{MComp}(\pi)$. The idea of simulating the adversarial scheduling is that each pair of simulator can jointly simulate the scheduling of each round of the protocol using the DELIVER service of $\mathcal{F}^{f}_{\text{msfe}}$. When the mediator is honest our goal is to simulate the view of corrupted parties.

Protocl $\mathrm{MComp}(\pi)$**: Party** P_j

- INITIALIZATION: P_j is initialized with the common reference string CRS produced by $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$.
- INPUT: Having received input (Input,sid,x_j) do:
  1. P_j chooses random s_j and computes $\mathrm{com}^{\mathrm{input}} = C(x_j, s_j)$. We denote by $\mathrm{dec}^{\mathrm{input}}$ be the decommitment owned by P_j.
  2. let ℓ is the number of coins needed to run π. P_j chooses random r'_j, s'_j, where $\left|r'_j\right| = \ell$ and computes $\mathrm{com}^{\mathrm{rand}} = C\left(r'_j, s'_j\right)$. Let $\mathrm{dec}^{\mathrm{rand}} = \left(r'_j, s'_j\right)$ the decommitment owned by P_j. Also, initialize $i = 1$.
  3. P_j provides $(\mathrm{Send},\ sid,\ \mathrm{com}^{\mathrm{input}}, \mathrm{com}^{\mathrm{rand}})$ to $\mathcal{F}_{\overline{\mathrm{smt}}}$.
- COMPUTE: Upon P_j receives $(\mathrm{Compute},\ sid)$ input, where P_j finished the INPUT phase, and there is no open (Compute,...) session with $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ do:
  1. let $\overrightarrow{C} = \left(\mathrm{com}^j_1, \ldots, \mathrm{com}^j_{i-2}\right)$ be the commitments that P_j received in the previous $i-2$ rounds. P_j sends $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ its commitments $\mathrm{com}^{\mathrm{input}}$ and $\mathrm{com}^{\mathrm{rand}}$, the vector of commitments $\overrightarrow{C}$, the round identifier rid=0i , the decommitments $\mathrm{dec}^{\mathrm{input}}$ and $\mathrm{dec}^{\mathrm{rand}}$. Additionally, it sends CRS.
- UPDATE: Having received output $(\mathrm{update}, sid, \mathrm{com}^j_{i-1}, \mathrm{OUT}_j)$ from $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ do:
  1. if $i > 1$ then update $\overrightarrow{C}$ to include com^j_{i-1}.
  2. if $i < r + 1$ then increment $i = i + 1$; else output OUT_j and halt.

Fig. 31: The general function evaluation protocol for party P_j

Simulation of the interaction with the setup. The simulator behaves identically to the dummy adversary and records the obtained CRS. That is, any instruction to interact with $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$ is honestly executed and any output received from $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$ is forwarded to $\mathcal{Z}$.

Simulation of honest party. Let j denote the party identity of party P_j. Then the simulator $\mathcal{S}_{(j,\mathcal{M})}$ proceeds as follows:

1. Upon output (Input,sid,P_j) from $\mathcal{F}^f_{\mathrm{msfe}}$ and no (Input,sid',P_j) previously received, the simulator $\mathcal{S}_{(j,\mathcal{M})}$ first initialize $rid = 1$, OUT=$\perp$ and ABORT=false. Next, record (Input,sid,P_j) and output $(\mathrm{OUTPUT}, \mathcal{M}, sid)$ to $\mathcal{Z}$.
2. Upon output (Compute,sid,P_j) from $\mathcal{F}^f_{\mathrm{msfe}}$ and (Input,sid',P_j) previously received, do the following:
   (a) if it is the first (Compute,...) message in round rid then $\mathcal{S}_{(j,\mathcal{M})}$ record (Compute,sid,P_j) and outputs $(\mathrm{OUTPUT}, \mathcal{M}, sid)$ to $\mathcal{Z}$.
3. Upon input $(\mathrm{APPROVE}, \mathcal{M}, sid)$ from $\mathcal{Z}$ proceed as follows:
   (a) if (Input,sid,P_j) recorded, then erase (Input,sid,P_j) and send $(\mathrm{APPROVE} - \mathrm{input}, \mathcal{M}, sid)$ to $\mathcal{S}_{(\mathcal{M},j)}$ (by giving a (DELIVER,$\mathcal{S}_{(\mathcal{M},j)}$,...) input to $\mathcal{F}^f_{\mathrm{msfe}}$).
   (b) if (Compute,sid,P_j) recorded, then erase (Compute,sid,P_j) and send $(\mathrm{APPROVE} - \mathrm{compute}, \mathcal{M}, sid)$ to $\mathcal{S}_{(\mathcal{M},j)}$
4. Upon message (Compute $-$ Med,sid) from $\mathcal{S}_{(\mathcal{M},j)}$ then $\mathcal{S}_{(j,\mathcal{M})}$ records this tuple.
5. Upon input `(update,sid)` from $\mathcal{Z}$ and (Compute $-$ Med,sid) recorded, then $\mathcal{S}_{(j,\mathcal{M})}$ does the following:
   (a) if $rid = r+1$ then sent `(output,`P_j`,p)` to $\mathcal{F}^f_{\mathrm{msfe}}$ and halt; otherwise erase (Compute $-$ Med,sid) and set $rid = rid + 1$.

Protocl $\mathrm{MComp}(\pi)$**: Mediator** $\mathcal{M}$

- INITIALIZATION: $\mathcal{M}$ is initialized with the common reference string CRS produced by $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$.
- INPUT: Having received commitments ($\mathrm{com}_j^{\mathrm{input}}$, $\mathrm{com}_j^{\mathrm{rand}}$) from $\mathcal{F}_{\overline{\mathrm{smt}}}$ for all $j \in \mathcal{P}$, initialize $i = 1$, ABT=false, and a round i message matrix $D^i_{|\mathcal{P}|\times|\mathcal{P}|-1}$ with dummy messages. In addition, for each recorded open session do as in COMPUTE REQUEST and not yet engaged with P_j in current round.
- COMPUTE REQUEST: Having received output (Receipt_party, sid, P_j) from $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ do:
  - if have not yet engaged with P_j in current round and i initialized computation then provides $\mathrm{com}_j^{\mathrm{input}}$ and $\mathrm{com}_j^{\mathrm{rand}}$, the commitments $\left(\mathrm{com}_1^j, \ldots, \mathrm{com}_{i-2}^j\right)$, the commitment com_{i-1}^j to round $i-1$ messages, the round identifier rid'=0i, the decommitments $\left(\mathrm{dec}_1^j, \ldots, \mathrm{dec}_{i-2}^j\right)$, and the decommitment dec_{i-1}^j. Additionally, it sends CRS. The commitment com_0^j is a commitment to zero for all $j \in \mathcal{P}$.
  - else if no open session recorded for P_j then record (Receipt_party, sid, P_j) as open session. Additionally, if $i = r + 1$ and engaged with all parties in $\mathcal{P}$ in current round computation then halt.
- UPDATE: Having received output $(\mathrm{message}, sid, \overrightarrow{\mathrm{msg}})$ from $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ where $sid = (P_j, \mathcal{M}; sid')$, and $i < r + 1$ do:
  1. if $\overrightarrow{\mathrm{msg}} = \perp$ then set ABT=true; else, update D^i to include $\overrightarrow{\mathrm{msg}}$.
  2. if received $\overrightarrow{\mathrm{msg}}$ for all $j \in \mathcal{P}$ in round i then:
     (a) if ABT=true then rewrite D^i with dummy messages.
     (b) for all $k \in \mathcal{P}$ choose random r_k and compute a commitment $\mathrm{com}_i^k = C\left(\left(D^i\right)_k, r_k\right)$ to the messages that P_k would receive in round i. Let $\mathrm{dec}_i^k = \left(\left(D^i\right)_k, r_k\right)$ be the decommitment owned by $\mathcal{M}$.
     (c) increment $i = i+1$ and initialize round i message matrix $D^i_{|\mathcal{P}|\times|\mathcal{P}|-1}$ with dummy messages.
     (d) for each recorded open session do as in COMPUTE REQUEST and not yet engaged with P_j in current round.

Fig. 32: The general function evaluation protocol for mediator $\mathcal{M}$

Simulation of corrupted party. Let j denote the party identity of party P_j. Then the simulator $\mathcal{S}_{(j,\mathcal{M})}$ proceeds as follows:

1. Upon input $(\text{Send}, sid, \text{com}^{\text{input}}, \text{com}^{\text{rand}})$ from $\mathcal{Z}$ do the following:
   (a) if this is the first $(\text{Send}, ...)$ input then record $\left(sid, \text{com}^{\text{input}}, \text{com}^{\text{rand}}\right)$ and initialize $rid = 1$, OUT=$\perp$ and ABORT=false and go to (d).
   (b) else if there exists a recorded commitment associated with sid and $\mathcal{S}_{(\mathcal{M},j)}$ yet send `okay` to $\mathcal{F}_{\text{msfe}}^{f}$ then change record to the received commitments.
   (c) else if sid was never used, record $\left(sid, \text{com}^{\text{input}}, \text{com}^{\text{rand}}\right)$ and go to (d).
   (d) Records (Input,sid,P_j) and outputs $(\text{OUTPUT}, \mathcal{M}, sid)$ to $\mathcal{Z}$.
2. Upon input $(\text{Compute} - \text{party}, sid, \overline{\text{com}}^{\text{input}}, \overline{\text{com}}^{\text{rand}}, \overrightarrow{C}', \text{rid}', \overline{\text{dec}}^{\text{input}}, \overline{\text{dec}}^{\text{rand}}, CRS')$ from $\mathcal{Z}$ do the following:
   (a) if $rid \leq r + 1$then:
      i. if sid was never used, then record this tuple.
      ii. Else if $(\text{Compute} - \text{party}, sid, ...)$ recorded and $\mathcal{S}_{(\mathcal{M},j)}$ yet send `(response,sid)` then update the existing record. Otherwise, ignore input and skip (b).
   (b) record (Compute,sid,P_j) and output $(\text{OUTPUT}, \mathcal{M}, sid)$ to $\mathcal{Z}$.
3. Upon input $(\text{APPROVE}, \mathcal{M}, sid)$ from $\mathcal{Z}$ then, do as in the honest party simulation.
4. Upon message (Committed,sid,P_j) from $\mathcal{S}_{(\mathcal{M},j)}$, then set $\left(sid, \text{com}^{\text{input}}, \text{com}^{\text{rand}}\right)$ as the commitments given by P_j to $\mathcal{M}$.
5. Upon message (Compute $-$ Med,sid) from $\mathcal{S}_{(\mathcal{M},j)}$, then record it.
6. Upon input `(update,sid)` from $\mathcal{Z}$ and (Compute $-$ Med,sid) recorded, then $\mathcal{S}_{(j,\mathcal{M})}$ does the following:
   (a) compute $\text{com}_{rid-1} = C(0^n)$, and erase (Compute $-$ Med,sid).
   (b) checked correctness. That is, Let $(\text{Compute} - \text{party}, sid, \overline{\text{com}}^{\text{input}}, \overline{\text{com}}^{\text{rand}}, \overrightarrow{C}', \text{rid}', \overline{\text{dec}}^{\text{input}}, \overline{\text{dec}}^{\text{rand}}, CRS')$ be the recorded tuple.
      – If rid=1 then set $\left(\text{dec}^{\text{input}}, \text{dec}^{\text{rand}}\right) = \left(\overline{\text{dec}}^{\text{input}}, \overline{\text{dec}}^{\text{rand}}\right)$ and send `(set_input,`P_i`,`x'`)` to $\mathcal{F}_{\text{msfe}}^{f}$ where x' is the value in $\overline{\text{dec}}^{\text{input}}$.
      – If $(\text{com}^{\text{input}}, \text{com}^{\text{rand}}, \overrightarrow{C}, \text{rid}, CRS, \text{dec}^{\text{input}}, \text{dec}^{\text{rand}})$ $\neq (\overline{\text{com}}^{\text{input}}, \overline{\text{com}}^{\text{rand}}, \overrightarrow{C}', \text{rid}', CRS', \overline{\text{dec}}^{\text{input}}, \overline{\text{dec}}^{\text{rand}})$ then set ABORT=true.
      – if $\overline{\text{dec}}^{\text{input}}\left(\text{resp.}, \overline{\text{dec}}^{\text{rand}}\right)$ is not a valid decommitment to $\text{com}^{\text{input}}\left(\text{resp.}, \text{com}^{\text{rand}}\right)$, then set ABORT=true.
      – Let $\overrightarrow{\text{msg}_1}, \ldots, \overrightarrow{\text{msg}_{\text{rid}'-1}}$ be the committed values in $\overrightarrow{C_j}$ and $C_j^{\text{rid}'-1}$. If any of these contain dummy values, set ABORT=true.
      – if $rid < r + 1$ and ABORT set to true then send `(abort,`P_j`)` to $\mathcal{F}_{\text{msfe}}^{f}$.
   (c) if $rid = r + 1$ and ABORT=false then sent `(output,`P_j`,`p`)` to $\mathcal{F}_{\text{msfe}}^{f}$ and set OUT to be the output of P_j given by $\mathcal{F}_{\text{msfe}}^{f}$; else if $rid < r + 1$ then set $rid = rid + 1$.
   (d) Update the commitments vector $\overrightarrow{C}$ to contain com_{rid-1} and output $(\text{update}, sid, \text{com}_{rid-1}, \text{OUT})$ to $\mathcal{Z}$.

Simulation of the honest mediator. The simulator $\mathcal{S}_{(\mathcal{M},j)}$ with identity $(\perp, (\mathcal{M}, j))$ initialize $rid^* = 0$ and $b = 1$. Next, it proceeds as follows:

1. Upon message $(\text{APPROVE} - \text{input}, \mathcal{M}, sid)$ from $\mathcal{S}_{(j,\mathcal{M})}$, then $\mathcal{S}_{(\mathcal{M},j)}$ records (Input,sid,P_j) and outputs $(\text{OUTPUT}, \mathcal{M}, sid)$ to $\mathcal{Z}$.
2. Upon message $(\text{APPROVE} - \text{compute}, \mathcal{M}, sid)$ from $\mathcal{S}_{(j,\mathcal{M})}$, then $\mathcal{S}_{(\mathcal{M},j)}$ records (Compute,sid,P_j) and outputs $(\text{OUTPUT}, \mathcal{M}, sid)$ to $\mathcal{Z}$.

3. Upon input $(\text{APPROVE}, \mathcal{M}, sid)$ received from $\mathcal{Z}$, $\mathcal{S}_{(\mathcal{M},j)}$ proceeds as follows:
   (a) if (Input,sid,P_j) recorded do:
      i. if this is the first input approved then send (Committed,sid,P_j) to $\mathcal{S}_{(j,\mathcal{M})}$ and `okay` to $\mathcal{F}_{\text{msfe}}^{f}$; otherwise, erase this record.
   (b) if (Compute,sid,P_j) recorded do:
      i. erase (Compute,sid,P_j).
      ii. if $b = 0$ then record (Compute,sid,rid^*,b) and sent (Compute − Med,sid) to $\mathcal{S}_{(j,\mathcal{M})}$.
      iii. if $b = 1$ and there is no record with $b = 1$ and $k = rid^*$ then record (Compute,sid,rid^*,b)
4. Upon input `(next_message,sid)` from $\mathcal{Z}$ and (Compute,sid,k,b) recorded then:
   (a) if $rid^* < r + 1$ and ($b = 0$ or $b = 1 \wedge rid^* > k$) then do:
      i. erase (Compute,sid,k,b) and instruct $\mathcal{S}_{(j,\mathcal{M})}$ to abort if the simulation of $\mathcal{F}_{\text{OblComp}}^{\pi}$ with session id sid results in ABORT=true .
      ii. send `(response,sid)` to $\mathcal{F}_{\text{msfe}}^{f}$.
   (b) otherwise, ignore this input.
5. Upon `continue` output from $\mathcal{F}_{\text{msfe}}^{f}$, the simulator sets $rid^* = rid^* + 1$. In addition, if a record with $b = 1$ exists then set $b = 1$ and send (Compute − Med,sid) to $\mathcal{S}_{(j,\mathcal{M})}$; otherwise, set $b = 0$.

In order to show that the environment's output in the real-life model is indistinguishable from its output in the ideal-process, we consider the following hybrids:

Ideal/Fake: The output of $\mathcal{Z}$ in an execution in the ideal process with $\mathcal{S}$ and $\mathcal{F}_{\text{msfe}}^{f}$ (with commitments to zero generated by $\mathcal{S}$).

Ideal/Genuine(i,k): The output of $\mathcal{Z}$ from the following interaction. Each simulator of a party, in each round, is given the real messages. That is, the messages that this party would receive commitments to them from the honest mediator in $\text{MComp}(\pi)$. If a simulator with identity $((j, \mathcal{M}), \perp)$ is supposed to generate commitment to a messages received by a corrupted party in a round smaller then i or if the current round equal i and $j < k$, then we let it to compute the commitment by committing to the real messages instead of a zero string. Note that Ideal/Genuine(1,1) is exactly the Ideal/Fake hybrid.

Real/Genuine: The output of $\mathcal{Z}$ in a real-life execution with parties running the protocol $\text{MComp}(\pi)$, honest mediator, and dummy adversary $\mathcal{D}$. This amounts to retrieving the global CRS and then running the protocol in the real-life model with $\mathcal{D}$ and $\mathcal{Z}$.

Indistinguishability of Ideal/Fake and Ideal/Genuine(r+2,1). Let us presume, for sake of contradiction, that $\mathcal{Z}$ tells apart the hybrids Ideal/fake and Ideal/Genuine(r+2,1) with non-negligible probability. From a standard hybrid argument follows that there exists (i,k), for some $1 \leq i \leq r + 1$ and $1 \leq k \leq m$ where $k + 1$ is an identity of a corrupted party, and an environment $\mathcal{Z}^{(i,k)}$ that can distinguish between Ideal/Genuine(i,k) and Ideal/Genuine(i,k+1). (In case $k = m$ we consider the hybrid Ideal/Genuine(i+1,1)).

We note that the only difference between Ideal/Genuine(i,k) and Ideal/Genuine(i,k+1) is the commitment to all zero string given by P_{k+1} in Ideal/Genuine(i,k). From this, we construct a receiver R^* that breaks the hiding property of the commitment. Details follow.

By definition, for a computationally hiding commitment scheme $C()$ the prediction probability of any polynomially-bounded receiver R^* should not exceed $\frac{1}{2}$ by a non-negligible amount. Given environment $\mathcal{Z}^{(i,k)}$ that distinguishes between Ideal/Genuine(i,k) and Ideal/Genuine(i,k+1), we construct a successful receiver for the hiding property of the commitment scheme C.

Let m_0 be the all zero string and m_1 be a vector of messages in the commitment given by $\mathcal{F}_{\text{OblComp}}^{\pi}$ to party P_{k+1} in round i. The receiver R^* gets 1^n as input. Let b be the random bit that determines if the committer commits to m_0 ($b = 0$) or m_1 ($b = 1$) messages. R^* tries to predict b by simulating a Ideal/Genuine(i,k+1) execution:

– R^* run internally $\mathcal{Z}^{(i,k)}$ and simulate an execution of Ideal/Genuine(i,k+1) with the following difference: in round i, instead of giving $\mathcal{Z}^{(i,k)}$ a commitment to all zero string on behalf of P_{k+1}, the receiver is giving the commitment it received from the committer S.

– at the end, R^* outputs whatever $\mathcal{Z}^{(i,k)}$ outputs.

If S commits to m_0 then R^* simulates an Ideal/Genuine(i,k) execution. We conclude that the probability that R^* outputs 1 in this case equals the probability that $\mathcal{Z}^{(i,k)}$ returns 1 in experiment Ideal/Genuine(i,k). Also, if the committer commits to m_1 then R^* simulates the experiment Ideal/Genuine(i,k+1) and outputs 1 exactly if $\mathcal{Z}^{(i,k)}$ gives output 1 in this experiment. Hence,

$$\begin{aligned}
&\mathrm{Prob}[R^* \; outputs \; b] \\
&= \mathrm{Prob}[b = 1 \wedge \mathcal{Z}^{(i,k)} \; outputs \; 1] + \mathrm{Prob}[b = 0 \wedge \mathcal{Z}^{(i,k)} \; outputs \; 0] \\
&= \frac{1}{2} \cdot \mathrm{Prob}[\mathcal{Z}^{(i,k)} \; outputs \; 1 \; in \; Ideal/Genuine(i, k+1)] \\
&\quad + \frac{1}{2} \cdot \mathrm{Prob}[\mathcal{Z}^{(i,k)} \; outputs \; 0 \; in \; Ideal/Genuine(i, k)] \\
&= \frac{1}{2} + \frac{1}{2} \cdot (\mathrm{Prob}[\mathcal{Z}^{(i,k)} \; outputs \; 1 \; in \; Ideal/Genuine(i, k+1)] \\
&\quad - \mathrm{Prob}[\mathcal{Z}^{(i,k)} \; outputs \; 1 \; in \; Ideal/Genuine(i, k)])
\end{aligned}$$

R^*'s prediction probability is therefore bounded away from $\frac{1}{2}$ by a non-negligible function, contradicting the hiding property of the commitment scheme C.

Indistinguishability of Ideal/Genuine(r+2,1) and Real/Genuine. In order to show that the environment's output in the real-life model is indistinguishable from its output in Ideal/Genuine(r+2,1), we first need to show that the scheduling observed by the environment in the real execution of $\mathrm{MComp}(\pi)$ is identical to the scheduling in Ideal/Genuine(r+2,1). By inspecting the code of the simulator and the protocol we observe the following:

– The delayed output simulation is done identically to the dummy adversary in the real execution of $\mathrm{MComp}(\pi)$. Moreover, the simulator engage in a delayed output simulation only if the rid, as recorded by the simulator, indicates that the party invoked an instance of $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ or $\mathcal{F}_{\overline{\mathrm{smt}}}$ in the real execution.
– The parties generate their output when the round id rid, as maintained by the party simulator, reaches $r + 1$.
– Commitments to messages received by a corrupted party are generated only if the rid of the corresponding mediator's simulator indicates that the honest mediator in the real execution submitted inputs an instance of $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ with a suitable sid.

Based on the observations above, it suffices to prove that at each point during the ideal execution the round id maintained by the simulators is identical to the round id of the participants in $\mathrm{MComp}(\pi)$. We start by proving this for the simulators associated with the mediator (i.e., the simulators with identity $((\mathcal{M}, j), \perp)$).

At first activation, the simulator $\mathcal{S}_{(\mathcal{M},j)}$ set the round id rid^* to zero. The rid^* advances upon the `continue` output received from $\mathcal{F}^{f}_{\mathrm{msfe}}$. The rid^* advances for the first time only after all simulators associated with the mediator announce the reception of the input-randomness commitment from the party associated with the identity of the simulator. Since the simulator behaves as a dummy adversary during the delayed output simulation, its reception announcement happens exactly when the mediator receive an input-randomness commitment from this party in the execution $\mathrm{MComp}(\pi)$. Therefore, when rid^* is set to '1' in the simulated execution it is also set to '1' by the mediator in the execution of $\mathrm{MComp}(\pi)$.In the rest of the simulation a `continue` output received only when all the simulators associated with the mediator send `(response,sid)` to $\mathcal{F}^{f}_{\mathrm{msfe}}$. A simulator sends `(response,sid)` to the functionality exactly when the environment instructs the adversary to let $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ to give output to the mediator. Therefore, when the rid^* is advanced in the simulated execution it is also advanced by

the honest mediator in the real execution of $\mathrm{MComp}(\pi)$. In addition, a `(response,sid)` message is never send after rid^* reaches $r+1$, which is the last round in $\mathrm{MComp}(\pi)$. This implies that the round id rid^* , as maintained by the simulators of the mediator, is identical to the round id of the honest mediator in the real execution.

Next, we prove that the round id rid, as maintained by the simulator of the parties (i.e., the simulators with identity $((j, \mathcal{M}), \perp)$) is identical to the round id of the corresponding party in the real execution. Since a corrupted party only execution the instructions received from the environment, we only need to show this with respect to the round id of honest parties.

We note that rid is set to '1' when the simulator receive the first (Input,sid,P_j) message. In the real execution, when a party receives the first (Input,sid,P_j) message, it set $rid = 1$ (and invoke an instance of $\mathcal{F}_{\overline{\text{smt}}}$); therefore the rid is identical during the first round of the protocol. Moreover, A simulator advance rid exactly when the environment instructs the adversary to let $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ to give output to this party (and the rid^* of the corresponding mediator's simulator indicates the mediator submitted its input to this instance of $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$). Therefore, based on the correctness of rid^*, when the rid is advanced in the simulated execution it is also advanced by the honest party in the real execution of $\mathrm{MComp}(\pi)$.

We note that up to round $r+1$ the produced transcript in both hybrids is indistinguishable. In order to complete this proof we need to show indistiguishability of the outputs of the honest parties. We consider the following possible events:

No deviations. In this case in the real-life execution of $\mathrm{MComp}(\pi)$ all parties submit to $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ inputs that match the input submitted by the honest mediator, and therefore, generate outputs according to the underlying protocol π. Moreover, the corrupted parties inputs and randomness in the execution of π is uniquely determined by the statistically binding commitment given in the INPUT phase of $\mathrm{MComp}(\pi)$. In the execution of Ideal/Genuine(r+2,1) the simulators associate with a corrupted party set the parties inputs (by sending it to $\mathcal{F}^{f}_{\mathrm{msfe}}$) according to the decoomitment received from the environment in the first round simulation (this input cannot later change due to the binding property of $C()$). Since the corrupted parties inputs to $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ are according to honest execution of $\mathrm{MComp}(\pi)$, then ABORT is never set to true and the output is generated according to $f(x_1, ..., x_n, r)$. Based on the correctness of π, these outputs are indistinguishable from the outputs in $\mathrm{MComp}(\pi)$.

At least one deviation. Since deviations in the last round of $\mathrm{MComp}(\pi)$ effect only the deviating parties output, which is $\perp$ in both executions, we concentrate on the case where the deviations occur in round $i < r+1$. Although up to the output generation the transcript is not influenced by the deviations (since in both executions the environment is exposed only to commitments, with an identical underlying values), it is crucial for a deviation to identically effect the output in both execution. The only feasible deviation for a corrupted party participating in $\mathrm{MComp}(\pi)$ is submitting inappropriate inputs to $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$, and causing all parties to output $\perp$. If the environment let a corrupted party P_j to submit inappropriate input, the simulator detect it, sets ABORT=true and sent `(abort,`P_j`)` to $\mathcal{F}^{f}_{\mathrm{msfe}}$. In is important to note that `(abort,`P_j`)` is sent to $\mathcal{F}^{f}_{\mathrm{msfe}}$ before any of the mediator's simulators engage in a simulation of final round (in which the outputs are generated). In other word, the output of all parties in Ideal/Genuine(r+2,1) is also set to $\perp$. This implies that the output in both executions is identical.

8.6.2 Corrupted mediator

In the corrupted mediator scenario the ideal functionality $\mathcal{F}^{f}_{\mathrm{msfe}}$ allow all the simulators to communicate, where with honest mediator only suitable pairs of simulators were allowed to communicate. This makes the simulation easier, since all the simulators can coordinate their view through $\mathcal{F}^{f}_{\mathrm{msfe}}$. In this scenario our goal is to simulate the view of the corrupted mediator during the execution of $\mathrm{MComp}(\pi)$. Let $\mathcal{D}_{\pi}$ be the dummy adversary interacting with π, and let $\mathcal{S}_{\pi}$ be the simulator guaranteed for $\mathcal{D}_{\pi}$. The

idea of the simulation is to internally invoke $\mathcal{S}_\pi$, where the instructions from the environment regarding the messages delivered to the honest parties is forwarded to the internal $\mathcal{S}_\pi$ and receiving from $\mathcal{S}_\pi$ the next round messages send by the honest parties. We denote by "master" the simulator $\mathcal{S}_{(\mathcal{M},j)}$ where j is the minimal identity of a player participating in $\mathrm{MComp}(\pi)$. The master run internally $\mathcal{S}_\pi$.

Simulation of the interaction with the setup. Whenever a simulator receives a notification from $\bar{\mathcal{G}}_{\overline{\text{acrs}}}$ it forward it to the master. The master forward the notification internally to $\mathcal{S}_\pi$. Once $\mathcal{S}_\pi$ request to output it to $\mathcal{Z}$, the simulator forward it to all identity related simulators in an increasing order. The simulators upon receiving this notification from the master output the notification to $\mathcal{Z}$. Any $(\mathrm{retrieve}, \mathrm{sid}, \mathrm{PID})$ instruction is given to a corrupted party is treated in the same manner. In addition, the simulators record the CRS.

Simulation of corrupted party. In this case both the party and the simulator are corrupted, and they can jointly simulate any mutual computation in $\mathrm{MComp}(\pi)$ using the DELIVER service of $\mathcal{F}_{\text{msfe}}^f$.

Simulation of honest party. Let j denote the party identity of party P_j. Then the simulator $\mathcal{S}_{(j,\mathcal{M})}$ proceeds as follows:

1. Upon output (Input,sid,P_j) from $\mathcal{F}_{\text{msfe}}^f$ and no (Input,sid',P_j) previously received, do as in the honest mediator case.
2. Upon output (Compute,sid,P_j) from $\mathcal{F}_{\text{msfe}}^f$ and (Input,sid',P_j) previously received, do the following:
   (a) if it is the first (Compute,...) message in round rid then $\mathcal{S}_{(j,\mathcal{M})}$ record (Compute,sid,P_j) and outputs $(\mathrm{OUTPUT}, \mathcal{M}, sid)$ to $\mathcal{Z}$. In addition, send (Compute $-$ party,sid, rid) to $\mathcal{S}_{(\mathcal{M},j)}$.
3. Upon input $(\mathrm{APPROVE}, \mathcal{M}, sid)$ from $\mathcal{Z}$ then do as in the honest mediator case.
4. Upon message (Compute $-$ Med,sid) from $\mathcal{S}_{(\mathcal{M},j)}$ then $\mathcal{S}_{(j,\mathcal{M})}$ records this tuple.
5. Upon input `(update,sid)` from $\mathcal{Z}$ and (Compute $-$ Med,sid) recorded, then $\mathcal{S}_{(j,\mathcal{M})}$ does the following:
   (a) if $rid = r + 1$ then ask $\mathcal{S}_{(\mathcal{M},j)}$ whether ABORT=true in simulation of $\mathcal{F}_{\mathrm{OblComp}}^\pi$ with session id sid and set p accordingly. Next, `(output,`P_j`,p)` to $\mathcal{F}_{\text{msfe}}^f$ and halt.
   (b) else set $rid = rid + 1$ and erase (Compute $-$ Med,sid).

Simulation of the corrupted mediator. The simulator $\mathcal{S}_{(\mathcal{M},j)}$ with identity $(\perp, (\mathcal{M}, j))$. Next, it proceeds as follows:

1. Upon message $(\mathrm{APPROVE} - \mathrm{input}, \mathcal{M}, sid)$ from $\mathcal{S}_{(j,\mathcal{M})}$, then $\mathcal{S}_{(\mathcal{M},j)}$ as in the honest mediator case.
2. Upon message $(\mathrm{APPROVE} - \mathrm{compute}, \mathcal{M}, sid)$ from $\mathcal{S}_{(j,\mathcal{M})}$, then $\mathcal{S}_{(\mathcal{M},j)}$ as in the honest mediator case.
3. Upon input $(\mathrm{APPROVE}, \mathcal{M}, sid)$ received from $\mathcal{Z}$, $\mathcal{S}_{(\mathcal{M},j)}$ proceeds as follows:
   (a) if (Input,sid,P_j) recorded then compute and record $\mathrm{com}^{\mathrm{input}} = C(0^n), \mathrm{com}^{\mathrm{rand}} = C(r_j)$ where r_j is chosen at random from $\{0,1\}^\ell$. Next, output $(\mathrm{Send}, sid, \mathrm{com}^{\mathrm{input}}, \mathrm{com}^{\mathrm{rand}})$ to $\mathcal{Z}$.
   (b) if (Compute,sid,P_j) recorded erase (Compute,sid,P_j) and output (Receipt_party, sid, P_j) to $\mathcal{Z}$.
4. Upon message (Compute $-$ party,sid, k) from $\mathcal{S}_{(j,\mathcal{M})}$ then record this message.
5. Upon receiving input $(\mathrm{Compute}, sid, \overline{\mathrm{com}}^{\mathrm{input}}, \overline{\mathrm{com}}^{\mathrm{rand}}, \overrightarrow{C}, C'^{\mathrm{rid}'-1}, \mathrm{rid}', \overrightarrow{\mathrm{dec}}, \mathrm{dec}^{\mathrm{rid}'-1}, CRS')$ and (Receipt_party, sid, P_j) outputted to $\mathcal{Z}$ do:
   (a) if a tuple with session id already recorded and $rid = k$, where k is the round in (Compute $-$ party,sid, k), then update the recorded tuple.
   (b) else record this tuple and send (Compute $-$ Med,sid) to $\mathcal{S}_{(j,\mathcal{M})}$.
6. Upon input `(next_message,sid)` from $\mathcal{Z}$ and$(\mathrm{Compute}, sid, ...)$ recorded. Let (Compute $-$ party,sid, k) be the recorded tuple. Then:
   (a) If $(\mathrm{com}^{\mathrm{input}}, \mathrm{com}^{\mathrm{rand}}, \overrightarrow{C}, \mathrm{k}, CRS) \neq (\overline{\mathrm{com}}^{\mathrm{input}}, \overline{\mathrm{com}}^{\mathrm{rand}}, \overrightarrow{C}', \mathrm{rid}', CRS')$ then set $\mathrm{ABORT}_{sid} = \mathrm{true}$.

(b) if $\overrightarrow{\text{dec}}\left(\text{resp.}, \text{dec}^{\text{rid}'-1}\right)$ is not a valid decommitment to $\overrightarrow{C'}\left(\text{resp.}, C'^{\text{rid}'-1}\right)$, then set $\text{ABORT}_{sid} = \text{true}$.
(c) Let $\overrightarrow{\text{msg}}_1, \ldots, \overrightarrow{\text{msg}}_{\text{rid}'-1}$ be the committed values in $\overrightarrow{C}_j$ and $C_j^{\text{rid}'-1}$. If any of these contain dummy values, set $\text{ABORT}_{sid} = \text{true}$.
(d) erase (Compute − party,sid, k).
(e) if $k < r + 1$ then update the recorded commitment vector $\overrightarrow{C}$ to contain $C'^{\text{rid}'-1}$ and send the messages in $\text{dec}^{\text{rid}'-1}$ to the master copy. Once receive $\overrightarrow{\text{msg}}$ do as follows:
  i. if $\text{ABORT}_{sid} = \text{true}$ then output $(\text{message}, sid, \perp)$ to $\mathcal{Z}$.
  ii. else output $(\text{message}, sid, \overrightarrow{\text{msg}})$ to $\mathcal{Z}$.
(f) if $k = r + 1$ then output $(\text{message}, sid, \perp)$ to $\mathcal{Z}$.

In addition to the above, whenever $\mathcal{S}_\pi$ wishes to send input `(set_input,`P_j`,`x'`)` to $\mathcal{F}_{\text{sfe}}$ the master simulator instructs $\mathcal{S}_{(j,\mathcal{M})}$ to send it to $\mathcal{F}_{\text{msfe}}$. Whenever, $\mathcal{S}_\pi$ wishes to request output of a corrupted party P_j, then the master simulator instructs $\mathcal{S}_{(j,\mathcal{M})}$ to send `(output,`P_j`,p)` to $\mathcal{F}_{\text{msfe}}$. Once $\mathcal{S}_{(j,\mathcal{M})}$ replies with the output of P_j, the master forwards it internally to $\mathcal{S}_\pi$.

To prove that $\mathcal{Z}$'s output in the real-life is indistinguishable from its output in the ideal process with a corrupted mediator we investigate the following hybrid variables:

Ideal/Fake: The output of $\mathcal{Z}$ in an execution in the ideal process with $\mathcal{S}$ and $\mathcal{F}_{\text{msfe}}^f$ (where $\mathcal{S}$ generate commitments to zero instead of commitments to parties inputs).

Ideal/Genuine(k): The output of $\mathcal{Z}$ from the following interaction. Each simulator of the mediator, is given the real input of the party it associated with. When a simulator with identity $((\mathcal{M}, j), \perp)$, where $j \leq \text{k}$, is generating a commitment to a party input, then we let it to compute the commitment by committing to the real input instead of a zero string. Note that Ideal/Genuine(0) is exactly the Ideal/Fake hybrid. For convenience, we denote by $\hat{\mathcal{S}}^k$ this modified simulator.

Real/Genuine: The output of $\mathcal{Z}$ in a real-life execution with parties running the protocol $\text{MComp}(\pi)$, corrupted mediator, and dummy adversary $\mathcal{D}$.

Suppose that the extreme hybrids Ideal/Fake and Real/Genuine are distinguishable. This means either that the hybrids Ideal/Fake and Ideal/Genuine(m) are distinguishable or that the hybrids Ideal/Genuine(m) and Real/Genuine are distinguishable. We will show that this leads to a contradiction to hiding property of the commitment scheme or to the UC security of the underlying protocol π.

Indistinguishability of Ideal/Fake and Ideal/Genuine(m). Let us assume towards a contradiction of a distinguishing environment. From a hybrid argument follows that there exists $0 \leq k < m$ and environment $\mathcal{Z}^k$ that manages to distinguish between Ideal/Genuine(k) and Ideal/Genuine(k+1) with non-negligible probability. Using this environment, we construct a receiver that breaks the hiding property of the commitment scheme with non-negligible probability. This is done in the same manner as in the honest mediator case with the following difference: the receiver plants the received commitment instead of the input commitment generated by the simulator with identity $((\mathcal{M}, k+1), \perp)$. Due to identical analysis, we omit the rest of the details.

Indistinguishability of Ideal/Genuine(m) and Real/Genuine. First, we need to show that no environment is able to distinguish between Ideal/Genuine(m) and Real/Genuine based on its activation. By inspecting the code of the simulator in the corrupted mediator case and the protocol we observe the following:

- The case of delayed output simulation is the same as in the honest simulation case and depend on rid. Moreover, the commitments to input and randomness outputted by a corrupted mediator are generated only when $\mathcal{Z}$ approve a delivery.
- The parties generate their output when the round id rid, as maintained by the party simulator, reaches $r + 1$.
- The messages send by the parties are outputted to $\mathcal{Z}$ only upon its request and only if both parties submitted inputs to the this simulated instance of $\mathcal{F}_{\text{OblComp}}^{\pi}$. It is important to note that based on the properties of the underlying protocol π and and its security, we are promised the following: upon

receiving a message vector addressed to honest party P_j , the simulator $\mathcal{S}_\pi$ immediately output the simulated messages send by P_j in the next round of π.

Therefore, in order to prove identical scheduling in both hybrids we need to show that the round id rid of each simulator associated with a honest party P_j (i.e a simulator with identity $((j, \mathcal{M}), \perp)$) in Ideal/Genuine(m) is identically set during all the execution. The rid is set to '1' as in the honest mediator case and is identically to the real execution for the same reasons. The rid advances exactly when the environment instructs the adversary to let $\mathcal{F}^{\pi}_{\text{OblComp}}$ to give output to this party (and the corresponding mediator's simulator declared on submission of input to this instance of $\mathcal{F}^{\pi}_{\text{OblComp}}$). Therefore, whenever rid is advanced in Ideal/Genuine(m) it is also advanced by the honest party in the Real/Genuine. Moreover, in round $r + 1$ the simulator request an output from $\mathcal{F}^{f}_{\text{msfe}}$ and halt, identically to Real/Genuine.

Now we are ready to complete the validity proof by showing that no environment can distinguish between Ideal/Genuine(m) and Real/Genuine. Assume for contradiction that there is an environment $\mathcal{Z}$ such that $\text{LEXEC}_{\text{MComp}(\pi),\mathcal{D},\mathcal{Z}} \not\approx \text{LEXEC}_{\mathcal{F}^{f}_{\text{msfe}},\hat{\mathcal{S}}^{m},\mathcal{Z}}$. We construct a UC environment $\mathcal{Z}_\pi$ such that $\text{EXEC}_{\pi,\mathcal{D}_\pi,\mathcal{Z}_\pi} \not\approx \text{EXEC}_{\mathcal{F}^{f}_{\text{sfe}},\mathcal{S}_\pi,\mathcal{Z}_\pi}$. Environment $\mathcal{Z}_\pi$ runs internally an interaction between simulated instances of $\mathcal{Z}$ and $\mathcal{D}$ in a presence of a global $\bar{\mathcal{G}}_{\text{acrs}}$. In addition:

1. Any output of $\bar{\mathcal{G}}_{\text{acrs}}$ coming from the external adversary is forwarded to $\mathcal{Z}$ as if they are coming from the dummy adversaries with the associated identities. Similarly, any $(\text{retrieve}, \text{sid}, \text{PID})$ instruction is forwarded to the external adversary.
2. Whenever $\mathcal{Z}$ passes input (Input,sid,x_j) to party P_j, $\mathcal{Z}_\pi$ honestly simulates an invocation of $\mathcal{F}_{\text{smt}}$ and gives$\mathcal{Z}$ a commitment to x_j and random string of length ℓ. Next it forwards x_j to the external party with party identity j.
3. Whenever $\mathcal{Z}$ passes input (Compute, sid) to party P_j, $\mathcal{Z}_\pi$ honestly simulates the delayed output of $\mathcal{F}^{\pi}_{\text{OblComp}}$ and gives $\mathcal{Z}$ a message (Receipt_party, sid, P_j).
4. Whenever $\mathcal{Z}$ passes input `(update,sid)` to $\mathcal{S}_{(j,\mathcal{M})}$, $\mathcal{Z}_\pi$ honestly simulates $\mathcal{F}^{\pi}_{\text{OblComp}}$. That is:
   (a) if the round id is smaller than update the round id of the local P_j.
   (b) else:
      i. check the correctness of the recorded tuple as $\mathcal{F}^{\pi}_{\text{OblComp}}$ and set ABORT accordingly.
      ii. instruct the external adversary to deliver to P_j the messages in $\text{dec}_j^{\text{rid}'-1}$.
      iii. upon receiving the output OUT from external P_j, give OUT or $\perp$to $\mathcal{Z}$ according to the ABORT value.
5. Whenever $\mathcal{Z}$ passes input $(\text{Compute}, sid, \text{com}_j^{\text{input}}, \text{com}_j^{\text{rand}}, \overrightarrow{C_j}, C_j^{\text{rid}'-1}, \text{rid}', \overrightarrow{\text{dec}_j}, \text{dec}_j^{\text{rid}'-1}, CRS')$ to $\mathcal{M}$, $\mathcal{Z}_\pi$ record this tuple.
6. Whenever $\mathcal{Z}$ passes input `(next_message,sid)` to $\mathcal{S}_{(\mathcal{M},j)}$, $\mathcal{Z}_\pi$ honestly simulates $\mathcal{F}^{\pi}_{\text{OblComp}}$. That is:
   (a) check the correctness of the recorded tuple as $\mathcal{F}^{\pi}_{\text{OblComp}}$ and set ABORT accordingly.
   (b) instruct the external adversary to deliver to P_j the messages in $\text{dec}_j^{\text{rid}'-1}$.
   (c) upon receiving the next round messages vector $\overrightarrow{\text{msg}}$ of P_j from the external adversary, give $\overrightarrow{\text{msg}}$ or $\perp$ to $\mathcal{Z}$ according to the ABORT value in the current round.
7. Finally, $\mathcal{Z}_\pi$ outputs whatever the simulated $\mathcal{Z}$ outputs.

From inspecting the code of $\mathcal{Z}_\pi$ and $\hat{\mathcal{S}^m}$ it follows that if $\mathcal{Z}_\pi$ interacts with parties running π then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\pi$ is distributed identically to the view of $\mathcal{Z}$ when interacting with $\text{MComp}(\pi)$ and adversaries running $\mathcal{D}$ in Real/Genuine. Similarly, if $\mathcal{Z}_\pi$ interacts with parties running $\text{IDEAL}_{\mathcal{F}^{f}_{\text{sfe}}}$ then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\pi$ is distributed identically to the view of $\mathcal{Z}$ when interacting with $\mathcal{F}^{f}_{\text{msfe}}$ and adversaries running $\hat{\mathcal{S}^m}$ in Ideal/Genuine(m). Therefore,

$$\text{EXEC}_{\pi,\mathcal{D}_\pi,\mathcal{Z}_\pi} \not\approx \text{EXEC}_{\mathcal{F}^{f}_{\text{sfe}},\mathcal{S}_\pi,\mathcal{Z}_\pi}$$

, as desired.

8.7 Putting all together

The constructed compiler $\mathrm{MComp}()$ operates in the $(\mathcal{F}_{\overline{\mathrm{smt}}}, \mathcal{F}^{\pi}_{\mathrm{OblComp}}, \bar{\mathcal{G}}_{\overline{\mathrm{acrs}}})$-hybrid model. In this section we show how to transform $\mathrm{MComp}()$ to a new compiler in a $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$-hybrid model. Moreover, we show that in the corrupted mediator case, the LUC realization is equivalent to UC realization. In order to achieve this goal, we first show how to LUC realize $\mathcal{F}_{\overline{\mathrm{smt}}}$ and $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$.

Recall that $\mathcal{F}_{\overline{\mathrm{smt}}}$ is a merger functionality of the UC secure message transmission functionality $\mathcal{F}_{\mathrm{smt}}$. Therefore, combining Claim 8.2.2, stating the existence of realizing protocol for $\mathcal{F}_{\mathrm{smt}}$, with the equivalence theorem 5 we obtain realization of $\mathcal{F}_{\overline{\mathrm{smt}}}$. That is:

Lemma 2. *There exists a protocol that LUC-realizes $\mathcal{F}_{\overline{\mathrm{smt}}}$ in the $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$-hybrid model in the presence of static adversaries over unauthenticated channels.*

Next, consider the UC functionality $\widetilde{\mathcal{F}}$ such that the merger functionality of $\widetilde{\mathcal{F}}$ is $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ (as defined in Definition 5). $\widetilde{\mathcal{F}}$ is a well-formed two-party functionality; therefore, by claim 8.2.2, $\widetilde{\mathcal{F}}$ can be UC-realized in the$\bar{\mathcal{G}}_{\mathrm{acrs}}$-hybrid model. Combining this with theorem 5, we obtain that:

Lemma 3. *There exists a protocol that LUC-realizes $\mathcal{F}^{\pi}_{\mathrm{OblComp}}$ in the $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$-hybrid model in the presence of static adversaries over unauthenticated channels.*

Now we can wrap it all together using the LUC composition theorem and obtain a compiler in the $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$-hybrid model. That is, combining Lemma 2 and 3 with theorem 11 we have the following:

Theorem 12. *Given a (poly-time) function $f = (f_1, ..., f_n)$ and a protocol π that is setup oblivious UC-realization of the well-formed aborting functionality $\mathcal{F}^{f}_{\mathrm{sfe}}$. Then there exists a compiled protocol $\widetilde{\mathrm{MComp}}(\pi)$ that LUC-realize $\mathcal{F}^{f}_{\mathrm{msfe}}$ in the $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$-hybrid model in the presence of static adversaries over unauthenticated channels, with PID-wise corruptions.*

We note that the compiler $\widetilde{\mathrm{MComp}}()$ is exactly the compiler $\mathrm{MComp}()$ composed with the protocols of [CDPW07] and [Wal08] (transformed to the $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$-hybrid model as described above) to replace the $\mathcal{F}_{\overline{\mathrm{smt}}}$ and $\mathcal{F}^{\pi}_{\mathrm{oblComp}}$ instances in $\mathrm{MComp}()$.

Finally that the compiler is instantiated, we are ready to show that $\widetilde{\mathrm{MComp}}()$ is also a UC secure realization of the SFE functionality when the mediator corrupted. More formally, let $f = (f_1, ..., f_n)$ be a poly-time function. Consider the UC functionality $\hat{\mathcal{F}}^{f}{}_{\mathrm{sfe}}$ such that the merger functionality of $\hat{\mathcal{F}}^{f}{}_{\mathrm{sfe}}$ is $\mathcal{F}^{f}_{\mathrm{msfe}}$ restricted on the malicious mediator case (as defined in 5). By theorem 12 and theorem5 we obtain that:

Theorem 13. *Given a (poly-time) function $f = (f_1, ..., f_n)$ and a protocol π that is setup oblivious UC-realization of the well-formed aborting functionality $\mathcal{F}^{f}_{\mathrm{sfe}}$. Then the compiled protocol $\widetilde{\mathrm{MComp}}(\pi)$ UC-realize $\hat{\mathcal{F}}^{f}{}_{\mathrm{sfe}}$ in the $\bar{\mathcal{G}}_{\mathrm{acrs}}$-hybrid model in the presence of static adversaries over unauthenticated channels, with PID-wise corruptions.*

We note that the only difference between $\hat{\mathcal{F}}^{f}{}_{\mathrm{sfe}}$ and $\mathcal{F}^{f}_{\mathrm{sfe}}$ is the additional inputs from parties, i.e., "Compute" inputs.

8.8 A Solution for adaptive adversaries

We now show that our construction LUC-realizes$\mathcal{F}^{f}_{\mathrm{msfe}}$ in the presence of adaptive adversaries. The difference between this protocol and the protocol for static adversaries is in the properties of the underlying commitment scheme C in use. Essentially, here we use a commitment scheme that is "adaptively secure". We start by introducing the new underlying components, and then we proceed to extending the above result to adaptive corruption.

Commitments. Let $\hat{C}$ be a non-interactive equivocal commitment scheme, where $\hat{C}(m;r)$ denotes a commitment to m using random coins r. The decommitment of $\text{com} = \hat{C}(m;r)$ is $\text{dec} = (m;r)$. In particular, the commitment scheme $\text{UCC}_{\text{ReUse/NotErase}}$ from [CF01] satisfies this requirement, where the local setup is replaced with a global knowledge-based registration functionality $\mathcal{G}_{\text{krk}}$ of [CDPW07]. The $\mathcal{G}_{\text{krk}}$, upon receiving a (register,sid) from the mediator generate public and secret keys for a claw-free trapdoor permutation. This can be improved in term of setup length by using the $\text{UCC}_{\text{ReUse}}$ protocol in the augmented CRS model, where the used encryption scheme is IBE CCA-secure scheme of [BCHK06]. However, this holds only for erasing parties. This commitment scheme is used only by the parties in the INPUT phase of the protocol.

Key Exchange functionality. The key exchange functionality $\mathcal{F}_{\text{ke}}$, presented in[Wal08], offers a symmetric key exchange service. The functionality allows a corrupted receiver to determine the symmetric key, where in the honest receiver case a fresh key is sampled. We extend this functionality to the LUC framework as follows. The functionality $\mathcal{F}_{\overline{\text{ke}}}$ behaves the same as $\mathcal{F}_{\text{ke}}$ with the difference that the receiver's adversary is allowed to determine the symmetric key in the corrupted receiver case. The functionality formally presented in Figure 33.

Functionality $\mathcal{F}_{\overline{\text{ke}}}$

Functionality $\mathcal{F}_{\overline{\text{ke}}}$ running with parties S and R and the adversaries $\mathcal{S}_{(S,R)}$, $\mathcal{S}_{(R,S)}$, proceeds as follows:

- Upon receiving message of the form $(\text{keyexchange}, \text{sid}, S, R)$ from party S, if S is not yet "active", mark S "active", and a send public delayed output $(\text{keyexchange}, \text{sid}, S, R)$ to R. (Otherwise, ignore the message).
- Upon receiving a message of the form $(\text{keyexchange}, \text{sid}, S, R)$ from party R, if R is not yet "active", mark R as "active" and send a public delayed output $(\text{keyexchange}, \text{sid}, S, R)$ to S. (Otherwise, ignore the message.)
- Upon receiving a message of the form $(\text{setkey}, \text{sid}, S, R, k')$ from $\mathcal{S}_{(R,S)}$, if R is corrupt and S is "active", then send private delayed output $(\text{setkey}, \text{sid}, S, R, k')$ to S and R, and halt. If R is "active" but not corrupt, then sample a fresh key $k \xleftarrow{\$} \{0,1\}^{\lambda}$ and send private delayed output $(\text{setkey}, \text{sid}, S, R, k)$ to R. Furthermore, if S is "active", then send private delayed output $(\text{setkey}, \text{sid}, S, R, k)$ to S as well. In all cases, this completes the protocol, and the functionality halts.

Fig. 33: The symmetric key exchange functionality $\mathcal{F}_{\overline{\text{ke}}}$

Claim. For any adaptively well-formed aborting ideal functionality $\mathcal{F}$, there exists a $(\bar{\mathcal{G}}_{\text{acrs}}, \mathcal{F}_{\text{ke}})$-setup oblivious protocol π that UC-realizes $\mathcal{F}$ in the $(\bar{\mathcal{G}}_{\text{acrs}}, \mathcal{F}_{\text{ke}})$-hybrid model in the presence of adaptive adversaries over unauthenticated channels, where all corruptions are PID-wise.

Here, as opposed to the static case we need to use adaptively secure authentication protocol. [Wal08] present such an authentication protocol in the $\mathcal{F}_{\text{ke}}$-hybrid model. as before, the setup obliviousness obtained by performing all the interaction with the setups in an offline stage.

We note that although the authentication functionality $\mathcal{F}_{\text{auth}}$ is cryptographically equivalent to $\mathcal{F}_{\text{ke}}$, protocols in the $\mathcal{F}_{\text{auth}}$-hybrid model are not $\mathcal{F}_{\text{auth}}$-setup oblivious.

Oblivious computation and the mediated compiler. The only difference in $\mathcal{F}^{\pi}_{\text{oblComp}}$ is that it expects also to receive the share keys from the party and it computes the messages of the next round using the shared keys as well. The initialization phase in the mediated compiler $\text{MComp}()$ for some party also includes invoking an instance of $\mathcal{F}_{\overline{\text{ke}}}$ for each pair of parties. In addition, the parties commit

to their inputs and randomness using $\hat{C}$. Note that the mediator commits to the message vectors using the previous commitment scheme C from the static case.

Now we can show the results with respect to adaptive corruption.

Theorem 14. *Given a (poly-time) function $f = (f_1, ..., f_n)$ and a protocol π that is $(\bar{\mathcal{G}}_{\mathrm{acrs}}, \mathcal{F}_{\mathrm{ke}})$-setup oblivious UC-realization of the well-formed aborting functionality $\mathcal{F}_{\mathrm{sfe}}^{f}$. Then the compiled protocol $\mathrm{MComp}(\pi)$ LUC-realize $\mathcal{F}_{\mathrm{msfe}}^{f}$ in the $(\mathcal{F}_{\overline{\mathrm{smt}}}, \mathcal{F}_{\mathit{oblComp}}^{\pi}, \bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}, \mathcal{F}_{\overline{\mathrm{ke}}})$-hybrid model in the presence of adaptive adversaries, where all corruptions are PID-wise.*

Proof. The proof of the theorem is almost identical to the one in the case of static adversaries. Only this time the ideal-model simulator needs also to reconstruct the internal state of the newly corrupted party. As for the static case, we analyze the validity of the simulator for the malicious and honest mediator separately.

Honest mediator. The simulator $\mathcal{S}$ works similarly to the static case with a difference in simulating $\mathcal{F}_{\mathrm{oblComp}}^{\pi}$. that is, if the environment manages at some point in the execution to present an input or randomness decommitment to a different values then the previous decommitment, then the simulator sends $(\mathtt{abort}, P_j)$ to $\mathcal{F}_{\mathrm{msfe}}^{f}$. In addition, a (Corrupt, P_j, p) input is forwarded to $\mathcal{F}_{\mathrm{msfe}}^{f}$ and upon receiving P_j's input x_j, the simulator reconstruct the internal state as follows:

- honestly generate commitments to x_j and random string r_j of length ℓ using $\hat{C}$.
- for each round up to the current round the simulator generate a commitment to all zero string using C, as a commitment that P_j received from $\mathcal{F}_{\mathrm{oblComp}}^{\pi}$.
- Provide the environment with the above commitments and a decommitments to the input and randomness.

We note that the internal state upon corruption is indistinguishable between Ideal/Fake and Real/Genuine.

The validity of $\mathcal{S}$ is proved as in the static case with a slightly different analysis in the "Indistinguishability of Ideal/Genuine(r+2,1) and Real/Genuine" scenario where no deviations occur. Potentially, the environment may present a different decommitment and cause the honest parties in Ideal/Genuine(r+2,1) to output $\perp$, where in the Real/Genuine execution they will generate an output according to π. However, due to the computational binding property of $\hat{C}$, this event can happen only with negligible probability. Therefore, the indistinguishability follows.

Corrupted mediator. The construction of the simulator is similar to the case of static corruptions. The differences are when the mediator commits to parties input and randomness or a party is corrupted. In addition, it also needs to simulate the interaction with the key-exchange functionality (that do not exist in the static case). That is:

Simulation of the interaction with the setup. The simulation of the interaction with $\bar{\mathcal{G}}_{\overline{\mathrm{acrs}}}$ is done as in the static case. To mimic the behavior of $\mathcal{F}_{\mathrm{ke}}$ we let $\mathcal{S}_\pi$ to schedule the delayed output in $\mathcal{F}_{\overline{\mathrm{ke}}}$. We note that in $\mathcal{F}_{\overline{\mathrm{ke}}}$ the scheduling is done by both adversaries, and therefore, we let $\mathcal{S}_\pi$ to play the role of the second adversary in the scheduling process. The scheduling done by the first adversary is simulated by executing the environment's instructions without involving $\mathcal{S}_\pi$. The scheduling done by the second adversary is simulated by forwarding the scheduling instructions of the environment internally to $\mathcal{S}_\pi$.

Generation of input and randomness commitments. Recall that once the mediator is corrupted it retrieves the trapdoor of the claw-free permutation. In out case, the simulator with identity $((\mathcal{M}, j), \perp)$ generate equivocal commitments using the trapdoor and hands it to the environment as if they were given by P_j (via $\mathcal{F}_{\overline{\mathrm{smt}}}$).

Party corruption. Assume that the environment demands to corrupt a party. Then the simulator forward it to $\mathcal{S}_\pi$ and gets all its internal state in π (this done by corrupting it in the ideal model and receive from $\mathcal{S}_\pi$ its internal state in π). Then the simulator modifies all decommitment information about input and randomness commitments of this party to match the

received data. Next, the simulator hands this modified internal information and the round commitments (received from the environment as part of input instruction to $\mathcal{F}^{\pi}_{\text{oblComp}}$) to the environment.

The validity proof is identical to the case of static corruptions. Moreover, it can be easily verified that the internal state presented to the environment in Ideal/Fake is indistinguishable from Real/Genuine.

8.8.1 Conclusions

As in the static case, we first need to extend the correspondence between the LUC and the UC realizations to the key exchange hybrid model.

Theorem 15. *Let $\mathcal{F}$ be a functionality in the UC model and π be some $(\bar{\mathcal{G}}_{\text{acrs}}, \mathcal{F}_{\text{ke}})$-hybrid protocol. Then π UC-realizes $\mathcal{F}$ with respect to static PIDs environments if and only if $\bar{\pi}$ LUC-realizes the merger functionality $m\mathcal{F}$ in the $(\bar{\mathcal{G}}_{\overline{\text{acrs}}}, \mathcal{F}_{\overline{\text{ke}}})$-hybrid model.*

Sketch of Proof The proof is similar to the proof presented for the static adversaries. First, we show that UC realization of $\mathcal{F}$ implies LUC realization of $m\mathcal{F}$ also with global setup. Let $\mathcal{S}$ be the LUC simulator constructed in theorem 6 for the dummy adversary. To mimic the behavior of $\mathcal{F}_{\text{ke}}$ we do the same as in the proof of corrupted mediator above. Except this, any message of $\mathcal{S}_{\mathcal{D}}$ is forwarded to the appropriate external simulator and any received messages is forwarded internally to $\mathcal{S}_{\mathcal{D}}$.

To show that LUC realization of $m\mathcal{F}$ implies UC realization of $\mathcal{F}$ with $\mathcal{F}_{\text{ke}}$ we take, as before, the simulator constructed in 6 and extend it to simulate interaction with the global setup. The scheduling in $\mathcal{F}_{\text{ke}}$ is done as follows: The $(\text{APPROVE}, P, p)$ messages from $\mathcal{F}_{\text{ke}}$ are forwarded to the first internal simulator. Once approved it is forwarded to the second internal simulator that takes the role of the scheduling adversary.

We do not present the validity proof as it repeats the same ideas as in 3.5. □

Repeating the same steps as in the static scenario we obtain the following:

Theorem 16. *Given a (poly-time) function $f = (f_1, ..., f_n)$ and a protocol π that is setup oblivious UC-realization of the well-formed aborting functionality $\mathcal{F}^{f}_{\text{sfe}}$. Then there exists a compiled protocol $\widetilde{\text{MComp}}(\pi)$ that LUC-realize $\mathcal{F}^{f}_{\text{msfe}}$ in the $(\bar{\mathcal{G}}_{\overline{\text{acrs}}}, \mathcal{F}_{\overline{\text{ke}}})$-hybrid model in the presence of adaptive adversaries over unauthenticated channels, with PID-wise corruptions.*

Theorem 17. *Given a (poly-time) function $f = (f_1, ..., f_n)$ and a protocol π that is setup oblivious UC-realization of the well-formed aborting functionality $\mathcal{F}^{f}_{\text{sfe}}$. Then the compiled protocol $\widetilde{\text{MComp}}(\pi)$ UC-realize $\hat{\mathcal{F}}^{f}{}_{\text{sfe}}$ in the $(\bar{\mathcal{G}}_{\text{acrs}}, \mathcal{F}_{\text{ke}})$-hybrid model in the presence of adaptive adversaries over unauthenticated channels, with PID-wise corruptions.*

Acknowledgments

We thank Noam Livne and Vassilis Zikas for helpful discussions. We also thank Ilia Gorelik and Daniel Shahaf for help.

References

AKL+09. J. Alwen, J. Katz, Y. Lindell, G. Persiano, A. shelat, and I. Visconti. Collusion-free multiparty computation in the mediated model. Crypto 2009, volume 5677 of LNCS, pages 524-540. Springer, 2009.

AKMZ12. J. Alwen, J. Katz, U. Maurer, V. Zikas. Collusion-Preserving computation. CRYPTO 2012 (to appear). An online version available at Eprint archive, http:/eprint.iacr.org/2011/433.pdf.

ASV08. J. Alwen, a. shelat, and I. Visconti. Collusion-free protocols in the mediated model. CRYPTO 08, pages 497–514, 2008.

BCHK06. D. Boneh, R. Canetti, S. Halevi, and J. Katz. Chosen-Ciphertext Security from Identity-Based Encryption. SIAM J. Comput., 36(5): 1301-1328 (2007).
CL05. J. Camenisch, A. Lysyanskaya: A Formal Treatment of Onion Routing. CRYPTO 2005, 169-187.
Can00. R. Canetti. Security and composition of multiparty cryptographic protocols. J. Cryptology, 13(1):143-202, 2000.
Can01. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. 42nd FOCS, 2001. Revised version (2005) available at eprint.iacr.org/2000/067.
CDPW07. R. Canetti, Y. Dodis, R. Pass, and S. Walsh. Universally composable security with global setup. TCC 2007, volume 4392 of LNCS, pages 61-85. Springer, 2007.
CF01. R. Canetti and M. Fischlin. Universally Composable Commitments. Crypto, 2001. Long version at eprint.iacr.org/2001/055.
CGT95. C. Crépeau, J. van de Graaf, and A. Tapp. Committed Oblivious Transfer and Private Multi-Party Computations. Advances in Cryptology: Proceedings of Crypto '95 , Springer-Verlag, pages 110-123, 1995.
CLOS02. R. Canetti, Y. Lindell, R. Ostrovsky, A. Sahai. Universally composable two-party and multi-party secure computation. 34th STOC, 2002.
DM00. Y. Dodis and S. Micali. Secure Computation. CRYPTO '00, 2000.
DKSW09. Y. Dodis, J. Katz, A. Smith, and S. Walsh. Composability and on-line deniability of authentication. In Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography, TCC '09, pages 146-162, Berlin, Heidelberg, 2009. Springer-Verlag.
GL90. S. Goldwasser, and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. CRYPTO '90, LNCS 537, 1990.
GLR10. R. Gradwohl, N. Livne and A. Rosen. Sequential Rationality in Cryptographic Protocols. Proceedings of the 2010 Annual IEEE Symposium on Foundations of Computer Science. 51: 623-632.
GM84. S. Goldwasser and S. Micali. Probabilistic encryption. JCSS, Vol. 28, No 2, April 1984, pp. 270-299.
GMR85. S. Goldwasser, S. Micali and C. Rackoff, The Knowledge Complexity of Interactive Proof-Systems", SIAM J. Comput. 18 (1989), pp. 186-208; (also in STOC 85, pp. 291-304.)
GMW87. O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game. 19th Symposium on Theory of Computing (STOC), ACM, 1987, pp. 218-229.
Gol04. O. Goldreich. Foundations of Cryptography, vol. 2: Basic Applications. Cambridge University Press, Cambridge, UK, 2004.
HKN05. Shai Halevi, Paul A. Karger, and Dalit Naor. Enforcing confinement in distributed storage and a cryptographic model for access control. Cryptology Eprint Archive Report 2005/169, 2005.
ILM05. S. Izmalkov, M. Lepinski, and S. Micali. Rational Secure Computation and Ideal Mechanism Design. In FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, pages 585-595, Washington, DC, USA, 2005. IEEE Computer Society.
ILM08. S. Izmalkov, M. Lepinski, and S. Micali. Veriably secure devices (and correlated equilibrium). In Theory of Cryptography Conference, February 2008.
ILM11. S. Izmalkov, M. Lepinski, and S. Micali. Perfect implementation. Games and Economic Behavior, 71(1):121-140, January 2011.
Lam73. Butler W. Lampson. A note on the confinement problem. Communications of the ACM, 16(10):613–615, October 1973.
LMS05. M. Lepinksi, S. Micali, and A. shelat. Collusion-Free Protocols. In STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pages 543-552, New York, NY, USA, 2005. ACM.
MR11. U. Maurer and R. Renner. Abstract cryptography. In Innovations in Computer Science. Tsinghua University Press, 2011.
MR91. S. Micali and P. Rogaway. Secure Computation. unpublished manuscript, 1992. Preliminary version in CRYPTO '91, LNCS 576, 1991.
NMO08. W. Nagao, Y. Manabe, T. Okamoto. Relationship of Three Cryptographic Channels in the UC Framework, ProvSec 2008, LNCS Vol. 5324, pp.268-282.
OPW11. A. O'Neill, C. Peikert, B. Waters. Bi-Deniable Public-Key Encryption. CRYPTO 2011: 525-542.
PS04. M. Prabhakaran, A. Sahai. New notions of security: achieving universal composability without trusted setup. 36th STOC, pp. 242–251. 2004.
PW00. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. 7th ACM Conf. on Computer and Communication Security, 2000, pp. 245-254.
Wal08. S. Walfish. Enhanced Security Models for Network Protocols. Available at http://www.cs.nyu.edu/web/Research/Theses/walfish_shabsi.pdf.