Universal Composability from Essentially Any Trusted Setup

Mike Rosulek*

Department of Computer Science, University of Montana. mikero@cs.umt.edu.

Abstract. It is impossible to securely carry out general multi-party computation in arbitrary network contexts like the Internet, unless protocols have access to some trusted setup. In this work we classify the power of such trusted (2-party) setup functionalities. We show that nearly every setup is either **useless** (ideal access to the setup is equivalent to having no setup at all) or else **complete** (composably secure protocols for *all* tasks exist in the presence of the setup). We further argue that those setups which are neither complete nor useless are highly unnatural.
The main technical contribution in this work is an almost-total characterization of completeness for 2-party setups. Our characterization treats setup functionalities as black-boxes, and therefore is the first work to classify completeness of *arbitrary setup functionalities* (*i.e.*, randomized, reactive, and having behavior that depends on the global security parameter).

1 Introduction

When a protocol is deployed in a vast network like the Internet, it may be executed in the presence of concurrent instances of other arbitrary protocols with possibly correlated inputs. A protocol that remains secure in such a demanding context is called *universally composable*. This security property is highly desirable; unfortunately, it is simply too demanding to be achieved for every task. Canetti's UC framework [5] provides the means to formally reason about universal composability in a tractable way, and it is widely regarded as the most realistic model of security for protocols on the Internet. A sequence of impossibility results [8,30,9] culminated in a complete characterization for which tasks are securely realizable in the UC framework [35]. Indeed, universal composability is impossible for almost all tasks of any cryptographic interest, under any intractability assumption.

For this reason, there have been many attempts to slightly relax the UC framework to permit secure protocols for more tasks, while still keeping its useful composition properties. Many of these variants are extremely powerful, permitting composably-secure protocols for *all* tasks; for example: adding certain trusted setup functionalities [8,11,1,12,17,23,21,31], allowing superpolynomial simulation [34,36,2,32,10], assuming bounded network latency [22], considering uniform adversaries [29], and including certain global setups [7], to name a

* Supported by NSF grant CCF-1149647.

few (for a more comprehensive treatment, see the survey by Canetti [6]). Other variants of the UC framework turn out to be no more powerful than the original framework; for example, adding certain setup functionalities [35,25] or global setups [7], and requiring only self-composition rather than universal composition [30]. A natural question is, therefore: under what circumstances can universal composability be achieved?

1.1 Our Results

In this work we study the power of 2-party trusted setups. In other words, given access to a particular trusted setup functionality (*e.g.*, a common random string functionality), what tasks have UC-secure protocols? In particular, two extremes are of interest. First, we call a trusted setup $\mathcal{F}$ **useless** if having ideal access to $\mathcal{F}$ is equivalent to having no trusted setup at all. More precisely, $\mathcal{F}$ is useless if it already has a UC-secure protocol in the plain (no trusted setups) model. A complete characterization for such functionalities was given in [35].

At the other extreme, we call a trusted setup $\mathcal{F}$ **complete** if *every* well-formed task has a UC-secure protocol given ideal access to $\mathcal{F}$. As mentioned above, many setups are known to be complete (*e.g.*, a common random string functionality), but the methods for demonstrating their completeness have been quite *ad hoc*. Our major contribution is to give a new framework for understanding when a trusted setup is complete.

Informal statement of the results. Our characterization is based on the concept of *splittability* from [35]. To give a sense of splittability, imagine the following two-party game between a "synchronizer" and a "distinguisher." The parties connect to two independent instances of $\mathcal{F}$, each party playing the role of Alice in one instance and the role of Bob in the other. They are allowed to arbitrarily interact with these two instances of $\mathcal{F}$. The synchronizer's goal is to make the two instances behave like a single instance of $\mathcal{F}$, from the distinguisher's point of view. The distinguisher's goal is to make the two instances act noticeably different from a single instance of $\mathcal{F}$, from his point of view.

Then, informally, we say that $\mathcal{F}$ is **splittable** if the synchronizer has a winning strategy in this game, and $\mathcal{F}$ is **strongly unsplittable** if the distinguisher has a winning strategy.[1] Importantly, these splittability-based conditions are relatively easy to check for a candidate setup, and apply uniformly to *completely arbitrary* functionalities in the UC framework (*i.e.*, possibly randomized, reactive, with behavior depending on the security parameter). Prabhakaran & Rosulek [35] showed that $\mathcal{F}$ is useless if and only if $\mathcal{F}$ is splittable. Analogously, our main result is the following:

Theorem (Informal). *If $\mathcal{F}$ is strongly unsplittable*, then $\mathcal{F}$ is complete.*

[1] In the formal definition, both players are computationally bounded, so it may be that neither party has a feasible winning strategy in the 2-party game. See Section 3.2.

The asterisk after "strongly unsplittable" indicates that the precise statement of our result involves a variant of strong unsplittability, in which the "synchronizer" is allowed to obtain the internal state of one of the instance of $\mathcal{F}$. However, in the case that $\mathcal{F}$ is a *nonreactive* functionality, the informal statement above is correct; the asterisk can be safely ignored.

Our completeness theorem is proved under the assumption that there exists a **s**emi-**h**onest secure **o**blivious **t**ransfer protocol (the SHOT assumption), an intractability assumption we show is necessary. We also show that a very slight modification of strong unsplittability is *necessary* for a setup to be complete, and so our characterization is nearly tight.

We argue that setups which are neither splittable nor strongly unsplittable exploit low-level technical "loopholes" of the UC framework or are otherwise highly unnatural. Thus, combining our result with that of [35], we can informally summarize the power of trusted setups by saying that every "natural" setup is either useless or complete.

Our notion of completeness. Many prior completeness results place restrictions on how protocols are allowed to access a setup functionality. A common restriction is to allow protocols to access only one instance of the setup — typically only at the beginning of a protocol. In these cases, we say that the protocols have only *offline access* to the setup. Additionally, some completeness results construct a *multi-session* commitment functionality from such offline access to the setup, modeling a more globally available setup assumption.

In contrast, the completeness results in this work are of the following form. We say that a functionality $\mathcal{F}$ is a **complete** setup if there is a UC-secure protocol for the ideal (single-session) commitment functionality in the $\mathcal{F}$-hybrid model. This corresponds to the complexity-theoretic notion of completeness, under the reduction implicit in the conventional UC security definition. As is standard for protocols in the $\mathcal{F}$-hybrid model, we place no restrictions on how or when protocols may access $\mathcal{F}$ or how many instances of $\mathcal{F}$ they may invoke. However, we point out that completeness for offline access can be achieved by simply using our construction to generate a common random string in an offline phase, then applying a result such as [11].

Technical overview. To prove that a functionality $\mathcal{F}$ is complete, it suffices to show that there is a UC-secure protocol for bit commitment, given ideal access to $\mathcal{F}$. This follows from the well-known result of Canetti *et al.* [11] that commitment is complete, under the SHOT assumption. We construct a commitment protocol in several steps: In Sections 4 & 5 we construct (for the two cases in our main theorem) commitment protocols that have a straight-line UC simulator for corrupt receivers (*i.e.*, an equivocating simulator) but have only a rewinding simulator for corrupt senders (*i.e.*, an extracting simulator). Then, in Section 6 we show how to use such a "one-sided" UC-secure commitment protocol to build a full-fledged UC-secure commitment protocol.

In Section 7 we show that several variants of strong unsplittability are necessary for a setup to be complete. These variants involve only very minor technical modifications to the strong unsplittability definition.

1.2 Related Work

Dichotomies in the cryptographic power of functionalities are known in several settings [13,28]. Our work follows a rich line of research exhibiting dichotomies specifically between *useless* and *complete* functionalities, for various security models and restricted to various classes of functionalities [3,26,27,18,31,24]. Among these results, only [31] considered *reactive* functionalities, and only [26] considered *randomized* functionalities. In comparison, ours is the first work to consider the full range of arbitrary functionalities allowed by the UC framework (the class of functionalities is stated explicitly in Section 2.2).

Among the results listed above, two [31,24] are cast in the UC framework; for these we give a more detailed comparison to our own results. Maji, Prabhakaran, and Rosulek [31] showed a uselessness/completeness dichotomy among *deterministic* functionalities whose internal state and input/output alphabets are constant-sized (*i.e.*, independent of the security parameter). Their approach relies heavily on deriving combinatorial criteria for such functionalities, expressed as finite automata, whereas our classification achieves much greater generality by treating functionalities essentially as black-boxes. Concurrently and independently of this work, Katz *et al.* [24] showed a similar result for deterministic, non-reactive (secure function evaluation) functionalities.[2] They construct UC-puzzles [29] for classes of SFE functionalities deemed impossible in the characterization of Canetti, Kushilevitz, and Lindell [9]. Our result achieves greater generality by being based not on the CKL characterization but the splittability characterization of [35]. Furthermore, we show completeness by directly constructing a commitment protocol rather than a UC puzzle (see below).

Lin, Pass, and Venkitasubramaniam [29] developed a framework for proving completeness results in the UC framework and many variants. Their framework is based on "UC-puzzles" — protocols with an explicit *trapdoor* and satisfying a *statistical simulation* property. Using UC-puzzles, they construct round-optimal protocols for all functionalities. Our work focuses on completeness in terms of *feasibility* rather than the efficiency. To explicitly highlight the uselessness/completeness dichotomy, our completeness criterion is closely tied to the existing notion of splittability. Consequently, our criterion is less demanding (requiring only a *distinguisher*) and is tailored exclusively towards setup functionalities (not more fundamental modifications to the UC framework). We leave it as an open problem whether strong unsplittability can be used to construct a UC puzzle to give an efficiency improvement for our result. In particular, the requirement of a statistical simulation seems incompatible with strong unsplittability, which gives only computational properties.

[2] Our result and that of [24] use different formulations for SFE functionalities. See the discussion in the full version.

In the full version we show that strong unsplittability can be used to understand many previous (*ad hoc*) completeness results involving trusted setups. However, not all setups considered in previous works are within the class of functionalities we consider for the general result here (in particular, many rely crucially on the setup interacting with the adversary).

2 Preliminaries

A function $f : \mathbb{N} \to [0, 1]$ is *negligible* if for all $c > 0$, we have $f(n) < 1/n^c$ for all but finitely many n. A function f is *noticeable* if there exists some $c > 0$ such that $f(n) \geq 1/n^c$ for all but finitely many n. We emphasize that there exist functions that are neither negligible nor noticeable (*e.g.*, $f(n) = n \bmod 2$). A probability $p(n)$ is *overwhelming* if $1 - p(n)$ is negligible.

2.1 Universal Composability

We assume some familiarity with the framework of universally composable (UC) security; for a full treatment, see [5]. We use the notation $\text{EXEC}[\mathcal{Z}, \mathcal{F}, \pi, \mathcal{A}, k]$ to denote the probability that the environment outputs 1, in an interaction involving environment $\mathcal{Z}$, a single instance of an ideal functionality $\mathcal{F}$, parties running protocol π, adversary $\mathcal{A}$, with global security parameter k. All entities in the system must be PPT interactive Turing machines (see [19] for a complete treatment of "polynomial time" definitions for the UC framework). We consider security only against *static* adversaries, who corrupt parties only at the beginning of a protocol execution. π_{dummy} denotes the *dummy protocol* which simply relays messages between the environment and the functionality.

A protocol π is a *UC-secure* protocol for functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if for all adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$, we have that $\left|\text{EXEC}[\mathcal{Z}, \widehat{\mathcal{G}}, \pi, \mathcal{A}, k] - \text{EXEC}[\mathcal{Z}, \mathcal{F}, \pi_{\mathsf{dummy}}, \mathcal{S}, k]\right|$ is negligible in k. Here, $\widehat{\mathcal{G}}$ denotes the multi-session version of $\mathcal{G}$, so that the protocol π is allowed to access multiple instances of $\mathcal{G}$ in the $\mathcal{G}$-hybrid model. The former interaction (involving π and $\mathcal{G}$) is called the **real process**, and the latter (involving π_{dummy} and $\mathcal{F}$) is called the **ideal process**.

We consider a communication network for the parties in which the adversary has control over the timing of message delivery. In particular, there is no guarantee of *fairness* in output delivery.

2.2 Class of Functionalities

Our results apply to essentially the same class of functionalities considered in the feasibility result of Canetti *et al.* [11]. First, the functionality must be *well-formed*, meaning that it ignores its knowledge of which parties are corrupt.

Second, the functionality must be represented as a (uniform) circuit family $\{C_k \mid k \in \mathbb{N}\}$, where C_k describes a single activation of the functionality when the security parameter is k. For simplicity, we assume that C_k receives k bits of

the functionality's internal state, k bits of randomness (independent randomness in each activation), a k-bit input from the activating party, and the identity of the activating party as input, and then outputs a new internal state and k bits of output to each party (including the adversary). Note that all parties receive outputs; in particular, all parties are informed of every activation. We focus on 2-party functionalities and refer to these parties as Alice and Bob throughout.

Finally, we require that the functionality do nothing when activated by the adversary.[3]

2.3 The SHOT Assumption and Required Cryptographic Primitives

The SHOT assumption is that there exists a protocol for $\binom{2}{1}$-oblivious transfer that is secure against semi-honest PPT adversaries (equivalently, standalone-secure, by standard compilation techniques). From the SHOT assumption it also follows that there exist *standalone-secure* protocols for every functionality in the class defined above [15,11]. We require a simulation-based definition of standalone security, equivalent to the restriction of UC security to environments that do not interact with the adversary during the execution of the protocol.

The SHOT assumption implies the existence of one-way functions [20], which in turn imply the existence of standalone-secure statistically-binding commitment schemes [33] and zero-knowledge proofs of knowledge [16,4] that we use in our constructions. One-way functions also imply the existence of **non-malleable secret sharing schemes** (NMSS) [21]. An NMSS consists of two algorithms, Share and $\mathsf{Reconstruct}$. We require that if $(\alpha, \beta) \leftarrow \mathsf{Share}(x)$, then the marginal distributions of α and β are each independent of x, but that $\mathsf{Reconstruct}(\alpha, \beta) = x$. The non-malleability property of the scheme is that, for all x and PPT adversaries $\mathcal{A}$ the following probability is negligible:

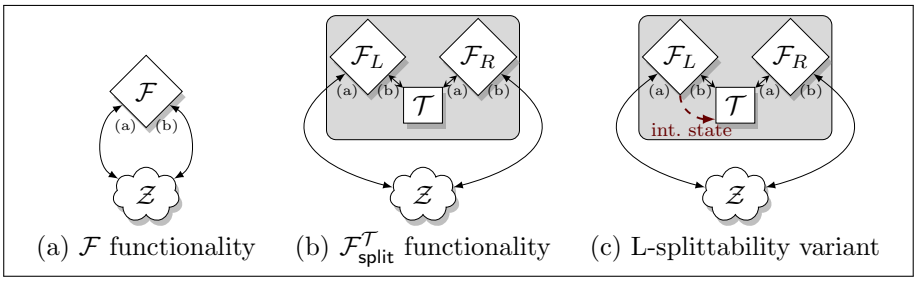
$$\Pr\left[(\alpha, \beta) \leftarrow \mathsf{Share}(x); \beta' \leftarrow \mathcal{A}(\beta, x) : \beta' \neq \beta \wedge \mathsf{Reconstruct}(\alpha, \beta') \neq \bot\right].$$

Furthermore, an NMSS has the property that given α, β, x, and x', where $(\alpha, \beta) \leftarrow \mathsf{Share}(x)$, one can efficiently sample a correctly distributed β' such that $\mathsf{Reconstruct}(\alpha, \beta') = x'$.

3 Splittability and Our Characterization

Our result is based on the alternative characterization of UC-realizability called *splittability*, introduced by Prabhakaran & Rosulek [35]. Intuitively, a two-party functionality $\mathcal{F}$ is splittable if there is a strategy to coordinate two independent

[3] In [11], this convention is also used. However, in the context of a *feasibility* result such as theirs, it is permissible (even desirable) to construct a protocol for a *stronger* version of $\mathcal{F}$ that ignores activations from the adversary. By contrast, in a *completeness* result, we must be able to use the given $\mathcal{F}$ as-is. Since we cannot reason about the behavior of an honest interaction if an external adversary could influence the setup, we make the requirement explicit.



(a) $\mathcal{F}$ functionality (b) $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ functionality (c) L-splittability variant

Fig. 1. Interactions considered in the splittability definitions. Small "a" and "b" differentiate a functionality's communication links for Alice and Bob, respectively.

instances of $\mathcal{F}$, so that together they behave as a single instance of $\mathcal{F}$. More formally, let $\mathcal{T}$ be an interactive Turing machine, and define $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ to be the 2-party functionality which behaves as follows (Figure 1b):

$\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ internally simulates two independent instances of $\mathcal{F}$, denoted $\mathcal{F}_L$ and $\mathcal{F}_R$. $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ associates its input-output link for Alice with the Alice-input/output link of $\mathcal{F}_L$, and similarly the Bob-input/output link with that of $\mathcal{F}_R$. $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ also internally simulates an instance of $\mathcal{T}$, which is connected to the Bob- and adversary-input/output links of $\mathcal{F}_L$ and the Alice- and adversary-input/output links of $\mathcal{F}_R$. $\mathcal{T}$ receives immediate delivery along its communication lines with $\mathcal{F}_L$ and $\mathcal{F}_R$. The $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ functionality does not end its activation until all three subprocesses cease activating. Finally, the instances $\mathcal{T}$, $\mathcal{F}_L$, and $\mathcal{F}_R$ are each given the global security parameter which is provided to $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$. We say that $\mathcal{T}$ is **admissible** if $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ is a valid PPT functionality. For an environment $\mathcal{Z}$, we define

$$\Delta_{\mathsf{split}}(\mathcal{Z}, \mathcal{F}, \mathcal{T}, k) := \left|\text{EXEC}[\mathcal{Z}, \mathcal{F}, \pi_{\mathsf{dummy}}, \mathcal{A}_0, k] - \text{EXEC}[\mathcal{Z}, \mathcal{F}_{\mathsf{split}}^{\mathcal{T}}, \pi_{\mathsf{dummy}}, \mathcal{A}_0, k]\right|,$$

where $\mathcal{A}_0$ denotes the dummy adversary that corrupts no one.

Definition 1 ([35]). *Call an environment* **suitable** *if it does not interact with the adversary except to immediately deliver all outputs from the functionality.*[4] *Then a functionality $\mathcal{F}$ is* **splittable** *if there exists an admissible $\mathcal{T}$ such that for all suitable environments $\mathcal{Z}$, $\Delta_{\mathsf{split}}(\mathcal{Z}, \mathcal{F}, \mathcal{T}, k)$ is negligible in k.*

Splittability provides a complete characterization of *uselessness:*

Theorem 1 ([35]). *Call a functionality* **useless** *if it has a (non-trivial) UC-secure protocol in the plain model. Then $\mathcal{F}$ is useless if and only if $\mathcal{F}$ is splittable.*

[4] The restriction on delivering outputs is analogous to the restriction to so-called "non-trivial protocols" in [9], which is meant to rule out the protocol which does nothing. Similarly, this definition of splittability rules out the trivial splitting strategy $\mathcal{T}$ which does nothing.

3.1 Our Main Theorem

Our classification is based on the following variant of splittability:

Definition 2. *A functionality $\mathcal{F}$ is* **strongly unsplittable** *if there exists a suitable, uniform environment $\mathcal{Z}$ and noticeable function δ such that for all admissible $\mathcal{T}$, $\Delta_{\mathsf{split}}(\mathcal{Z}, \mathcal{F}, \mathcal{T}, k) \geq \delta(k)$.*

Due to technical subtleties (described in Section 4) involving the internal state of functionalities, we also consider the following variants of splittability. If in $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ we let $\mathcal{T}$ obtain the internal state of $\mathcal{F}_L$ (resp. $\mathcal{F}_R$) after every activation (Figure 1c), then we obtain the notions of **L-splittability** and **L-strong-unsplittability** (resp. R-splittability, R-strong-unsplittability). We emphasize that $\mathcal{T}$ is only allowed *read-only access* to the internal state of $\mathcal{F}_L$ (resp. $\mathcal{F}_R$). In fact, most natural functionalities that are strongly unsplittable also appear to be also either L- or R-strongly-unsplittable. For example, the 3 notions are equivalent for *secure function evaluation (SFE)* functionalities — those which evaluate a (possibly randomized) function of the two parties' inputs (definitions and details deferred to the full version). As another example, the ideal commitment functionality $\mathcal{F}_{\mathsf{com}}$ is R-strongly-unsplittable (assuming we identify Alice as the sender), since the sender already knows the entire internal state of $\mathcal{F}_{\mathsf{com}}$ at all times.

Main Theorem. *$\mathcal{F}$ is complete if the* SHOT *assumption is true and either of the following conditions is true:*

1. *$\mathcal{F}$ is L-strongly-unsplittable or R-strongly-unsplittable, or*
2. *$\mathcal{F}$ is strongly unsplittable and L-splittable and R-splittable, and at least one of the $\mathcal{T}$ machines from the L- and R-splittability conditions is* uniform.

Given the equivalence of these notions for SFE functionalities, we have:

Corollary 2 *If the* SHOT *assumption is true, and $\mathcal{F}$ is a strongly unsplittable, SFE functionality, then $\mathcal{F}$ is complete.*[5]

3.2 Interpreting (L-/R-) Splittability & Strong Unsplittability

Nearly all functionalities of interest can be quite easily seen to be either splittable or strongly unsplittable, and thus we informally summarize our results by stating that every "natural" functionality is either useless or complete. However, there are functionalities with intermediate cryptographic power, which are

[5] Though similar, this corollary is somewhat incomparable to the main result of [24]. The two works use fundamentally different formulations of SFE functionalities. In ours (following our convention of Section 2.2) the functionality gives an empty output to both parties after receiving the first party's input. In [24], the functionality gives no output (except to the adversary) until receiving both party's inputs. Our result also applies to randomized functions, whereas the results of [24] involve only deterministic functionalities.

neither splittable or strongly unsplittable. We give concrete examples of such functionalities in the full version, and here briefly describe properties of such functionalities:

First, a functionality's behavior may fluctuate in an unnatural way with respect to the security parameter. This may force every environment's distinguishing probability in the splittability definition to be neither *negligible* (as required for splittability) nor *noticeable* (as required for strong unsplittability). Second, a functionality may have cryptographically useful behavior that can only be accessed by non-uniform computations, while uniform computations (such as a hypothetical commitment protocol using such a functionality) can elicit only trivial behavior. Both of these properties heavily rely on low-level technical restrictions used in the UC framework, and can easily be mitigated by relaxing these restrictions — for example, by considering notions of "infinitely-often useless/complete" or by allowing protocols to be nonuniform machines. We point out that analogous gaps also appear in other contexts involving polynomial-time cryptography [18].

Finally, as in the introduction, interpret the splittability definitions as a 2-party game between $\mathcal{T}$ and $\mathcal{Z}$. Then splittability corresponds to a winning strategy for $\mathcal{T}$ (*i.e.*, a fixed $\mathcal{T}$ which fools all $\mathcal{Z}$), and strong unsplittability corresponds to a winning strategy for $\mathcal{Z}$ (*i.e.*, a fixed $\mathcal{Z}$ which detects all $\mathcal{T}$). Yet some functionalities may not admit winning strategies for either player. (Similarly, there may exist an environment which can distinguish $\mathcal{F}$ from $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ with noticeable probability for every $\mathcal{T}$, but the distinguishing bias may necessarily depend on $\mathcal{T}$.) An example of such a functionality is presented in the full version; however, the functionality is outside the class considered in this work (it cannot be expressed as a circuit family with *a priori* bound on input length). We do not know whether similar behavior is possible within the scope of our results.

4 UC-Equivocal Commitment from R-Strong Unsplittability

In this section we show that any R-strongly unsplittable (symmetrically, L-strongly unsplittable) functionality $\mathcal{F}$ can be used to construct a certain kind of commitment protocol. The resulting protocol has a UC simulation only in the case where the receiver is corrupt (*i.e.*, an equivocating simulator). Its other properties are of the standalone-security flavor. We call such a protocol a *UC-equivocal commitment.* Later in Section 6 we show that such a protocol can be transformed into a fully UC-secure protocol for commitment. From this it follows that $\mathcal{F}$ is complete. The other case of our main theorem is simpler, similar in its techniques, and presented in Section 5.

Simplest instantiation. We first motivate the general design of the protocol with a concrete example. Suppose $\mathcal{F}$ is a functionality which takes input x from Bob and gives $f(x)$ to Alice, where f is a one-way function. Then our UC-equivocal commitment using $\mathcal{F}$ is as follows:

Commit phase; Alice has input b. Alice commits to b using a standalone-secure commitment protocol COM. Let C denote the commitment transcript, and let σ denote the noninteractive decommitment (known only to Alice).

Reveal phase Alice sends b to Bob. Bob chooses a random string x, and sends it to $\mathcal{F}$; Alice receives $y = f(x)$. Both parties engage in a *standalone-secure protocol* for the following functionality, with common input (C, b):

> "On input (σ, z) from Alice, if σ is a valid COM-opening of C to value b, then give output z to Bob; otherwise give output $f(z)$ to Bob."

Alice uses (σ, y) as input to this subprotocol. Bob accepts iff he receives output $f(x)$.

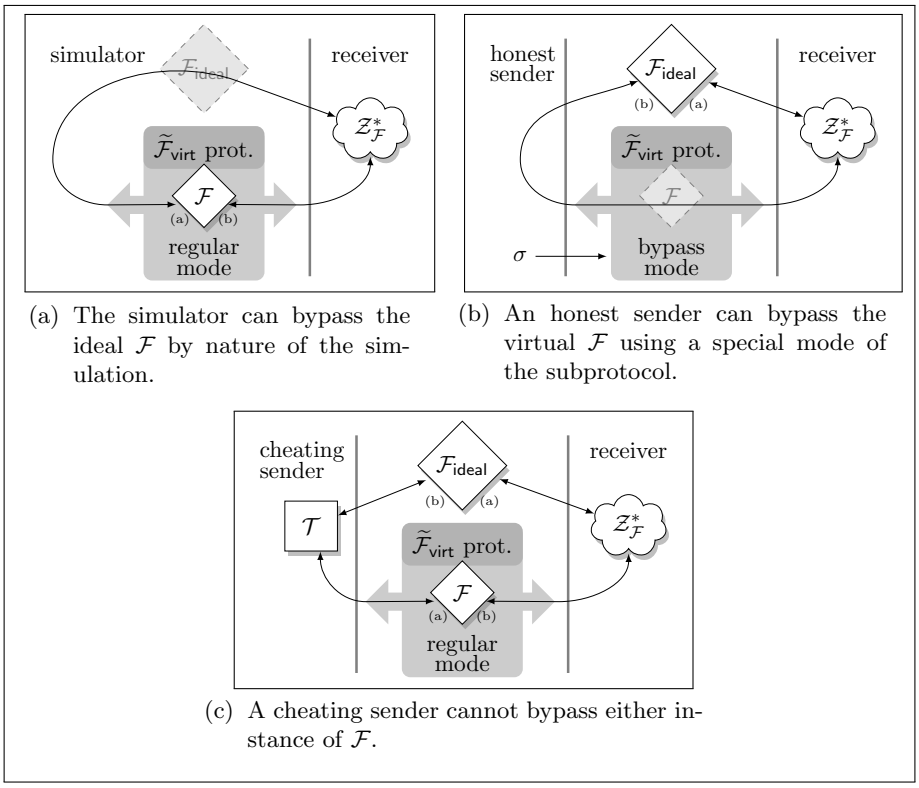
The protocol has a straight-line simulator for a corrupt Bob. The simulator commits to a junk value, then obtains Bob's input x in the reveal phase. As such, it can give $(\perp, x)$ as input to the subprotocol, and Bob will receive output $f(x)$ just as in the real interaction. Note that the subprotocol need only be standalone-secure — the simulator runs the subprotocol honestly, and in particular does not need to rewind the subprotocol.

The protocol is also binding in a standalone-security sense. Intuitively, for an equivocating Alice to succeed, she must provide an input (σ, z) to the subprotocol, where either σ is a valid COM-opening of C to $1-b$, or $y = f(z)$. The former is infeasible by the standalone binding property of COM; the latter is infeasible because it requires Alice to invert the one-way function f.

Connection to splittability. How does this simple protocol relate to the notion of splittability? Observe that Bob's strategy in our protocol is the obvious strategy for the *environment* when showing that $\mathcal{F}$ is strongly unsplittable; namely, choose a random x, send it as Bob, and see whether Alice receives $f(x)$. An honest Alice and the simulator are able to succeed because they effectively "bypass" one of the two applications of f — either the one that happens within $\mathcal{F}$ (by virtue of the simulation) or the one that happens within the subprotocol (by knowing the correct σ value). These interactions are analogous to the environment interacting with a *single* instance (not a split instance) of $\mathcal{F}$ in the splittability game. However, a cheating Alice is "stuck" between two applications of f, analogous to the role of $\mathcal{T}$ in the splittability game.

Generalizing to our final protocol. Following the ideas from the previous paragraph, we generalize the protocol as follows. As before, the commit phase of our final UC-equivocal protocol is essentially just a commitment under a statistically binding, standalone-secure commitment protocol (for technical reasons, it must have a rewinding extracting simulator). Again, the non-trivial part of our protocol is the reveal phase. To be UC-equivocal, the honest sender and the simulator (who can each cause the receiver to accept) must each have some *advantage* over a cheating sender (who should not be able to force the receiver to accept).

Imagine the sender and receiver both connected to two instances of $\mathcal{F}$, in opposite roles (similar to the splittability interaction in Figure 1b). Further imagine



(a) The simulator can bypass the ideal $\mathcal{F}$ by nature of the simulation.

(b) An honest sender can bypass the virtual $\mathcal{F}$ using a special mode of the subprotocol.

(c) A cheating sender cannot bypass either instance of $\mathcal{F}$.

Fig. 2. Intuition behind the reveal phase of $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ and its security properties.

that the receiver is running the code of $\mathcal{Z}^*_{\mathcal{F}}$, the distinguishing environment from the definition of strong unsplittability.

Let us denote one instance of $\mathcal{F}$ (the one associated with $\mathcal{F}_L$) as $\mathcal{F}_{\mathsf{ideal}}$. In our protocol, this instance will indeed be an ideal instance of $\mathcal{F}$, as our protocol is in the $\mathcal{F}$-hybrid model. As such, the simulator for a corrupt receiver is able to *bypass* this instance — by which we mean that the simulator can directly obtain the receiver's inputs and set its outputs on behalf of $\mathcal{F}_{\mathsf{ideal}}$. The simulator then simply "re-routes" the receiver's connection with $\mathcal{F}_{\mathsf{ideal}}$ directly into the second instance of $\mathcal{F}$. Thus, from the receiver's point of view, $\mathcal{Z}^*_{\mathcal{F}}$ appears to be interacting with only a *single* instance of $\mathcal{F}$ (Figure 2a).

An honest sender's only advantage is that the underlying standalone commitment can indeed be opened to the value being claimed, whereas a cheating sender cannot generate a valid decommitment to the false value it claims. We would like to translate this advantage into a similar "bypassing" capability as the simulator. Let us denote the other instance of $\mathcal{F}$ (the one associated with $\mathcal{F}_R$) as $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ (virtual-$\mathcal{F}$), and modify it as follows. It now takes as input the commitment-phase transcript C and the purported value b. It also takes as input a candidate decommitment string σ from the sender. If σ is indeed a valid decommitment of C to b, then $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ allows the sender to *bypass* just as above (directly giving

the receiver's inputs to the sender and allowing the sender to directly fix the receiver's outputs). Otherwise, the functionality simply acts exactly like $\mathcal{F}$. Now the honest sender can bypass $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ so that, again, from the receiver's point of view, $\mathcal{Z}^*_{\mathcal{F}}$ is interacting with a single instance of $\mathcal{F}$ (Figure 2b). This advantage for the honest sender holds even if we have just a **standalone-secure** protocol for $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ (importantly, since constructing a *UC-secure protocol* for $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ in the $\mathcal{F}$-hybrid model might even be harder than our goal of constructing UC-secure commitment in the $\mathcal{F}$-hybrid model).

Finally, a cheating (equivocating) sender cannot provide such a value σ to $\widetilde{\mathcal{F}}_{\mathsf{virt}}$, so $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ behaves just like an instance of $\mathcal{F}$. Thus the cheating sender can bypass neither instance and is "stuck" between two instances of $\mathcal{F}$ (Figure 2c). The receiver's environment $\mathcal{Z}^*_{\mathcal{F}}$ is specifically designed to detect this difference, no matter what the cheating sender does. The distinguishing bias of $\mathcal{Z}^*_{\mathcal{F}}$ is guaranteed to be noticeable, so by repeating this basic interaction a polynomial number of times $\mathcal{Z}^*_{\mathcal{F}}$ can distinguish with overwhelming probability. The receiver will therefore accept the decommitment if $\mathcal{Z}^*_{\mathcal{F}}$ believes it is interacting with instances of $\mathcal{F}$ rather than instances of $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$.

Technical subtleties. We outline some important technical considerations that affect the final design of our UC-equivocal commitment protocol. First, our $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ subprotocol only has standalone security, so to apply any of its properties may require using a rewinding simulation. If the $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ subprotocol is ongoing while the parties interact with $\mathcal{F}_{\mathsf{ideal}}$, or while the receiver is executing its $\mathcal{Z}^*_{\mathcal{F}}$ instance, then these instances may also be rewound. Since rewinding $\mathcal{Z}^*_{\mathcal{F}}$ and $\mathcal{F}_{\mathsf{ideal}}$ would jeopardize our ability to apply the splittability condition, we let our subprotocol only perform a *single activation* of the virtual $\mathcal{F}$ per execution. We allow $\mathcal{F}$ to be reactive, so we need a way to maintain the internal state of the virtual-$\mathcal{F}$ between activations of $\widetilde{\mathcal{F}}_{\mathsf{virt}}$. For this purpose we have the $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ subprotocol share $\mathcal{F}$'s internal state between the two parties using a non-malleable secret sharing (NMSS) scheme, which was proposed for precisely this purpose [21].

The other important technicality is that activations of the $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ subprotocol and of $\mathcal{F}_{\mathsf{ideal}}$ are decoupled, meaning that the receiver can observe the relative timings of these activations. This differs from the splittability interaction, in which successive activations of $\mathcal{F}_L$ and $\mathcal{F}_R$ within $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ are atomic from the environment's perspective. This difference makes the "bypassing" technique more subtle. For instance, when the receiver gives an input to the $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ subprotocol, the sender must obtain this input, then make a "round trip" to $\mathcal{F}_{\mathsf{ideal}}$ to obtain the output that it forces to the receiver through the $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ subprotocol. For this reason, and to avoid having the subprotocol running while $\mathcal{F}_{\mathsf{ideal}}$ is accessed, we split such an activation (initiated by the receiver activating the virtual $\mathcal{F}$) of the virtual $\mathcal{F}$ into two phases: one for input-extraction and one for output-fixing.

Similarly, consider the case where the simulator is bypassing $\mathcal{F}_{\mathsf{ideal}}$. In the real-process interaction, every activation of $\mathcal{F}_{\mathsf{ideal}}$ is instantaneous, so the simulator must make such activations appear instantaneous as well. In particular, the simulator has no time to make a "round-trip" to the $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ subprotocol to

determine the appropriate response to simulate on behalf $\mathcal{F}_{\mathsf{ideal}}$. A moment's reflection shows that the only way for the simulator to *immediately* give the correct response is if it already knows the internal state of the virtual-$\mathcal{F}$. For this reason, we let the subprotocol give this information to the sender, even in its normal (non-bypassing) mode. Since this internal state information then becomes available to a cheating sender as well, we require that $\mathcal{F}$ be strongly unsplittable even when $\mathcal{F}_R$ leaks its internal state in this way (*i.e.*, R-strongly unsplittable).

We use only indistinguishability properties of the $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ subprotocol to prove the soundness of the straight-line equivocating simulator. Indeed, the simulation is valid even against arbitrary corrupt receivers, even though for simplicity we have portrayed here all receivers to be running $\mathcal{Z}^*_{\mathcal{F}}$ as the protocol prescribes. We use the splittability condition to prove only the (standalone) binding property of the commitment, where the receiver is honest and indeed runs $\mathcal{Z}^*_{\mathcal{F}}$.

Lemma 1. *If the* SHOT *assumption is true, then $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ has a UC simulator in the case that the receiver is corrupt (i.e., an equivocating simulator). If $\mathcal{F}$ is R-strongly-unsplittable, then $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ has a rewinding simulator in the case that the sender is corrupt (i.e., an extracting simulator).*

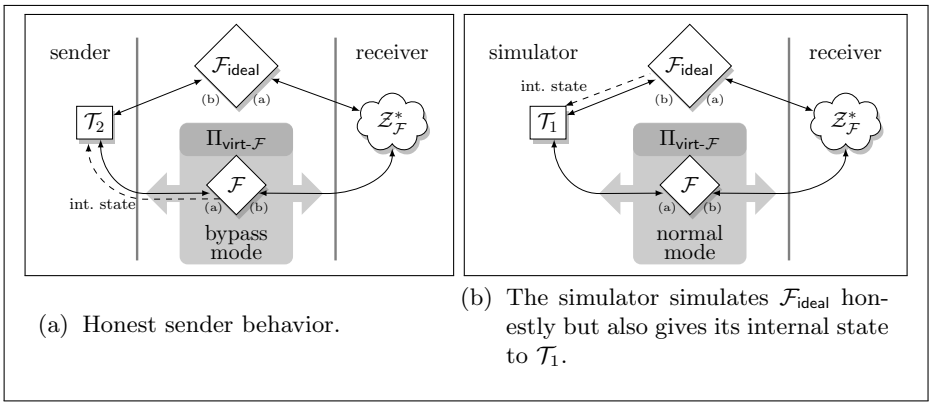
5 UC-Equivocal Commitment from L/R-Splittability

In this section we show that if $\mathcal{F}$ is strongly unsplittable, L-splittable, and R-splittable, then $\mathcal{F}$ can be used to construct a UC-equivocal commitment protocol. This is the second case in our main theorem.

5.1 Overview

Our approach for this case is quite similar to that of Section 4 — our protocol involves an ideal instance of $\mathcal{F}$ along with a "virtual" instance of $\mathcal{F}$ within a standalone-secure subprotocol. The receiver runs instances of $\mathcal{Z}^*_{\mathcal{F}}$, the environment guaranteed by the strong unsplittability condition. The receiver accepts the commitment if $\mathcal{Z}^*_{\mathcal{F}}$ believes it is interacting with a single instance of $\mathcal{F}$ as opposed to some $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$. The primary difference from the previous section is in the "bypass mode" of the virtual-$\mathcal{F}$ subprotocol. In this subprotocol, the bypass mode does not completely bypass the virtual $\mathcal{F}$, but instead it simply leaks the internal state of the virtual $\mathcal{F}$ to the receiver. Intuitively, leaking the internal state is sufficient to "fool" $\mathcal{Z}^*_{\mathcal{F}}$, since $\mathcal{F}$ is L/R-splittable. We have the following observations about the protocol (Figure 3):

- An honest sender who can activate the bypass mode of the $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ subprotocol is situated between an ideal instance of $\mathcal{F}$ and (intuitively) an instance of $\mathcal{F}$ that leaks its internal state. By the R-splittability of $\mathcal{F}$, there exists a $\mathcal{T}_2$ that the sender can execute to make two such instances behave to the sender as a single instance of $\mathcal{F}$. Note that $\mathcal{T}_2$ must be a uniform machine, since it is used as a subroutine in the description of the protocol.



(a) Honest sender behavior.

(b) The simulator simulates $\mathcal{F}_{\mathsf{ideal}}$ honestly but also gives its internal state to $\mathcal{T}_1$.

Fig. 3. Interactions in $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ and its security proof.

- The simulator can honestly simulate $\mathcal{F}_{\mathsf{ideal}}$ while also having access to its internal state. The remainder of the simulator is intuitively between this simulated instance of $\mathcal{F}$ and a (normal, non-bypassed) instance of $\mathcal{F}$. By the L-splittability of $\mathcal{F}$, there exists a $\mathcal{T}_1$ that the simulator can execute to make these two instances behave to the sender as a single instance of $\mathcal{F}$. Note that $\mathcal{T}_1$ need not be uniform, since it is used only by the simulator. Finally, since the simulator is designed to interact with a corrupt receiver, $\mathcal{T}_1$ and $\mathcal{T}_2$ must be able to "fool" every environment, not just the environment $\mathcal{Z}^*_{\mathcal{F}}$ from the strong unsplittability condition.
- A cheating sender cannot obtain the internal state from either the ideal or the virtual instance of $\mathcal{F}$. As such, it plays the role of the machine $\mathcal{T}$ in the (normal) splittability interaction. By the strong unsplittability of $\mathcal{F}$, the receiver's environment $\mathcal{Z}^*_{\mathcal{F}}$ can detect a noticeable deviation from the expected behavior.

In this section, we never have need to completely bypass an instance of $\mathcal{F}$. The technical complications described in Section 4 are not present here, and the actual construction is a much more straight-forward implementation of the intuition described above.

Lemma 2. *If the* SHOT *assumption is true and $\mathcal{F}$ is L- & R-splittable, then $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ has a UC simulator in the case that the receiver is corrupt (i.e., an equivocating simulator). If $\mathcal{F}$ is strongly-unsplittable, then $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ has a rewinding simulator in the case that the sender is corrupt (i.e., an extracting simulator).*

6 Full-Fledged UC Commitment from Equivocal Commitment

We now show how the UC-equivocal commitment protocol from the previous sections can be used to construct a full-fledged UC commitment protocol. Due to space limitations, we give only a high-level overview of the protocol $\Pi^{\mathcal{F}}_{\mathsf{com}}$ here; the full details and security proof are deferred to the full version.

Commit phase:

1. The receiver first uses $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ to commit to a random string r containing half 0s and half 1s.
2. To commit to a bit b, the sender uses $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ to commit (bitwise) to a random string x, and gives $b \oplus (\bigoplus_i x_i)$ to the receiver.
3. The two parties engage in a standalone-secure subprotocol in which the receiver learns a random half of the bits of x. The sender does not learn which positions of x the receiver obtains.
4. The receiver opens his commitment to r. The receiver opens her commitments to all bits x_i such that $r_i = 1$.

Intuitively, a malicious receiver can expect to learn no more than about 3/4 of the bits of x in this way (a random half in step 3 and an independent random half in step 4), so the secret bit b remains hidden. However, the simulator for a corrupt sender can equivocate (in $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$) while revealing r so that it learns *all* the bits of x, and hence can extract b.

Reveal phase: The parties engage in a standalone-secure subprotocol in which the sender learns a position i^* for which the receiver did not learn the bit x_{i^*} in steps 3 and 4 the commit phase. Then the sender opens all the remaining commitments to the bits of x.

Intuitively the information i^* is useless to a corrupt sender, but the simulator for a corrupt receiver can equivocate (in $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$) on its opening of the bit x_{i^*}, and hence open its commitment to either value of b.

Lemma 3. *$\Pi^{\mathcal{F}}_{\mathsf{com}}$ is a UC-secure protocol for commitment, in the $\mathcal{F}$-hybrid model.*

7 Necessity of SHOT Assumption & Strong Unsplittability

The SHOT assumption is necessary for many strongly unsplittable functionalities (*e.g.*, coin-tossing, commitment) to be complete under static corruption [14,31]. Thus the SHOT assumption is the minimal assumption for a completeness result such as ours.

In this work we showed that strong unsplittability (and its variants) is a sufficient condition for completeness. Ideally, we would like to prove that it is also a necessary condition; currently we are able to prove that several minor modifications of strong unsplittability are necessary.

To prove necessity of some kind of strong unsplittability, we show that $\mathcal{F}$ is not complete by showing that there is no UC-secure protocol for coin-tossing in the $\mathcal{F}$-hybrid model. Consider an interaction involving a purported coin-tossing protocol in the $\mathcal{F}$-hybrid model. This protocol invokes possibly many instances of $\mathcal{F}$, and it is likely that we will have to apply some property of $\mathcal{F}$ to each of them (indeed, this is what we must do in the proofs below). If $\mathcal{F}$ is not strongly unsplittable, then for every suitable $\mathcal{Z}$, there is a machine $\mathcal{T}$ so that $\Delta_{\mathsf{split}}(\mathcal{Z}, \mathcal{F}, \mathcal{T}, k)$ is non-noticeable — that is, negligible only *for infinitely many*

values of the security parameter. In the following proofs, we must apply this condition in a hybrid argument, each time with a slightly different environment and thus a potentially different subset of security parameter values. There may not be an infinite number of security parameter values for which *every* step of the hybrid argument succeeds, and thus we must settle for slight variants of strong unsplittability in the following:

Lemma 4. *If $\mathcal{F}$ is complete, then all of the following are true:*

1. *$\mathcal{F}$ is infinitely-often strongly unsplittable with respect to* uniform *$\mathcal{T}$.*
2. *The multi-session version of $\mathcal{F}$ is strongly unsplittable.*
3. *$\mathcal{F}$ is strongly unsplittable via an environment with multi-bit output.*

In item (1), "infinitely-often" refers to the relaxation in which $\Delta_{\mathsf{split}}(\mathcal{Z}, \mathcal{F}, \mathcal{T}, k)$ is non-negligible (rather than noticeable, as required in the usual definition).

The complete proofs are given in the full version, but for a flavor of the techniques we give a proof of item (1) here:

Proof (of item 1). We prove the contrapositive. Suppose that $\mathcal{F}$ is **not** as in item (1), then for every suitable $\mathcal{Z}$ there is a *uniform* machine $\mathcal{T}$ so that $\Delta_{\mathsf{split}}(\mathcal{Z}, \mathcal{F}, \mathcal{T}, k)$ is negligible. It suffices to show that there is no protocol for coin-tossing in the $\mathcal{F}$-hybrid model.

Suppose for contradiction that π is such a secure protocol for coin-tossing. Let $\mathcal{Z}$ be the environment that invokes one session of coin tossing and outputs 1 if both parties output the same coin. Then we inductively define sequences of machines $\mathcal{T}_i$ and $\mathcal{Z}_i$ as follows. $\mathcal{Z}_i$ is the environment that internally simulates $\mathcal{Z}$ and two honest parties running π. For $j > i$, the jth instance of $\mathcal{F}$ invoked by π is simulated internally to $\mathcal{Z}_i$. The ith instance of $\mathcal{F}$ is routed to an external ideal instance of $\mathcal{F}$. And for $j < i$, the jth instance of $\mathcal{F}$ is simulated internally as $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}_j}$ rather than $\mathcal{F}$. The machine $\mathcal{T}_i$ is defined as the uniform machine that "fools" environment $\mathcal{Z}_i$. By a straight-forward hybrid argument, we have that $\mathcal{Z}$ outputs 1 with overwhelming probability when invoking an instance of π in which the ith instance of $\mathcal{F}$ is replaced by $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}_i}$.

By applying the UC-security of π, we can replace Alice (who is honestly running π) and the collection of $\mathcal{F}_L$ instances in this interaction with an ideal instance of coin-tossing (and appropriate simulator). Similarly, we can replace Bob and the set of $\mathcal{F}_R$ instances in the interaction with another ideal instance of coin-tossing. Both of these changes have only a negligible effect on the outcome of the interaction, by the security of π. Now $\mathcal{Z}$ receives two coins from *totally independent* instances of an ideal coin-tossing functionality, and outputs 1 with overwhelming probability. Since $\mathcal{Z}$ only outputs 1 if its two inputs agree, this is a contradiction. Thus, no such protocol π can exist.

Acknowledgements

The author would like to thank Tal Malkin for helpful discussions surrounding Lemma 4, and Manoj Prabhakaran, Hong-Sheng Zhou, and several anonymous referees for various constructive suggestions.

References

1. B. Barak, R. Canetti, J. B. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195. IEEE Computer Society, 2004.
2. B. Barak and A. Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *FOCS*, pages 543–552. IEEE Computer Society, 2005.
3. A. Beimel, T. Malkin, and S. Micali. The all-or-nothing nature of two-party secure computation. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 1999.
4. M. Bellare and O. Goldreich. On defining proofs of knowledge. In E. F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer, 1993.
5. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In M. Naor, editor, *FOCS*, pages 136–145. IEEE Computer Society, 2001. Revised version (2005) on Cryptology ePrint Archive: http://eprint.iacr.org/2000/067.
6. R. Canetti. Obtaining universally composable security: Towards the bare bones of trust. In K. Kurosawa, editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 88–112. Springer, 2007.
7. R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In S. P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.
8. R. Canetti and M. Fischlin. Universally composable commitments. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.
9. R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.
10. R. Canetti, H. Lin, and R. Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In L. Trevisan, editor, *FOCS*, pages 541–550. IEEE Computer Society, 2010.
11. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503. ACM, 2002.
12. R. Canetti, R. Pass, and a. shelat. Cryptography from sunspots: How to use an imperfect reference string. In *FOCS*, pages 249–259. IEEE Computer Society, 2007.
13. B. Chor and E. Kushilevitz. A zero-one law for boolean privacy. *SIAM J. Discrete Math.*, 4(1):36–47, 1991.
14. I. Damgård, J. B. Nielsen, and C. Orlandi. On the necessary and sufficient assumptions for UC computation. In D. Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2010.
15. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.
16. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304. ACM, 1985.
17. J. Groth and R. Ostrovsky. Cryptography in the multi-string model. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 323–341. Springer, 2007.

18. D. Harnik, M. Naor, O. Reingold, and A. Rosen. Completeness in two-party secure computation: A computational view. *J. Cryptology*, 19(4):521–552, 2006.
19. D. Hofheinz, D. Unruh, and J. Müller-Quade. Polynomial runtime and composability. Cryptology ePrint Archive, Report 2009/023, 2009. http://eprint.iacr.org/2009/023.
20. R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography (extended abstract). In *FOCS*, pages 230–235. IEEE, 1989.
21. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In D. Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591. Springer, 2008.
22. Y. T. Kalai, Y. Lindell, and M. Prabhakaran. Concurrent general composition of secure protocols in the timing model. In H. N. Gabow and R. Fagin, editors, *STOC*, pages 644–653. ACM, 2005.
23. J. Katz. Universally composable multi-party computation using tamper-proof hardware. In M. Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2007.
24. J. Katz, A. Kiayias, R. Kumaresan, a. shelat, and H.-S. Zhou. From impossibility to completeness for deterministic two-party SFE. Unpublished manuscript, 2011.
25. D. Kidron and Y. Lindell. Impossibility results for universal composability in public-key models and with fixed inputs. *J. Cryptology*, 24(3):517–544, 2011.
26. J. Kilian. More general completeness theorems for secure two-party computation. In *STOC*, pages 316–324. ACM, 2000.
27. J. Kilian, E. Kushilevitz, S. Micali, and R. Ostrovsky. Reducibility and completeness in private computations. *SIAM J. Comput.*, 29(4):1189–1208, 2000.
28. G. Kreitz. A zero-one law for secure multi-party computation with ternary outputs. In Y. Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 382–399. Springer, 2011.
29. H. Lin, R. Pass, and M. Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In M. Mitzenmacher, editor, *STOC*, pages 179–188. ACM, 2009.
30. Y. Lindell. Lower bounds for concurrent self composition. In M. Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 203–222. Springer, 2004.
31. H. K. Maji, M. Prabhakaran, and M. Rosulek. A zero-one law for cryptographic complexity with respect to computational UC security. In T. Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 595–612. Springer, 2010.
32. T. Malkin, R. Moriarty, and N. Yakovenko. Generalized environmental security from number theoretic assumptions. In S. Halevi and T. Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 343–359. Springer, 2006.
33. M. Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.
34. R. Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 160–176. Springer, 2003.
35. M. Prabhakaran and M. Rosulek. Cryptographic complexity of multi-party computation problems: Classifications and separations. In D. Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 262–279. Springer, 2008.
36. M. Prabhakaran and A. Sahai. New notions of security: achieving universal composability without trusted setup. In L. Babai, editor, *STOC*, pages 242–251. ACM, 2004.