Universal Composability from Essentially Any Trusted Setup*

Mike Rosulek†

May 31, 2012

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).

*An extended abstract of this work appeared in *CRYPTO 2012*.

†Department of Computer Science, University of Montana. mikero@cs.umt.edu. Supported by NSF grant CCF-1149647.

Contents

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 [C01] 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 [CF01, L04, CKL06] culminated in a complete characterization for which tasks are securely realizable in the UC framework [PR08]. 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 [CF01, CLOS02, BCNP04, CPS07, GO07, K07, IPS08, MPR10], allowing superpolynomial simulation [P03, PS04, BS05, MMY06, CLP10], assuming bounded network latency [KLP05], considering uniform adversaries [LPV09], and including certain global setups [CDPW07], to name a few (for a more comprehensive treatment, see the survey by Canetti [C07]). Other variants of the UC framework turn out to be no more powerful than the original framework; for example, adding certain setup functionalities [PR08, KL11] or global setups [CDPW07], and requiring only self-composition rather than universal composition [L04]. 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 [PR08].

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 [PR08]. 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 [PR08] 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.*

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 [PR08], 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 [CLOS02].

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.* [CLOS02] 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] 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.

1.2 Related Work

Dichotomies in the cryptographic power of functionalities are known in several settings [CK91, K11]. 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 [BMM99, K00, KKMO00, HNRR06, MPR10, KKK+11]. Among these results, only [MPR10] considered *reactive* functionalities, and only [K00] 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 [MPR10, KKK+11] are cast in the UC framework; for these we give a more detailed comparison to our own results. Maji, Prabhakaran, and Rosulek [MPR10] 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.* [KKK+11] showed a similar result for deterministic, non-reactive (secure function evaluation) functionalities.[2] They construct UC-puzzles [LPV09] for classes of SFE functionalities deemed impossible in the characterization of Canetti, Kushilevitz, and Lindell [CKL06]. Our result achieves greater generality by being based not on the CKL characterization but the splittability characterization of [PR08]. Furthermore, we show completeness by directly constructing a commitment protocol rather than a UC puzzle (see below).

Lin, Pass, and Venkitasubramaniam [LPV09] 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.

In Appendix C 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] Our result and that of [KKK+11] use different formulations for SFE functionalities. See the discussion in Appendix B.

2.1 Universal Composability

We assume some familiarity with the framework of universally composable (UC) security; for a full treatment, see [C01]. 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 [HUM09] 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.* [CLOS02]. 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.

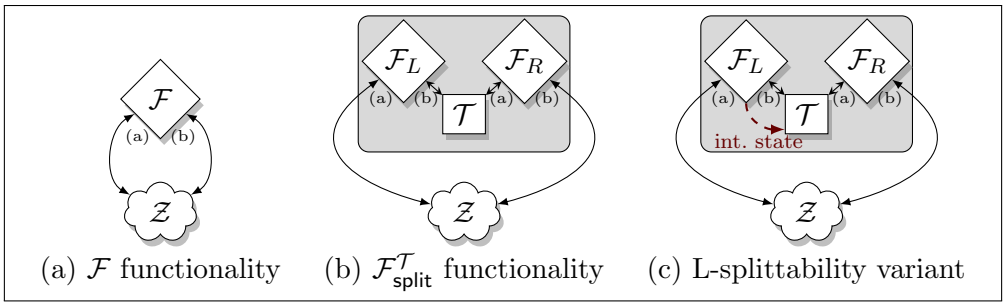
Finally, we require that the functionality do nothing when activated by the adversary.[3]

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 [GMW87, CLOS02]. 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 [IL89], which in turn imply the existence of standalone-secure statistically-binding commitment schemes [N91] and zero-knowledge

[3] In [CLOS02], 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) $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ functionality (c) L-splittability variant

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

proofs of knowledge [GMR85, BG93] that we use in our constructions. One-way functions also imply the existence of **non-malleable secret sharing schemes** (NMSS) [IPS08]. 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 [PR08]. Intuitively, a two-party functionality $\mathcal{F}$ is splittable if there is a strategy to coordinate two independent 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 $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ to be the 2-party functionality which behaves as follows (Figure 1b):

$\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ internally simulates two independent instances of $\mathcal{F}$, denoted $\mathcal{F}_L$ and $\mathcal{F}_R$. $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ 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 $\mathcal{F}_R$. $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ 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 $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ 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 $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$. We say that $\mathcal{T}$ is **admissible** if $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ is a valid PPT functionality. For an environment $\mathcal{Z}$, we define

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

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

Definition 3.1 ([PR08])**.** *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* **split-**

[4]The restriction on delivering outputs is analogous to the restriction to so-called "non-trivial protocols" in [CKL06], 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.

table *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 3.2 ([PR08])**.** *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.*

3.1 Our Main Theorem

Our classification is based on the following variant of splittability:

Definition 3.3. *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 $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ 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 (see Appendix B for full definitions and details). 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.[5]

Given the equivalence of these notions for SFE functionalities (Appendix B), we have:

Corollary 3.4. *If the* SHOT *assumption is true, and* $\mathcal{F}$ *is a strongly unsplittable, SFE functionality, then* $\mathcal{F}$ *is complete.*[6]

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

[5] One of these machines is used as a subroutine in a protocol; the other $\mathcal{T}$ machine is only used by the simulator. We can allow both $\mathcal{T}$ machines to be non-uniform if we permit UC protocols to be non-uniform.

[6] Though similar, this corollary is somewhat incomparable to the main result of [KKK+11]. 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 [KKK+11], 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 [KKK+11] involve only deterministic 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 [HNRR06].

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 Appendix A; 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.

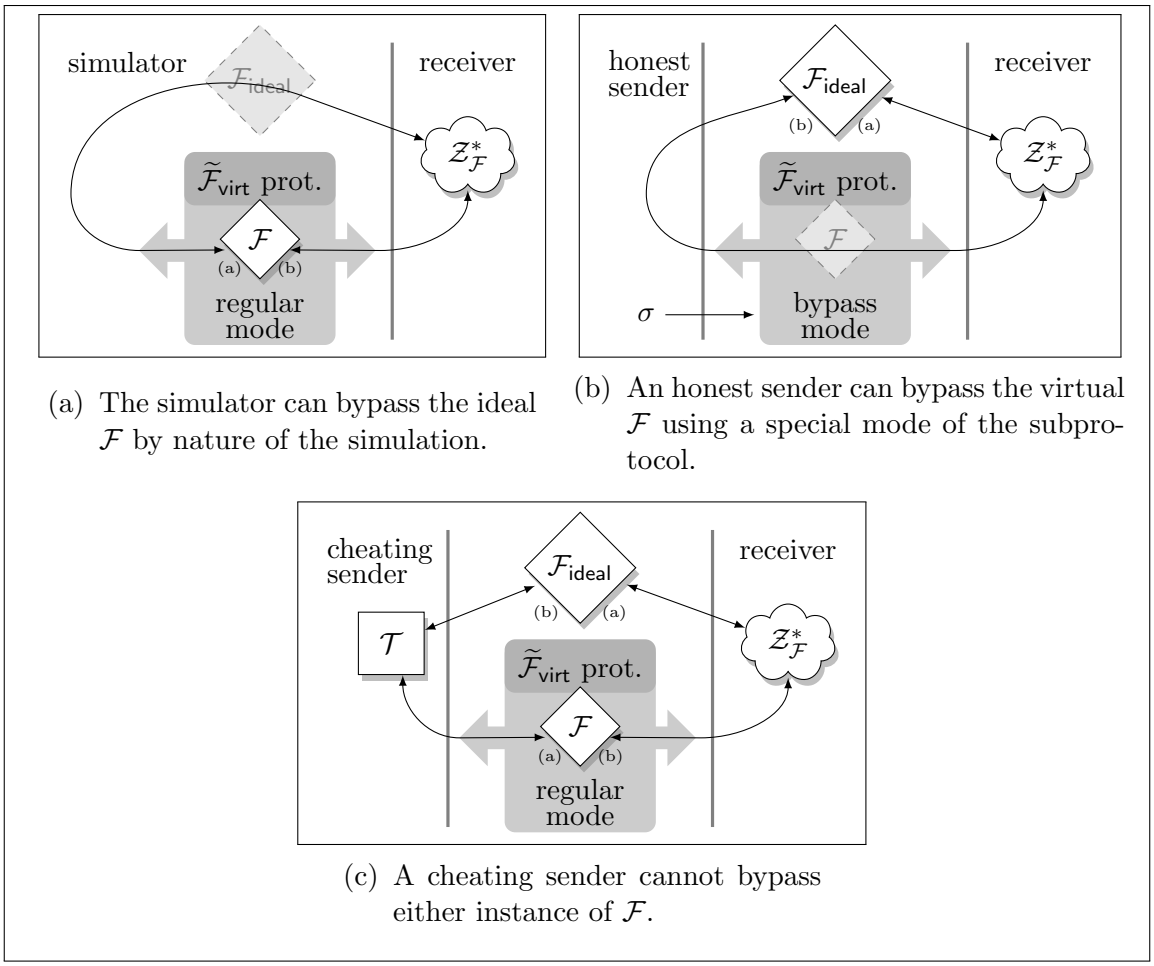
Connection to splittability. How does this simple protocol relate to the notion of splittability? Observe that Bob's strategy in our protocol is the obvious strategy for the *environment* when showing that $\mathcal{F}$ is strongly unsplittable; namely, choose a random x, send it as Bob, and see whether 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 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*



(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 $\mathcal{F}$.

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

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 [IPS08].

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 $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ 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 $\mathcal{Z}_{\mathcal{F}}^*$.

4.1 The Virtual-$\mathcal{F}$ Subprotocol

We specify the behavior of our virtual-$\mathcal{F}$ subprotocol in the form of an ideal functionality:

The functionality $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ for simulating activations of $\mathcal{F}$, with bypass mode. Let $\mathsf{NMSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ be a non-malleable secret sharing scheme that can support messages of length $2k$. For notational simplicity, we suppose that in NMSS the pair (ϵ, ϵ) constitutes a valid secret share of the value 0^{2k}. We also assume that 0^k is the initial internal state of $\mathcal{F}$. Let COM be a standalone-secure, plain-model commitment protocol with non-interactive opening phase. Then $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ is the non-reactive, 2-party ideal functionality defined as follows, with global security parameter k:

1. $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ waits for an input of the form (x, q, σ, S_1) from the **sender**, an input of the form (y, S_2) from the **receiver**, and a common input of the form (a, b, C).[7]

2. If $\mathsf{Reconstruct}(S_1, S_2) = \perp$, then output $\perp$ to both parties and halt. Otherwise let $S \| \hat{y} = \mathsf{Reconstruct}(S_1, S_2)$.

3. If σ is **not** a valid opening of transcript C to the value b (in COM), then we say that the functionality is in **virtual-$\mathcal{F}$ mode.**

[7] By "common input" we simply mean a value that is public and agreed-upon by both sender and receiver. One may imagine both parties submitting these values, and the functionality aborting if the two sets of values disagree.

(a) If $a = $ B-IN, simulate an activation of $\mathcal{F}$ with internal state S, and input y from Bob ($\hat{y}$ is ignored). Suppose this activation ends with $\mathcal{F}$ delivering output p to Alice (at this point, we ignore the output to Bob and the new internal state). Then let $(S'_1, S'_2) \leftarrow \mathsf{Share}(S \| y)$ and give output (p, S'_1, S) to the sender and output S'_2 to the receiver.

(b) If $a = $ B-OUT, then simulate an activation of $\mathcal{F}$ with internal state S and input $\hat{y}$ from Bob (y is ignored). If $a = $ A-IN, then simulate an activation of $\mathcal{F}$ with internal state S and input x from Alice. Suppose this activation ends with $\mathcal{F}$ in internal state S' and generating output q for Bob (we ignore the output for Alice). Then let $(S'_1, S'_2) \leftarrow \mathsf{Share}(S' \| 0^k)$ and give output (S'_1, S') to the sender and output (q, S'_2) to the receiver.

4. If σ is a valid opening of C to the value b (in COM), then we say that the functionality is in **bypass mode.**

(a) If $a = $ B-IN, then let $(S'_1, S'_2) \leftarrow \mathsf{Share}(0^{2k})$. Give output (y, S'_1, S'_2) to the sender and output S'_2 to the receiver.

(b) If $a \in \{$B-OUT, A-IN$\}$ then let $(S'_1, S'_2) \leftarrow \mathsf{Share}(0^{2k})$. Give output (S'_1, S'_2) to the sender and output (q, S'_2) to the receiver.

Standalone-secure protocol. Under the SHOT assumption, there exists a standalone-secure protocol $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ for $\widetilde{\mathcal{F}}_{\mathsf{virt}}$.

4.2 UC-Equivocal Commitment

Let COM be a statistically-binding, standalone-secure commitment protocol with non-interactive opening phase. Let $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ be as above, with respect to the same COM protocol. Let $\mathcal{Z}^*_{\mathcal{F}}$ be the environment guaranteed by the R-strong-unsplittability of $\mathcal{F}$, which outputs 1 with probability at least $\frac{1}{2} + \delta(k)$ when interacting with an instance of $\mathcal{F}$, and with probability at most $\frac{1}{2} - \delta(k)$ when interacting with any appropriate instance of $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$. The function δ is guaranteed to be a noticeable function. The protocol $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ proceeds as follows, with security parameter k:

Commit phase: When the sender receives the command (COMMIT, b):

1. The sender commits to b under the COM scheme. Let C be the resulting transcript, and let σ be the non-interactive opening of C to b.

2. The sender uses a (standalone-secure) zero-knowledge proof of knowledge to prove knowledge of (σ, b) such that σ is a valid opening of C to b.

Reveal phase: Both parties are connected to an ideal instance of the multi-session version of $\mathcal{F}$, which for clarity we denote $\mathcal{F}_{\mathsf{ideal}}$. The sender is connected to $\mathcal{F}_{\mathsf{ideal}}$ in the role of Bob, and the receiver is connected in the role of Alice. When the sender receives the command REVEAL, the parties continue as follows:

1. The sender gives b to the receiver.

2. The parties do the following, $N(k) = O(k/\delta(k)^2)$ times, each with a new session of $\mathcal{F}_{\mathsf{ideal}}$:

(a) The sender initializes local value $S_1 := \epsilon$, and the receiver initializes local value $S_2 := \epsilon$. (By our convention, these are valid NMSS shares of 0^{2k}, and 0^k is the initial internal state of $\mathcal{F}$.)

(b) [IdealActivation **sequence**] Whenever $\mathcal{Z}_{\mathcal{F}}^*$ generates a command x to send to Alice, the parties do the following (see Figure 3a):

i. The receiver sends x to $\mathcal{F}_{\mathsf{ideal}}$. Say the activation of $\mathcal{F}_{\mathsf{ideal}}$ ends with output q for the sender and output p for the receiver.

ii. The sender provides input $(\perp, q, \sigma, S_1)$ and the receiver provides input $(\perp, S_2)$ to a fresh instance of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$, with common input $(\textsc{a-in}, b, C)$. If the instance aborts or outputs $\perp$, or if $\mathcal{F}_{\mathsf{ideal}}$ is activated during the execution of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$, then both parties abort.

iii. The $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ instance eventually terminates with output (S_1', S_2') for the sender and (q, S_2') for the receiver. The receiver delivers q to $\mathcal{Z}_{\mathcal{F}}^*$ (on its output tape from Bob) and p to $\mathcal{Z}_{\mathcal{F}}^*$ (on its output tape from Alice). Both parties update $S_1 := S_1'$ or $S_2 := S_2'$, appropriately.

(c) [VirtActivation **sequence**] Whenever $\mathcal{Z}_{\mathcal{F}}^*$ generates a command y to send to Bob, the receiver notifies the sender to initiate an instance of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ as follows (see Figure 3c):

i. The sender provides input $(\perp, \perp, \sigma, S_1)$ and the receiver provides input (y, S_2) to a fresh instance of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$, with common input $(\textsc{b-in}, b, C)$. If the instance aborts or outputs $\perp$, or if $\mathcal{F}_{\mathsf{ideal}}$ is activated during the execution of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$, then both parties abort.

ii. The $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ instance eventually terminates with output (y, S_1', S_2') for the sender and output S_2' to the receiver. Both parties update $S_1 := S_1'$ or $S_2 := S_2'$, appropriately.

iii. The sender gives y as input to $\mathcal{F}_{\mathsf{ideal}}$. Say the activation of $\mathcal{F}_{\mathsf{ideal}}$ ends with output q for the sender and output p for the receiver.

iv. Same as steps (2.b.ii-iii) above, except the common input is $(\textsc{b-out}, b, C)$. Using $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$, the sender forces output q for the receiver, and the receiver delivers outputs p and q to $\mathcal{Z}_{\mathcal{F}}^*$.

(d) When $\mathcal{Z}_{\mathcal{F}}^*$ terminates, the receiver privately records its output, and notifies the sender to begin the next iteration of this loop.

3. If a majority of the simulated $\mathcal{Z}_{\mathcal{F}}^*$-instances output 1, then the receiver terminates with local output $(\textsc{reveal}, b)$. Otherwise the receiver aborts.

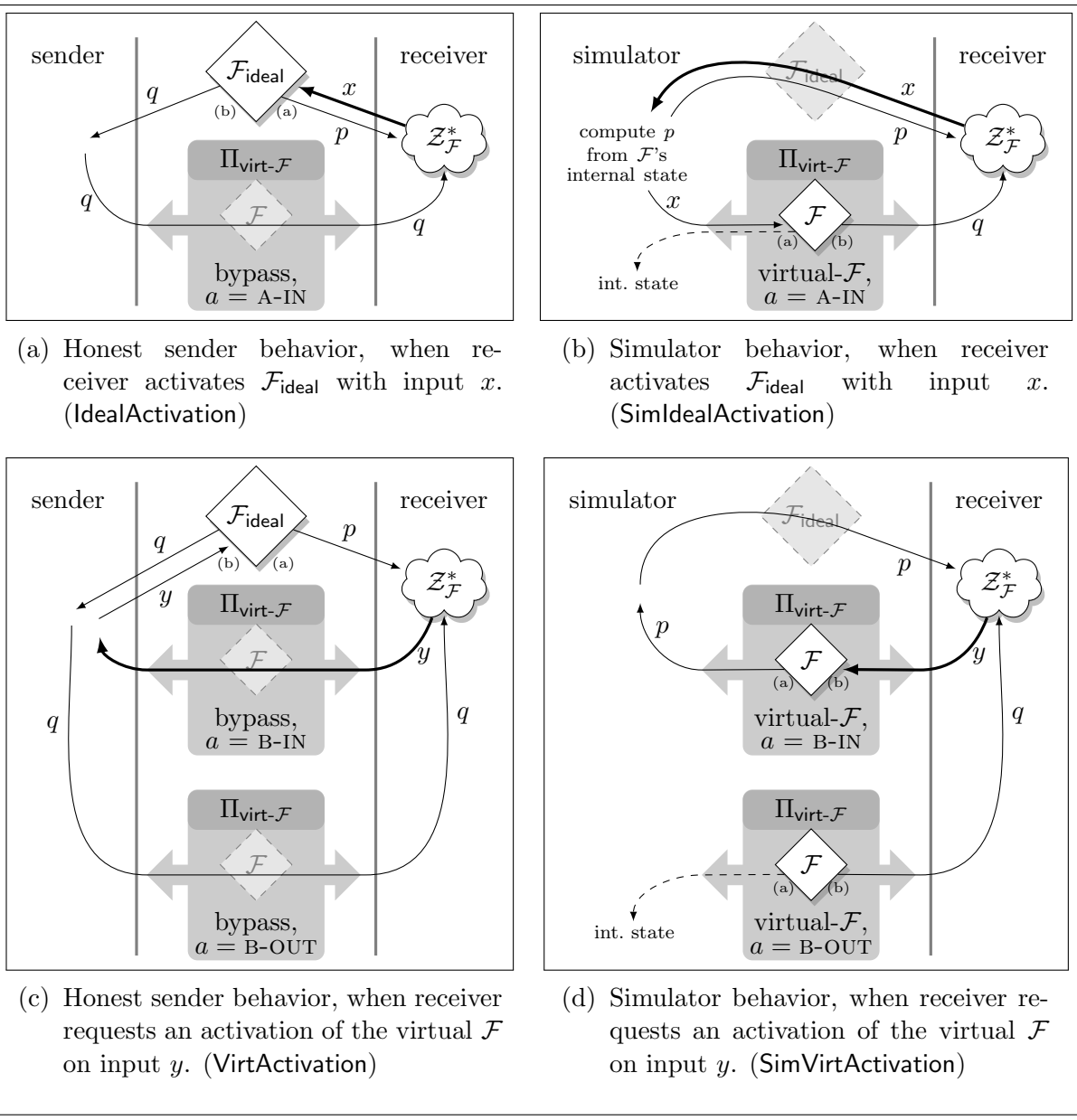
4.3 Security Properties

By inspection, it can be seen that an honest receiver accepts the decommitment of an honest sender with overwhelming probability. This is because the view of $\mathcal{Z}_{\mathcal{F}}^*$ (each iteration of the main loop of $\Pi_{\mathsf{EqCom}}^{\mathcal{F}}$) is that of interacting with a single instance of $\mathcal{F}$, just as in the splittability definition.[8] In this case, $\mathcal{Z}_{\mathcal{F}}^*$ outputs 1 with probability at least $\frac{1}{2} + \delta(k)$. By the Chernoff bound, it is with overwhelming probability that the majority of these $\mathcal{Z}_{\mathcal{F}}^*$ instances output 1, and the receiver outputs $(\textsc{reveal}, b)$.

We formalize the security of $\Pi_{\mathsf{EqCom}}^{\mathcal{F}}$ in the following lemmas.

Lemma 4.1. *$\Pi_{\mathsf{EqCom}}^{\mathcal{F}}$ has a UC simulator in the case that the receiver is corrupt (i.e., an equivocating simulator).*

[8] Both parties are honest, but an external adversary may still interact with $\mathcal{F}_{\mathsf{ideal}}$. This is the step where we must use the fact that $\mathcal{F}_{\mathsf{ideal}}$ ignores all activations from the adversary.



(a) Honest sender behavior, when receiver activates $\mathcal{F}_{\mathsf{ideal}}$ with input x. (IdealActivation)

(b) Simulator behavior, when receiver activates $\mathcal{F}_{\mathsf{ideal}}$ with input x. (SimIdealActivation)

(c) Honest sender behavior, when receiver requests an activation of the virtual $\mathcal{F}$ on input y. (VirtActivation)

(d) Simulator behavior, when receiver requests an activation of the virtual $\mathcal{F}$ on input y. (SimVirtActivation)

Figure 3: Interactions used in $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ and its security proof. Grayed-out instances of $\mathcal{F}$ are being bypassed (by the bypass mode of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ or by virtue of the simulation).

Proof. Consider an interaction in $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ between a corrupt receiver and a simulator who runs the instance of $\mathcal{F}_{\mathsf{ideal}}$ and the sender (on the correct input) honestly. This interaction is identical to the real-process interaction with the corrupt receiver.

Suppose the ith invocation of the $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ subprotocol does not abort or result in output of $\perp$. Then by the standalone security of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$, there is a (possibly rewinding) simulator that can extract an *effective* input of the adversary, $(\hat{y}, \hat{S}_2)$. The honest party's output from $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ is computationally indistinguishable from the prescribed output of $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ with the honest party's input and the adversary's effective input. Thus $\mathsf{Reconstruct}(S_1, \hat{S}_2) \neq \perp$ except with negligible probability, where S_1 is the secret-share from the $(i-1)$th invocation of the $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ subprotocol. Then by the security of the NMSS scheme, $\hat{S}_2 = S_2$ except with negligible probability, where S_2

is the secret-share from the previous invocation.[9] We conclude that the adversary's *effective input* has the form $(\cdot, S_2)$ — a fact which we use below.

We now construct the simulator and argue its soundness via a sequence of hybrid interactions. We have defined subprotocols IdealActivation and VirtActivation as part of the prescribed protocol. Our simulation differs from the real interaction in that these subprotocols are carried out as follows:

SimVirtActivation: (Figure 3d) Same as VirtActivation, except that the simulator uses input $(\perp, \perp, \perp, S_1)$ in the first execution of $\Pi_{\text{virt-}\mathcal{F}}$ (so $\perp$ is given instead of σ). Then the simulator receives output p (among others) from $\Pi_{\text{virt-}\mathcal{F}}$, which it immediately delivers to the receiver on behalf of $\mathcal{F}_{\text{ideal}}$. The simulator uses input $(\perp, \perp, \perp, S_1)$ in the second execution of $\Pi_{\text{virt-}\mathcal{F}}$. The simulator no longer receives S_2' as output but does receive S (the internal state of the virtual $\mathcal{F}$) as output. The simulator internally records S in the variable S^*.

SimIdealActivation: (Figure 3b) Same as IdealActivation, except that when the receiver provides input x to $\mathcal{F}_{\text{ideal}}$, the simulator computes the output p by simulating an activation of $\mathcal{F}$ on internal state S^* (recorded as above) with input x from Alice. The simulator then gives input $(x, \perp, \perp, S_1)$ to the execution of $\Pi_{\text{virt-}\mathcal{F}}$ (so $\perp$ is given instead of σ, and x is given instead of $\perp$).

We now define our sequence of hybrid simulations. Let $F(k)$ be a (polynomial) bound on the number of times the simulator calls its VirtActivation and IdealActivation subroutines combined.

Hybrid 0: As described above, the simulator honestly simulates the sender and the instance of $\mathcal{F}_{\text{ideal}}$. This interaction is identical to the real-process interaction.

Hybrid $(1, i)$*:* (for $0 \leq i \leq F(k)$) Same as Hybrid 0, with the following modifications: After $F(k) - i$ invocations of either VirtActivation or IdealActivation, let S^* denote the internal state of the simulated $\mathcal{F}_{\text{ideal}}$. At this point the simulator generates a random $\widehat{S}_1$ subject to the constraint $\mathsf{Reconstruct}(\widehat{S}_1, S_2) = S^* \| 0^k$, and updates $S_1 := \widehat{S}_1$. Thereafter, it replaces subroutines VirtActivation and IdealActivation with their Sim- versions.

Hybrid 0 is equivalent to Hybrid $(1, 0)$. The main step in this proof is to show the indistinguishability of Hybrids $(1, i)$ and $(1, i-1)$. We do so below.

Hybrid 2: Same as Hybrid $(1, F(k))$, except that the sender uses the (rewinding) ZK simulator in step 2 of the commit phase, to prove the given statement. Note that in this interaction, the value σ is never used (in the commit or reveal phases). The hybrids are indistinguishable by the security of the ZK proof scheme.

Hybrid 3: Same as Hybrid 2, except in the commit phase the sender chooses b independently at random, rather than using the b given as input. The hybrids are indistinguishable by the computational hiding property of COM, since the opening of this commitment is not used.

Hybrid 4: Same as Hybrid 3, except that the sender honestly proves the ZK proof in step 2 of the commit phase. The hybrids are indistinguishable by the security of the ZK proof scheme. This is our final hybrid, and although some intermediate steps employed rewinding, Hybrid 4 is a straight-line simulation.

We now prove that Hybrids $(1, i-1)$ and $(1, i)$ are indistinguishable. The only difference is in how the $j = (F(k) - i + 1)$th call to either VirtActivation or IdealActivation is handled. We consider two cases, depending in which kind of activation is being performed:

[9] S_2 is included in the output of the honest party in this interaction, so it is well-defined.

IdealActivation: Suppose the jth such call is to IdealActivation. Let $\mathcal{F}_A(S,x)$, $\mathcal{F}_B(S,x)$ and $\mathcal{F}_S(S,x)$ denote the distributions of output for Alice, output for Bob and internal state, respectively, resulting from an activation of $\mathcal{F}$ with internal state S and Alice input x. Let S_2 denote the value known to the simulator at this point, and let S^* denote the internal state of the simulated $\mathcal{F}_{\mathsf{ideal}}$.

In Hybrid $(1, i-1)$, the simulator computes $p = \mathcal{F}_A(S^*, x)$ and $q = \mathcal{F}_B(S^*, x)$ (implicitly, by simulating $\mathcal{F}_{\mathsf{ideal}}$) and then delivers p to the receiver. It then invokes $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ in bypass mode with inputs q and S_1, so that the receiver's prescribed output includes the value q. Then it updates S_1 to be random subject to $\mathsf{Reconstruct}(S_1, S_2') = \mathcal{F}_S(S^*, x)\|0^k$, and it updates $S^* := \mathcal{F}_S(S^*, x)$.

In Hybrid $(1, i)$, the simulator computes $p = \mathcal{F}_A(S^*, x)$ (explicitly) and delivers it to the receiver. It then invokes $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ in virtual-$\mathcal{F}$ mode with input x and a value S_1 chosen so that $\mathsf{Reconstruct}(S_1, S_2) = S^*\|0^k$. Thus the prescribed output of the receiver includes the value $q = \mathcal{F}_B(S^*, x)$, and the prescribed outputs S_1' and S_2' are a random sharing of $\mathcal{F}_S(S^*, x)\|0^k$. The sender's prescribed output includes the value $\mathcal{F}_S(S^*, x)$, which is stored in the variable S^*.

It is straight-forward to verify that the joint distribution of the simulator's S_1, S^* variables, the receiver's output p, and the receiver's prescribed output from $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ is identical in the two hybrids, for any (effective) receiver input including the correct S_2 value. By our previous argument, the receiver's effective input contains either the correct S_2 value or a value of S_2 that causes $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ to output $\perp$, with overwhelming probability. Thus the views of the receiver in the two hybrids are indistinguishable, by the standalone security of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$.

VirtActivation: For the other case, suppose the jth such call is to VirtActivation. Similar to above, let S_1, S_2, S^* be local variables to the simulation, and let $\mathcal{F}_A$, $\mathcal{F}_B$, and $\mathcal{F}_S$ be the same, except considering an input from Bob instead of Alice.

In Hybrid $(1, i-1)$, the simulator obtains the receiver's input y from the bypass invocation of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$. It gives y to $\mathcal{F}_{\mathsf{ideal}}$, resulting in an output of $p = \mathcal{F}_A(S^*, y)$ to the receiver and $q = \mathcal{F}_B(S^*, y)$ to the sender. It then uses the bypass mode of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ to give output q to the receiver. Finally, S^* is updated to $\mathcal{F}_S(S^*, y)$, and the simulator updates S_1 so that $\mathsf{Reconstruct}(S_1, S_2) = \mathcal{F}_S(S^*, y)\|0^k$.

In Hybrid $(1, i)$, the simulator obtains prescribed output $p = \mathcal{F}_A(S^*, y)$ from $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$, which it immediately delivers to the receiver on behalf of $\mathcal{F}_{\mathsf{ideal}}$. In the second invocation of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ the receiver's prescribed output includes $q = \mathcal{F}_B(S^*, y)$, while the sender's prescribed output includes $\mathcal{F}_S(S^*, y)$, which is stored in variable S^*. The prescribed values of S_1 and S_2 are a random share of $\mathcal{F}_S(S^*, y)\|0^k$.

Thus we again have that the joint distribution of the simulator's S_1, S^* variables, the receiver's output p, and the receiver's prescribed output from $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ is identical in the two hybrids, for any (effective) receiver input including the correct S_2 value. As above, the hybrids are indistinguishable.[10]

Hybrid 4 does not require the simulator to know the sender's correct bit during the commit phase, and therefore it is a valid UC simulation (*i.e.*, it can be carried out in the $\mathcal{F}_{\mathsf{com}}$-hybrid model). The simulator implicit in Hybrid 4 — namely, one which commits to a random bit and then in the

[10] Technically, one must introduce another intermediate hybrid since Hybrids $(1, i-1)$ and $(1, i)$ differ in the executions of *two* $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ subprotocols. We omit the straight-forward details.

reveal phase replaces VirtActivation and IdealActivation subroutines with their Sim- counterparts — is our final equivocating UC simulator. We have shown that the real-process interaction with the adversary and honest sender is indistinguishable from the ideal-process interaction with the given simulator. □

Lemma 4.2. *If $\mathcal{F}$ is R-strongly-unsplittable, then $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ is binding in the standalone sense. That is, for all adversarial senders, there exists a (possibly rewinding) simulator that extracts a value b during the commit phase such that* $\Pr[$*receiver outputs* $(\textsc{reveal}, 1-b)]$ *is negligible.*

Proof. We construct the rewinding simulator and argue its correctness via the following sequence of hybrids:

Hybrid 0: The real-process interaction, between a corrupt sender, honest receiver, and instances of $\mathcal{F}_{\mathsf{ideal}}$ simulated honestly.

Hybrid 1: Same as Hybrid 0, except that the receiver uses the rewinding proof-of-knowledge extractor in step 2 of the commit phase, to obtain a witness (σ, b) to the statement being proved. The indistinguishability of these two hybrids follows from the security of the zero-knowledge proof scheme. With overwhelming probability, b is the *unique* value to which the COM-commitment C can be opened, by the statistical binding property of COM and the soundness of the knowledge extraction. Hereafter, we condition on this event. Note that no instance of $\mathcal{F}$ within $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$, nor any instance of $\mathcal{Z}^*_{\mathcal{F}}$ simulated by the receiver is active during the commit phase, so the rewinding does not affect them. This hybrid defines our extracting simulator (where the extracted value is b obtained from the proof-of-knowledge extractor). The remainder of the hybrids establish that $\Pr[$receiver outputs $(\textsc{reveal}, 1-b)]$ is negligible. We therefore assume that the sender is attempting to decommit to $1-b$ in the reveal phase.

Hybrid 2: Same as Hybrid 1, except that every instance of the $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ protocol is replaced by the receiver simulating an ideal instance of $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ and running the (possibly rewinding) simulator with the sender. The two hybrids are indistinguishable by the standalone security of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$. We are trying to bound $\Pr[$receiver outputs $(\textsc{reveal}, 1-b)]$, and the receiver can only make such an output if it reaches step 3 of the reveal phase. In particular, the receiver will abort early if it is notified of any activation of $\mathcal{F}_{\mathsf{ideal}}$ during the execution of a $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ instance. Thus, without loss of generality we assume that the sender does not interact with $\mathcal{F}_{\mathsf{ideal}}$ during any instance of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$.[11] Similarly, the honest receiver's simulated instance of $\mathcal{Z}^*_{\mathcal{F}}$ is not activated at any time during the execution of a $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ instance. Thus no instance of either $\mathcal{Z}^*_{\mathcal{F}}$ or $\mathcal{F}_{\mathsf{ideal}}$ is affected by the rewinding introduced in this hybrid.

Hybrid 3: Same as Hybrid 2, except that the collected instances of $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ in each main loop of $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ are replaced with a single instance of $\mathcal{F}$ that leaks its internal state to Alice. More formally, we see that the parties' prescribed interactions with $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ constitute a UC-secure protocol for such a variant of $\mathcal{F}$ (in the case where there does not exist any valid opening of C to the value $1-b$, as we conditioned on). We replace this (implicit) protocol with an ideal instance of such an $\mathcal{F}$ and the appropriate simulator for the sender. These two hybrids are indistinguishable by the security of this implicit protocol (from the statistical binding property of COM and the non-malleability of NMSS).

[11] A corrupt sender could interact with $\mathcal{F}_{\mathsf{ideal}}$ and delay the notification sent to the receiver, but without loss of generality the interaction with $\mathcal{F}_{\mathsf{ideal}}$ could itself be delayed instead.

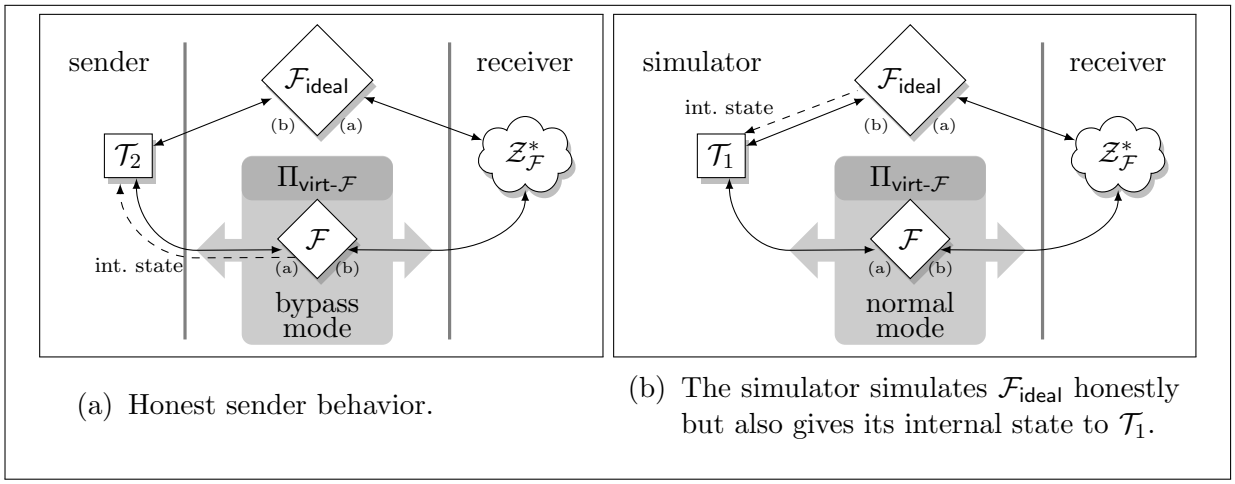
In each iteration of the main loop of the reveal phase, Hybrid 3 consists of an instance of $\mathcal{Z}^*_{\mathcal{F}}$, $\mathcal{F}_{\mathsf{ideal}}$, and the new (leaky) instance of $\mathcal{F}$, connected as in the R-splittability interaction. At no point are any of these instances rewound. Everything else in the interaction (including the global environment and the corrupt sender, each possibly being rewound) can therefore be taken as a machine $\mathcal{T}$ in the definition of $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$. By the R-strong-unsplittability of $\mathcal{F}$, we have that $\Pr[\mathcal{Z}^*_{\mathcal{F}} \text{ outputs } 1] < \frac{1}{2} - \delta(k)$. For the receiver to output $(\text{REVEAL}, 1-b)$, a majority of these $N(k) = O(k/\delta(k)^2)$ independent splittability interactions must end with $\mathcal{Z}^*_{\mathcal{F}}$ outputting 1. By the Chernoff bound, this can happen only with negligible probability. This completes the proof. □

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 4):



(a) Honest sender behavior.

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

Figure 4: Interactions in $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ and its security proof.

- 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.

- 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.

5.2 The Virtual-$\mathcal{F}$ Subprotocol

Let $\mathsf{NMSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ be a non-malleable secret sharing scheme that can support messages of length k. For notational simplicity, we suppose that in NMSS the pair (ϵ, ϵ) constitutes a valid secret share of the value 0^k. We also assume that 0^k is the initial internal state of $\mathcal{F}$. Let COM be a standalone-secure, plain-model commitment protocol with non-interactive opening phase. Then $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ is the non-reactive, 2-party ideal functionality defined as follows, with global security parameter k:

1. $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ waits for an input of the form (x, σ, S_1) from the **sender**, and an input of the form (y, S_2) from the **receiver**, and a common input of the form (a, b, C).
2. If $\mathsf{Reconstruct}(S_1, S_2) = \perp$, then output $\perp$ to both parties and halt. Otherwise let $S = \mathsf{Reconstruct}(S_1, S_2)$.
   (a) If $a =$ A-IN then simulate an activation of $\mathcal{F}$ on internal state S and input x from Alice (y is ignored). If $a =$ B-IN, then simulate an activation of $\mathcal{F}$ on internal state S and input y from Bob (x is ignored). Suppose the activation results in new internal state S', output p for Alice and output q for Bob.
   (b) Generate $(S'_1, S'_2) \leftarrow \mathsf{Share}(S')$ and give output (q, S'_2) to the receiver. If σ is a valid opening of commitment transcript C to value b, then we say that the functionality is in **bypass mode**. In bypass mode, give output (p, S'_1, S) to the sender. Otherwise, in **normal mode** give output (p, S'_1) to the sender.

Under the SHOT assumption, there exists a standalone-secure protocol $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ for $\widetilde{\mathcal{F}}_{\mathsf{virt}}$.

5.3 UC-Equivocal Commitment

Let COM be a statistically-binding, standalone-secure commitment protocol with non-interactive opening phase. Let $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ be as above, with respect to the same COM protocol. Let $\mathcal{Z}^*_{\mathcal{F}}$ be the environment guaranteed by the strong unsplittability of $\mathcal{F}$, which outputs 1 with probability at least $\frac{1}{2} + \delta(k)$ when interacting with an instance of $\mathcal{F}$, and with probability at most $\frac{1}{2} - \delta(k)$ when interacting with an instance of $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$. The function δ is guaranteed to be a noticeable function. Let

$\mathcal{T}_1$ and $\mathcal{T}_2$ be the machines guaranteed by the L- and R-splittability of $\mathcal{F}$, respectively. We require $\mathcal{T}_2$ to be a uniform machine. The protocol $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ proceeds as follows, with security parameter k:

Commit phase: When the sender receives the command (COMMIT, b):

1. The sender commits to b under the COM scheme. Let C be the resulting transcript, and let σ be the non-interactive opening of C to b.
2. The sender uses a (standalone-secure) zero-knowledge proof of knowledge to prove knowledge of (σ, b) such that σ is a valid opening of C to b.

Reveal phase: Both parties are connected to an ideal instance of the multi-session version of $\mathcal{F}$, which for clarity we denote $\mathcal{F}_{\mathsf{ideal}}$. The sender is connected to $\mathcal{F}_{\mathsf{ideal}}$ in the role of Bob, and the receiver is connected in the role of Alice. When the sender receives the command REVEAL, the parties continue as follows:

1. The sender gives b to the receiver.
2. The parties do the following, $N(k) = O(k/\delta(k)^2)$ times, each with a new session of $\mathcal{F}_{\mathsf{ideal}}$:
   (a) The receiver internally simulates a fresh instance of $\mathcal{Z}^*_{\mathcal{F}}$ on security parameter k. The sender internally simulates a fresh instance of $\mathcal{T}_2$ on security parameter k.
   (b) The sender initializes local value $S_1 := \epsilon$, and the receiver initializes local value $S_2 := \epsilon$. (By our convention, these are valid NMSS shares of 0^k, the initial internal state of $\mathcal{F}$.)
   (c) Whenever $\mathcal{Z}^*_{\mathcal{F}}$ generates a command x to send to Alice, the parties do the following:
      i. The receiver sends x to $\mathcal{F}_{\mathsf{ideal}}$, resulting in output p for the receiver and q for the sender. The sender gives q to $\mathcal{T}_2$ on behalf of $\mathcal{F}_L$.
      ii. When $\mathcal{T}_2$ generates an input x' for $\mathcal{F}_R$, the parties initiate a new instance of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$. The sender provides input (x', σ, S_1) and the receiver provides input $(\perp, S_2)$, with common input $(\text{A-IN}, b, C)$. If $\mathcal{F}_{\mathsf{ideal}}$ is activated during the execution of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$, both parties abort.
      iii. The $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ instance terminates with output (p', S'_1, S) to the sender and (q', S'_2) to the receiver. The sender gives p' and S to $\mathcal{T}_2$ on behalf of $\mathcal{F}_R$ (as the output and internal state, respectively). The receiver gives both p and q' to $\mathcal{Z}^*_{\mathcal{F}}$, on its Alice- and Bob- input/output tapes, respectively. Both parties update $S_1 := S'_1$ and $S_2 := S'_2$, appropriately.
   (d) Whenever $\mathcal{Z}^*_{\mathcal{F}}$ generates a command y to send to Bob, the parties do the following:
      i. The parties initiate a new instance of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$. The sender provides input $(\perp, \sigma, S_1)$ and the receiver provides input (y, S_2), with common input $(\text{B-IN}, b, C)$. If $\mathcal{F}_{\mathsf{ideal}}$ is activated during the execution of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$, both parties abort.
      ii. The $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ instance terminates with output (p, S'_1, S) to the sender and (q, S'_2) to the receiver. The sender gives p and S to $\mathcal{T}_2$, on behalf of $\mathcal{F}_R$ (as the output and internal state, respectively).
      iii. When $\mathcal{T}_2$ generates an input y' to $\mathcal{F}_L$, the sender gives input y' to $\mathcal{F}_{\mathsf{ideal}}$, resulting in output p' for the receiver and q' for the sender. The sender gives q' to $\mathcal{T}_2$ on behalf of $\mathcal{F}_L$. The receiver gives both p' and q to $\mathcal{Z}^*_{\mathcal{F}}$, on its Alice- and Bob- input/output tapes, respectively. Both parties update $S_1 := S'_1$ and $S_2 := S'_2$, appropriately.

(e) When $\mathcal{Z}^*_{\mathcal{F}}$ terminates, the receiver privately records its output, and notifies the sender to begin the next iteration of this loop.

3. If a majority of the simulated $\mathcal{Z}^*_{\mathcal{F}}$-instances output 1, then the receiver terminates with local output (REVEAL, b). Otherwise the receiver aborts.

5.4 Security Properties

Correctness of the protocol can be seen by inspection, similar to the previous section. We now sketch the security of $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ in the following lemmas.

Lemma 5.1. $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ *has a UC simulator in the case that the receiver is corrupt (i.e., an equivocating simulator).*

Proof. Many technical aspects of the proof follow those of the previous section, so we focus on the important differences. We start by considering an interaction between an honest sender and corrupt receiver, with all instances of $\mathcal{F}_{\mathsf{ideal}}$ being simulated honestly.

Hybrid 1: Same as the real interaction, except that instead of running the $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ protocol, the sender gives its input to a simulated instance of $\widetilde{\mathcal{F}}_{\mathsf{virt}}$, and then runs the (possibly rewinding) simulator for the $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ protocol with the receiver. As before, it is without loss of generality that the rewinding does not involve any instance of $\mathcal{F}_{\mathsf{ideal}}$. This hybrid is indistinguishable from the real interaction by a straight-forward application of the standalone security of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$.

Hybrid 2: Same as Hybrid 1, except that each time through the main loop of $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$, the multiple activations of $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ are replaced with a single instance of $\mathcal{F}$ that leaks its internal state to Alice. These hybrids are indistinguishable by the non-malleability property of NMSS, and the definition of $\widetilde{\mathcal{F}}_{\mathsf{virt}}$.

Hybrid 3: Same as Hybrid 2, except that instead of $\mathcal{F}_{\mathsf{ideal}}$, $\mathcal{T}_2$, and this new instance of $\mathcal{F}$ each time through the main loop of $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$, the sender uses an instance of $\mathcal{F}_{\mathsf{ideal}}$ that leaks its internal state to Bob, $\mathcal{T}_1$, and a (non-leaking) instance of $\mathcal{F}$, respectively. Such a change is indistinguishable by the L/R-splittability of $\mathcal{F}$.[12]

Hybrid 4: Same as Hybrid 3, except that the plain (virtual) instance of $\mathcal{F}$ is replaced with successive executions of the $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ protocol, in which the sender does *not* provide the value σ. These hybrids are indistinguishable, by essentially the same argument for Hybrids 1–2 but in reverse. We also use the fact that Bob's prescribed output from $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ is identical whether $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ is in bypass or normal mode. In this hybrid, the value σ is not used in the reveal phase.

Hybrid 5: Same as Hybrid 4, except that the sender commits to an independently random bit in the commit phase. Using the same argument as earlier, we apply the security of the ZK proof system and the computational binding property of COM to show that these hybrids are indistinguishable.

Although the intermediate steps involved rewinding, Hybrid 5 is a straight-line simulation. The simulation implicit in Hybrid 5 (namely, it commits to a random bit, then in the reveal phase honestly simulates $\mathcal{F}_{\mathsf{ideal}}$ while leaking internal state to $\mathcal{T}_1$ and using the normal mode of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$) is our final UC simulation, since it does not require the value b until the reveal phase. □

[12] Here it is important that $\mathcal{T}_1$ and $\mathcal{T}_2$ be able to "fool" *all* environments, not just $\mathcal{Z}^*_{\mathcal{F}}$.

Lemma 5.2. $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ *is binding in the standalone sense. For all adversarial senders, there exists a (possibly rewinding) simulator that extracts a value b during the commit phase such that* $\Pr[$*receiver outputs* $(\textsc{reveal}, 1-b)]$ *is negligible.*

Proof. This proof is similar to the analogous one in the previous section. The rewinding simulator uses the proof-of-knowledge extractor in step 2 of the commit phase to extract b. To show that $\Pr[$receiver outputs $(\textsc{reveal}, 1-b)]$ is negligible, we condition on the event that the standalone commitment C can be opened only to a unique value b. In that case, a corrupt sender cannot use the bypass mode of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$. As before, we repeatedly apply the standalone security of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}}$ and the specification of $\widetilde{\mathcal{F}}_{\mathsf{virt}}$ to obtain an indistinguishable interaction that involves $\mathcal{Z}^*_{\mathcal{F}}$, two instances of $\mathcal{F}$, and an adversarial machine between them just as in the unsplittability definition. Now neither of the two instances of $\mathcal{F}$ leak their internal state, so we have an interaction as in the definition of (plain) strong unsplittability. Therefore $\mathcal{Z}^*_{\mathcal{F}}$ outputs 1 with probability at most $\frac{1}{2} - \delta(k)$, by the strong unsplittability of $\mathcal{F}$. It follows that $\Pr[$receiver outputs $(\textsc{reveal}, 1-b)]$ is negligible. □

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. We use the following additional properties of the $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ protocols from Section 4 and Section 5:

- To commit to b in $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$, the sender commits to b under COM and gives a zero-knowledge proof of knowledge of the opening to the commitment. Therefore, we could define a non-interactive reveal phase in which the sender simply reveals σ (the non-interactive opening to the COM commitment). It follows immediately from the statistical binding property of COM that the $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ commitment is *statistically* binding with respect to this alternative non-interactive opening phase.
- The (equivocating) UC simulator for a corrupt receiver gives *honest* commitments to a random bit in the commit phase.

Our final UC commitment protocol is as follows:

$\Pi^{\mathcal{F}}_{\mathsf{com}}$ commitment protocol: Let $\mathsf{NMSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ be a non-malleable secret sharing scheme, and let $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ be the commitment protocol from Section 4. Our final commitment protocol $\Pi^{\mathcal{F}}_{\mathsf{com}}$ is defined as follows:

Commit phase: When the sender receives input $(\textsc{commit}, b)$:

1. The receiver chooses random $r \leftarrow \{0,1\}^k$ containing half 0s and half 1s, then commits to r under $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$.
2. The sender chooses random $x \leftarrow \{0,1\}^k$ and commits to x, bitwise under $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$. Let C_i and σ_i denote the commitment transcripts and *non-interactive* decommitment values, respectively, for the commitment to bit x_i. The sender gives gives $z = b \oplus (\bigoplus_i x_i)$ to the receiver.
3. Both parties engage in a standalone-secure subprotocol ρ for the following task, with common input $(C_1, \ldots, C_k)$ the sender providing input $(x, \sigma_1, \ldots, \sigma_k)$:
   (a) If σ_i is not a valid opening of C_i to x_i, for any i, then abort.

(b) Choose random $s \leftarrow \{0,1\}^k$ containing half 0s and half 1s. Define $x|_s$ to be the string in $\{0,1,\perp\}^k$, where

$$(x|_s)_i = \begin{cases} x_i & \text{if } s_i = 1 \\ \perp & \text{if } s_i = 0 \end{cases}$$

(c) Generate $(\alpha, \beta) \leftarrow \mathsf{Share}(s)$.

(d) Give output α to the sender and $(\beta, x|_s)$ to the receiver.

If the subprotocol aborts or outputs $\perp$, both parties abort.

4. The receiver opens the $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ commitment to r. If r does not contain half 0s and half 1s (or if the commitment cannot be successfully opened), then the sender aborts. Otherwise the sender gives $x|_r$ to the receiver.

5. The receiver outputs (COMMITTED).

Reveal phase: When the sender receives input (REVEAL):

1. The parties engage in a standalone-secure subprotocol ϕ for the following task, with common input r, input α from the sender, and input β from the receiver:

   (a) If $\mathsf{Reconstruct}(\alpha, \beta) = \perp$ then output $\perp$ to both parties.

   (b) Otherwise, let $s = \mathsf{Reconstruct}(\alpha, \beta)$. If $s = \overline{r}$ then choose random $i^* \leftarrow \{1, \ldots, k \mid r_i = 0\}$. If $s \neq \overline{r}$ then choose random $i^* \leftarrow \{1, \ldots, k \mid r_i = s_i = 0\}$. Give i^* to the sender and output OK to the receiver.

   If the subprotocol aborts or gives output $\perp$ then both parties abort.

2. The sender gives b and x to the receiver and opens the commitments $C_1, \ldots, C_k$ using the interactive opening of $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$. The receiver aborts if for any i, the commitment C_i is not opened to x_i. The receiver further verifies that x is consistent with the values $x|_s$ and $x|_r$ from the commit phase, and that $z = b \oplus (\bigoplus_i x_i)$. If so, then the receiver outputs (REVEAL, b); otherwise it aborts.

Lemma 6.1. *$\Pi^{\mathcal{F}}_{\mathsf{com}}$ is a UC-secure protocol for commitment, in the $\mathcal{F}$-hybrid model.*

Proof. The correctness of the protocol can be seen by inspection. We must demonstrate a simulation for both of the following cases:

When the sender is corrupt: We construct the simulator via the following sequence of hybrids:

Hybrid 0: The real-process interaction, between an honest receiver and corrupt sender. All instance of $\mathcal{F}$ are simulated honestly.

Hybrid 1: Same as Hybrid 0, except that the receiver uses the simulator for $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ to give an equivocal commitment to r. Then the random selection of r can be postponed until step 4 of the commit phase. These hybrids are indistinguishable by the security of $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$.

Hybrid 2: Same as Hybrid 1, except that in step 4, r is chosen as $\overline{s}$. This correlates r and s, whereas before they were independent. However, the sender's view is only influenced by s within the ρ and ϕ subprotocols. The sender's prescribed output in the ρ and ϕ subprotocols is independent of s (in the ϕ subprotocol, the sender's output is always a randomly chosen

position at which r contains a zero). Thus by the security of the ϕ and ρ subprotocols, these two hybrids are indistinguishable. As in all of the hybrids, the honest receiver will only accept a decommitment if x is revealed to be consistent with both $x|_s$ and $x|_r$. But in this interaction, there is a *unique* value $\hat{x}$ consistent with both $x|_s$ and $x|_r$. The sender can compute $\hat{b} = z \oplus (\bigoplus_i \hat{x}_i)$, and we have that Pr[receiver outputs $(\text{REVEAL}, 1 - \hat{b})$] is zero.

Hybrid 2 defines our final simulator: it uses equivocal commitments to r, and then after running the ρ subprotocol opens these commitments to $r = \overline{s}$. As such, it learns all the bits of x and can extract the committed value $\hat{b}$. The soundness of this simulation follows from the indistinguishability of Hybrids 0–2.

When the receiver is corrupt: We construct the simulator via the following sequence of hybrids:

Hybrid 0: The real-process interaction, in which an honest sender commits to b to an adversarial receiver. All instances of $\mathcal{F}$ are simulated honestly.

Hybrid 1: Same as Hybrid 0, except that the sender runs the (rewinding) extracting simulator for the receiver's commitment of r in the $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ protocol. The sender thus obtains a value r that with overwhelming probability equals the value r to which the receiver opens the commitment in step 4. These two hybrids are indistinguishable by the standalone security property of $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$.

Hybrid 2: Same as Hybrid 1, except that in step 3 the sender chooses random $s \in \{0,1\}^k$ with half 0s and half 1s. It generates $(\alpha, \beta) \leftarrow \mathsf{Share}(s)$. It then runs the (rewinding) simulator for the ρ subprotocol with $(\beta, x|_s)$ as its input. These two hybrids are indistinguishable by the standalone security of the ρ subprotocol.

Hybrid 3: Same as Hybrid 2, except that the sender chooses s as above, subject to the additional constraint that $s \neq \overline{r}$. Therefore we have that $s \vee r$ contains a zero in at least one position. These two hybrids are statistically indistinguishable.

Hybrid 4: Same as Hybrid 3, except that after step 1 of the commit phase, the sender chooses a random value $i^* \leftarrow \{i \in \{1, \ldots, k\} \mid r_i = 0\}$. Then the sender chooses the values $x_1, \ldots, x_{i^*-1}, x_{i^*+1}, \ldots, x_k, z$ to be random bits, and sets $x_{i^*} = b \oplus z \oplus (\bigoplus_{i \neq i^*} x_i)$. Later, in step 3 the sender chooses $s \in \{0,1\}^k$ subject to the constraint that $s \vee r$ contains a zero in position i^*. Hybrids 3 and 4 are thus identically distributed. Note that b is used only to determine the value x_{i^*}. In the commit phase, the value x_{i^*} is used *only* to generate a $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$-commitment, except with overwhelming probability (corresponding to the event that the receiver opens its $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$-commitment to a different value of r than was extracted in step 1, and the sender would have to explicitly reveal x_{i^*} in step 4).

Hybrid 5: Same as Hybrid 4, except the sender runs the UC simulator rather than the honest protocol for all of the $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ commitments it sends and opens. These hybrids are indistinguishable by the security of $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$. Importantly, this interaction does not use the value of x_{i^*} (and thus b) during the commit phase (either by sending it to the receiver of as input to a $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ commitment). We also know that the $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ simulator gives honest commitments to randomly chosen bits in the commit phase. Thus we also make the following modification which does not affect the distribution of the interaction: all x_i values (including x_{i^*}) are chosen to be the random bits that are honestly committed to by the $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ simulator. Only after step 1 of the reveal phase do we then change x_{i^*} to the value $b \oplus z \oplus (\bigoplus_{i \neq i^*} x_i)$.

Hybrid 6: Same as Hybrid 5, except that the sender does not pick i^* during the commit phase. Instead it chooses s as it did in Hybrid 3. Then after step 1 of the reveal phase, it determines a position i^* at which $s \vee r$ contains a zero (which must exist because of how s is chosen). This hybrid is distributed identically to Hybrid 5.

Hybrid 7: Same as Hybrid 6, except that instead of honestly running the ϕ subprotocol, it simulates an ideal instance of the appropriate functionality and uses the (rewinding) simulator on the adversary. By the standalone security of ϕ, the hybrids are indistinguishable. Furthermore, the sender obtains the receiver's effective input β' to ϕ. By the non-malleability of NMSS, we have that with overwhelming probability, either the sender aborts after the ϕ subprotocol, or the sender's output from the ϕ subprotocol is distributed identically to the i^* value chosen by the sender at this point. Thus it does not affect the interaction for the sender to use the output of ϕ as its i^* value, instead of how i^* is chosen in Hybrid 6. This hybrid therefore does not use the value s generated in the commit phase.

Hybrid 8: Same as Hybrid 7, except that the sender runs the ρ and ϕ subprotocols honestly. By the same arguments as in Hybrids 2–3 and 7, we have that Hybrids 7 and 8 are indistinguishable. We note that because the sender is giving honest $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ commitments to the bits $x_1, \ldots, x_k$ in the commit phase, it has appropriate inputs $\sigma_1, \ldots, \sigma_k$ to give as input to the ρ subprotocol to yield the correct prescribed output for the receiver.

Hybrid 9: Same as Hybrid 8, except that the sender does not extract the value r; instead it runs the $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ protocol receiver protocol honestly in step 1 of the commit phase. The value r is no longer being used in the commit phase in these interactions, so these hybrids are indistinguishable from the standalone security property of $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$.

Hybrid 9 finally defines our simulation: the sender commits to random values x in the commit phase. Then later in the reveal phase, the sender learns a position i^* at which $r \vee s$ contains a zero (except in the negligible-probability event that $r = \overline{s}$). It then equivocates on the opening of commitment $\#i^*$, if necessary, to decommit to the value of its choice. The soundness of the simulation follows from the indistinguishability of Hybrids 0 and 9. □

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 [DNO10, MPR10]. 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 7.1. *If $\mathcal{F}$ is complete, then $\mathcal{F}$ is infinitely-often strongly unsplittable for* uniform $\mathcal{T}$.

Here "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).

Proof. 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 straightforward 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. □

Lemma 7.2. *If $\mathcal{F}$ is complete, then the multi-session version of $\mathcal{F}$ is strongly unsplittable.*

Proof. Let $\widehat{\mathcal{F}}$ denote the multi-session version of $\mathcal{F}$, and suppose it is not strongly unsplittable. It suffices to show that the fair coin tossing functionality $\mathcal{F}_{\mathsf{coin}}$ cannot be securely realized in the $\mathcal{F}$-hybrid model. For the sake of contradiction, let π be such a purported protocol. Consider an environment $\mathcal{Z}$ that internally simulates two honest parties executing one instance of the π protocol, where the communication is routed to an external instance of $\widehat{\mathcal{F}}$. The environment outputs 1 if both parties obtain the same output from π. By the correctness of π, we have that $\mathcal{Z}$ outputs 1 with overwhelming probability.

This environment $\mathcal{Z}$ is uniform and suitable, in the sense of the splittability definition. Then since $\widehat{\mathcal{F}}$ is not strongly unsplittable, there exists a machine $\mathcal{T}$ such that $\Delta_{\mathsf{split}}(\mathcal{Z}, \widehat{\mathcal{F}}, \mathcal{T}, k)$ is non-noticeable. In other words, there exists a negligible function ν so that $\Delta_{\mathsf{split}}(\mathcal{Z}, \widehat{\mathcal{F}}, \mathcal{T}, k) \leq \nu(k)$ for infinitely many k.

The interaction with $\mathcal{Z}$ and $\widehat{\mathcal{F}}$ is infinitely-often indistinguishable from an interaction with $\mathcal{Z}$ and $\widehat{\mathcal{F}}_{\mathsf{split}}^{\mathcal{T}}$. Take this interaction and "repackage" it as follows: Take the honest Bob out of the environment, and subsume $\mathcal{T}$ and $\widehat{\mathcal{F}}_L$ into the environment, so that only $\widehat{\mathcal{F}}_R$ and the honest Bob are outside of the environment. We also insert a dummy adversary relaying between $\mathcal{T}$ (within the environment) and the external $\widehat{\mathcal{F}}_R$. We apply the security of π to this interaction, replacing the dummy adversary, $\widehat{\mathcal{F}}_R$, and honest π with a simulator $\mathcal{S}$, ideal functionality $\mathcal{F}_{\mathsf{coin}}$, and dummy protocol, respectively. The resulting interaction is indistinguishable by the security of π.

Again we repackage the resulting interaction, so that only the honest Alice and $\widehat{\mathcal{F}}_L$ are outside of the environment (along with a dummy adversary interacting as Bob with $\widehat{\mathcal{F}}_L$). Again, we apply the security of π to replace the real interaction with an ideal one involving an instance of $\mathcal{F}_{\mathsf{coin}}$.

However, in this final interaction, the environment outputs 1 if two *independent, ideal instances* of $\mathcal{F}_{\mathsf{coin}}$ output the same bit. This can only happen with probability 1/2, but by the indistinguishability of the interactions we see that it in fact the environment must output 1 with probability negligibly close to 1, for infinitely many values of k. This is a contradiction, so the purported protocol π cannot exist. □

Lemma 7.3. *If $\mathcal{F}$ is complete then $\mathcal{F}$ is strongly unsplittable with respect to environments with multi-bit output.*

Proof. We show that if the condition in the lemma is not met, then the multi-session version of $\mathcal{F}$ is not strongly unsplittable (with respect to environments with single-bit output), as in Lemma 7.2. Let $\mathcal{Z}$ be a suitable (single-bit output) environment that expects to interact with $\widehat{\mathcal{F}}$, the multi-session version of $\mathcal{F}$. Let $N(k)$ be a polynomial upper bound on the number of instances of $\mathcal{F}$ invoked by $\mathcal{Z}$, on security parameter k.

Define $\mathcal{Z}^*$ to be the environment that first chooses a random $i^* \leftarrow \{1, \ldots, N(k)\}$. It then internally simulates $\mathcal{Z}$ and all sessions of $\mathcal{F}$, except for the i^*-th instance, which it routes to an external instance of $\mathcal{F}$. When the internal instance of $\mathcal{Z}$ terminates, $\mathcal{Z}^*$ outputs the *entire view* of $\mathcal{Z}$, as well as the output of $\mathcal{Z}$. Since $\mathcal{Z}^*$ is a suitable and uniform (multi-bit output) environment that expects to interact with a single instance of $\mathcal{F}$, there exists a machine $\mathcal{T}$ such that $\Delta_{\mathsf{split}}(\mathcal{Z}^*, \mathcal{F}, \mathcal{T}, k)$ is non-noticeable. In other words, there is an infinite set $K \subseteq \mathbb{N}$ for which $\Delta_{\mathsf{split}}(\mathcal{Z}^*, \mathcal{F}, \mathcal{T}, k)$ is negligible when restricted to $k \in K$. Hereafter, we implicitly restrict the security parameter to the set K.

Define $\mathcal{Z}^*_j$ to be $\mathcal{Z}^*$ conditioned on $i^* = j$. We have that for each j, $\Delta_{\mathsf{split}}(\mathcal{Z}^*_j, \mathcal{F}, \mathcal{T}, k) \leq \Delta_{\mathsf{split}}(\mathcal{Z}^*, \mathcal{F}, \mathcal{T}, k) \cdot N(k)$, and thus $\Delta_{\mathsf{split}}(\mathcal{Z}^*_j, \mathcal{F}, \mathcal{T}, k)$ is negligible (for security parameters $k \in K$).

Let $\widehat{\mathcal{T}}$ be the "multi-session" extension of $\mathcal{T}$; that is, for every session *sid* of $\mathcal{F}$ started within $\widehat{\mathcal{F}}_L$ or $\widehat{\mathcal{F}}_R$, $\widehat{\mathcal{T}}$ creates a new session of $\mathcal{T}$ with the same *sid*, and routes it to the sessions of $\mathcal{F}$ within $\widehat{\mathcal{F}}_L$ and $\widehat{\mathcal{F}}_R$. In other words, $\widehat{\mathcal{F}}^{\widehat{\mathcal{T}}}_{\mathsf{split}}$ behaves exactly as the multi-session version of $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$. We now argue that $\widehat{\mathcal{T}}$ demonstrates the non-strong-splittability of $\widehat{\mathcal{F}}$ with respect to $\mathcal{Z}$; *i.e.*, $\Delta_{\mathsf{split}}(\mathcal{Z}, \widehat{\mathcal{F}}, \widehat{\mathcal{T}}, k)$ is non-noticeable.

We consider a series of hybrids indexed by $h \in \{0, \ldots, N(k)\}$, which involve $\mathcal{Z}$ interacting with $\widehat{\mathcal{F}}^{\widehat{\mathcal{T}}}_{\mathsf{split}}$ for the first h sessions of $\mathcal{F}$, and $\widehat{\mathcal{F}}$ for the remaining sessions. Hybrids 0 and $N(k)$ are the two interactions referenced in the definition of $\Delta_{\mathsf{split}}(\mathcal{Z}, \widehat{\mathcal{F}}, \widehat{\mathcal{T}}, k)$. Our invariant is that in every hybrid h and for every session j, the input/output to the jth session of $\mathcal{F}$ is indistinguishable from that induced by $\mathcal{Z}^*_j$ above. This is trivially true in hybrid 0. Assume the invariant holds in hybrid $h-1$. Hybrid h differs only in whether the h-th session of $\mathcal{F}$ is serviced by $\mathcal{F}$ or $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$. We have that $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ and $\mathcal{F}$ induce indistinguishable input/output views when interacting with $\mathcal{Z}^*_h$, as this view is included in the output of $\mathcal{Z}^*_h$; thus they must also do so against environments whose input sequence to $\mathcal{F}$ is indistinguishable from that of $\mathcal{Z}^*_h$. Therefore the invariant holds for hybrid h. Finally, if the input/output view of $\mathcal{Z}$ is indistinguishable between hybrids 0 and $N(k)$, then it follows that the output of $\mathcal{Z}$ is indistinguishable between these hybrids. This completes the claim, and we have that the multi-session version of $\mathcal{F}$ is not strongly splittable. □

Acknowledgements

The author would like to thank Tal Malkin for helpful discussions surrounding Lemma 7.1, and Manoj Prabhakaran, Hong-Sheng Zhou, and several anonymous referees for various constructive suggestions.

References

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195. IEEE Computer Society, 2004.

[BG93] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer, 1993.

[BMM99] Amos Beimel, Tal Malkin, and Silvio Micali. The all-or-nothing nature of two-party secure computation. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 1999.

[BS05] Boaz Barak and Amit 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.

[C01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In Moni Naor, editor, *FOCS*, pages 136–145. IEEE Computer Society, 2001. Revised version (2005) on Cryptology ePrint Archive: http://eprint.iacr.org/2000/067.

[C07] Ran Canetti. Obtaining universally composable security: Towards the bare bones of trust. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 88–112. Springer, 2007.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.

[CK91] Benny Chor and Eyal Kushilevitz. A zero-one law for boolean privacy. *SIAM J. Discrete Math.*, 4(1):36–47, 1991.

[CKL06] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503. ACM, 2002.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In Luca Trevisan, editor, *FOCS*, pages 541–550. IEEE Computer Society, 2010.

[CPS07] Ran Canetti, Rafael Pass, and abhi shelat. Cryptography from sunspots: How to use an imperfect reference string. In *FOCS*, pages 249–259. IEEE Computer Society, 2007.

[DNO10] Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi. On the necessary and sufficient assumptions for UC computation. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2010.

[GK92] Oded Goldreich and Hugo Krawczyk. Sparse pseudorandom distributions. *Random Struct. Algorithms*, 3(2):163–174, 1992.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304. ACM, 1985.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 323–341. Springer, 2007.

[HMU07] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. *Tatra Mountains Mathematical Publications*, 37:93–103, 2007. Short version in Proc. MORAVIACRYPT 2005.

[HNRR06] Danny Harnik, Moni Naor, Omer Reingold, and Alon Rosen. Completeness in two-party secure computation: A computational view. *J. Cryptology*, 19(4):521–552, 2006.

[HUM09] Dennis Hofheinz, Dominique Unruh, and Jörn Müller-Quade. Polynomial runtime and composability. Cryptology ePrint Archive, Report 2009/023, 2009. http://eprint.iacr.org/2009/023.

[IL89] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography (extended abstract). In *FOCS*, pages 230–235. IEEE, 1989.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591. Springer, 2008.

[K07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2007.

[K00] Joe Kilian. More general completeness theorems for secure two-party computation. In *STOC*, pages 316–324. ACM, 2000.

[KKK+11] Jonathan Katz, Aggelos Kiayias, Ranjit Kumaresan, abhi shelat, and Hong-Sheng Zhou. From impossibility to completeness for deterministic two-party SFE. Unpublished manuscript, 2011.

[KKMO00] Joe Kilian, Eyal Kushilevitz, Silvio Micali, and Rafail Ostrovsky. Reducibility and completeness in private computations. *SIAM J. Comput.*, 29(4):1189–1208, 2000.

[KL11] Dafna Kidron and Yehuda Lindell. Impossibility results for universal composability in public-key models and with fixed inputs. *J. Cryptology*, 24(3):517–544, 2011.

[KLP05] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In Harold N. Gabow and Ronald Fagin, editors, *STOC*, pages 644–653. ACM, 2005.

[K11] Gunnar Kreitz. A zero-one law for secure multi-party computation with ternary outputs. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 382–399. Springer, 2011.

[L04] Yehuda Lindell. Lower bounds for concurrent self composition. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 203–222. Springer, 2004.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In Michael Mitzenmacher, editor, *STOC*, pages 179–188. ACM, 2009.

[MMY06] Tal Malkin, Ryan Moriarty, and Nikolai Yakovenko. Generalized environmental security from number theoretic assumptions. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 343–359. Springer, 2006.

[MPR10] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. A zero-one law for cryptographic complexity with respect to computational UC security. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 595–612. Springer, 2010.

[N91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[P03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 160–176. Springer, 2003.

[PR08] Manoj Prabhakaran and Mike Rosulek. Cryptographic complexity of multi-party computation problems: Classifications and separations. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 262–279. Springer, 2008.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In László Babai, editor, *STOC*, pages 242–251. ACM, 2004.

A Between Splittability & Strong Unsplittability

We give several examples of functionalities which are neither splittable nor strongly unsplittable, for different reasons:

Example: neither noticeable nor negligible. The following functionality has "fluctuating" behavior as a function of the security parameter, even though it is *uniform* in the sense that its internal code does not depend on the security parameter:

Let f be a one-way function, and consider an $\mathcal{F}$ which does the following. On input $x \in \{0,1\}^*$ from Alice, compute $y = f(x)$. If $|x|$ is a tower-of-twos ($2^{2^{2^{\cdots}}}$), then give y to Bob; otherwise give x to Bob. Then $\mathcal{F}$ is not splittable because no efficient $\mathcal{T}$ can succeed against environments who provide random inputs to $\mathcal{F}$ with tower-of-twos length. However, the "nontrivial" behavior of $\mathcal{F}$ is out of reach for most values of the security parameter k (input lengths that are a tower-of-twos length are either exponential in k, or logarithmic in k, in which case the OWF can be inverted in polynomial time), so $\mathcal{F}$ is splittable for infinitely many values of k. This example was previously observed in [MPR10].

Example: arms race. The following functionality does not allow either party to have a fixed winning strategy in the splittability "game."

Let f be a one-way function and consider a functionality $\mathcal{F}$ that does the following: Upon receiving input $(x, 1^s)$ from Alice and 1^t from Bob, it computes $y = f(x)$. If $s > t$, it gives $(y, 1^s)$ as output to Bob, but if $t \geq s$, it gives $(x, y, 1^s)$ as output to Bob. For any fixed $\mathcal{T}$, there exists a polynomial bound (in k) on the maximum length of 1^t it sends to $\mathcal{F}$ in the first activation. Then a $\mathcal{Z}$ which picks a random x and sends $(x, 1^s = 1^{t+1})$ to $\mathcal{F}$ on behalf of Alice "wins" the splittability game against this $\mathcal{T}$, since $\mathcal{T}$ must now invert the one-way function. Similarly, for any fixed $\mathcal{Z}$ there exists a polynomial bound on the length of 1^s that it sends to $\mathcal{F}$. A $\mathcal{T}$ which uses $1^t = 1^{s+1}$ can "win" the splittability game against this $\mathcal{Z}$ because it obtains Alice's entire input. Such a functionality $\mathcal{F}$ admits an "arms race" between $\mathcal{T}$ and $\mathcal{Z}$.[13]

Note that this $\mathcal{F}$ **cannot** be represented as a circuit family in which all inputs are bounded in length by a fixed polynomial $p(k)$ in the input parameter k. The discussion above crucially uses the fact that there is no *a priori* limit to the length of inputs that the parties can give. Limiting the input length allows a successful splitting strategy, by always providing 1^t as large as allowed. For this reason, such "arms race" behavior seems to require a functionality that is outside the class of functionalities we consider in this work (defined formally in Section 2.2).

Example: uniform vs. non-uniform. The following functionality exhibits non-trivial behavior that only non-uniform parties can access. An *evasive* set [GK92] is a non-empty set X in the complexity class P, such that no *uniform* PPT machine can output an element of X except with negligible probability.

Let f be a one-way function and X be an evasive set. Consider a functionality $\mathcal{F}$ which does the following on input $(x, z) \in \{0,1\}^k \times \{0,1\}^k$ from Alice: Compute $y = f(x)$. If $z \in X$, then give output y to Bob; otherwise, give output (x, y) to Bob. The functionality is not splittable, because the splittability definition permits a non-uniform environment that can choose a random string $x \in \{0,1\}^k$, and has an element $z \in X$ hard-coded. Any splitting strategy $\mathcal{T}$ would therefore be required to invert the one-way function. On the other hand, $\mathcal{F}$ is not strongly unsplittable because a uniform environment cannot give an element of X to $\mathcal{F}$ to access its non-trivial behavior. Then $\mathcal{T}$ will always obtain the input x of Alice and its successful strategy is straight-forward.

As in the case of the negligible vs. noticeable gap, this gap can be essentially mitigated by considering a notion of *UC-realizable via non-uniform protocol.* The only reason we restrict $\mathcal{Z}$ in the strong unsplittability definition to be a uniform machine is because it is used as a subroutine in a protocol.

[13] If $\mathcal{F}$ also gives 1^t as output to Alice, then it is strongly unsplittable for reasons unrelated to the one-way function.

B Variants of Splittability and their Equivalence

Because of some technical subtleties that arise in our construction, we introduced the notions of L/R-strong-unsplittability. Here we show that for the case of secure function evaluation, these notions are equivalent to the simpler definition of strong unsplittability. We also show an example functionality for which the notions differ.

Definition B.1. *A functionality $\mathcal{F}$ is a* **secure function evaluation (SFE)** *functionality if it does the following (where f_A and f_B are deterministic functions):*

1. *On input x from Alice, if a value of y has previously been recorded, then sample a random $r \leftarrow \{0,1\}^k$ and give output $f_A(x,y,r)$ to Alice and $f_B(x,y,r)$ to Bob and stop responding. If a value of x has previously been recorded, then stop responding. Otherwise, internally record x and give output $\perp$ to both parties.*

2. *On input y from Bob, if a value of x has previously been recorded, then sample a random $r \leftarrow \{0,1\}^k$ and give output $f_A(x,y,r)$ to Alice and $f_B(x,y,r)$ to Bob and stop responding. If a value of y has previously been recorded, then stop responding. Otherwise, internally record y and give output $\perp$ to both parties.*

Notably, our formulation of an SFE functionality prescribes an output for both parties when the first party gives an input. This is slightly non-standard; typically an SFE functionality is non-reactive in the sense that it does not give output (except a notification to the adversary) until receiving inputs from both parties, as in [KKK+11]. However, the formulation here is in line with the class of functionalities (Section 2.2) to which our results apply.

Lemma B.2. *The notions of strong unsplittability, L-strong-unsplittability, and R-strong-unsplittability are equivalent for SFE functionalities.*

Proof. Suppose $\mathcal{F}$ is strongly unsplittable. Then the environment $\mathcal{Z}^*_{\mathcal{F}}$ samples inputs x and y to give to $\mathcal{F}$. Sending each of these inputs to $\mathcal{F}$ requires two activations of $\mathcal{F}$. To show that $\mathcal{Z}^*_{\mathcal{F}}$ is also successful in the L-strong-unsplittability definition, we need only ensure that $\mathcal{Z}^*_{\mathcal{F}}$ gives y first.

When $\mathcal{Z}^*_{\mathcal{F}}$ gives input y and is interacting with $\mathcal{F}$, both parties will receive an empty output. In $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$, the input y will go to $\mathcal{F}_R$. In order to give an empty output to Alice, the machine $\mathcal{T}$ must provide some input y' to $\mathcal{F}_L$. Now $\mathcal{F}_R$ does not leak its internal state, and $\mathcal{T}$ is the only one who has influenced $\mathcal{F}_L$'s internal state. So in this step $\mathcal{T}$ receives no more information than in the (plain) splittability interaction. Then after $\mathcal{Z}^*_{\mathcal{F}}$ provides input x to $\mathcal{F}_L$, the machine $\mathcal{T}$ receives output, but that functionality has no need for further internal state. So again $\mathcal{T}$ receives no more information than in the (plain) splittability interaction. $\mathcal{T}$'s overall behavior is exactly the same as in the (plain) splittability interaction, so again the environment successfully distinguishes $\mathcal{F}$ from the $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$ (with leaking internal state) with noticeable probability.

The reverse direction follows trivially: L/R-strong-unsplittability always implies (plain) strong unsplittability. □

A functionality that is strongly unsplittable but L/R-splittable. The notions of L/R-splittability and plain splittability are not equivalent for all functionalities. Consider the following (pathological) functionality $\mathcal{F}$, which does the following on security parameter k: It first chooses a random string $s \leftarrow \{0,1\}^k$ and random bit $b \leftarrow \{0,1\}$. It waits for input (t_0, b_0) from Alice and (t_1, b_1) from Bob. If $s \notin \{t_0, t_1\}$ then $\mathcal{F}$ gives output b to both parties. Otherwise, if $s = t_i$ for some $i \in \{0,1\}$, it gives output b_i to both parties. (If both parties send $t_i = s$, then the functionality gives output b_0 to both).

Intuitively, $\mathcal{F}$ simply provides a fair coin toss. It is only with negligible probability that either party can guess the secret value s to force the output to be b_i. In fact, it is not hard to see that $\mathcal{F}$ is **equivalent** to the fair coin-toss functionality (there is a UC-secure protocol for either functionality, using the other as a setup). Similarly, $\mathcal{F}$ can be seen to be strongly unsplittable, via the functionality that uses $(0^k, 0)$ as input for both parties and then checks whether both parties receive the same output.

However, in the L-splittability interaction, $\mathcal{T}$ receives the internal state of $\mathcal{F}_L$, including the secret value s. It can then receive the fair coin b generated from $\mathcal{F}_R$, and then send input (s, b) to $\mathcal{F}_L$. Thus $\mathcal{T}$ is able to make both parties' outputs match. This $\mathcal{T}$ demonstrates that $\mathcal{F}$ is L-splittable (the functionality is symmetric with respect to Alice and Bob, and so a symmetric argument holds for R-splittability).

We note that in this example, the difference between L- and plain splittability appears to be an artifact of the **code** of $\mathcal{F}$ rather than the **behavior** of $\mathcal{F}$. In this example, it was crucial that s was chosen before the functionality received both inputs. Indeed, for all purposes which don't involve $\mathcal{F}$'s internal state, $\mathcal{F}$ is equivalent to a fair coin-toss functionality (which is L- and R-strongly-unsplittable). The L/R-splittability properties of $\mathcal{F}$ also appear to be very sensitive to minor changes in $\mathcal{F}$'s behavior. For this reason we offer the following conjecture:

Conjecture B.3. *If $\mathcal{F}$ is strongly unsplittable, then $\mathcal{F}$ is equivalent (in the sense of UC-secure reductions) to a functionality that is L-strongly-unsplittable.*

If true, this conjecture would imply that every strongly unsplittable functionality is complete, as well as eliminate the second case of our main construction.

C Unifying Existing Results

Several previous results have proved the completeness of various setup functionalities. In this section, we show how our strong unsplittability characterization can "explain" and unify all these disparate completeness results.

Common random string (CRS) and variants. The first functionality to be shown complete for UC security is the common random string (CRS), in [CLOS02]. It is trivial to see that a CRS is strongly unsplittable. The environment simply activates the setup and checks whether both parties receive the same output. In any $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$, we have that $\mathcal{F}_L$ and $\mathcal{F}_R$ give completely independent outputs regardless of $\mathcal{T}$, so the parties' outputs agree only with probability $1/2^n$ (n is the length of the reference string).[14]

The *multi-string* model [GO07] captures situations in which many parties generate common random strings, and a functionality enforces that a majority of them are honestly generated. Similarly, the *sunspots model* [CPS07] allows a reference string to be sampled from an adversarially-influenced distribution. The setup functionality in each of these models crucially depends on the adversary's ability to interact with it, and thus it is beyond the scope of our characterization. Additionally in the sunspots model, completeness can only be proven with respect to environments that influence the setup functionality in non-degenerate ways.

[14] A similar argument applies to common reference strings selected from any distribution X for which $\Pr[x \leftarrow X; x' \leftarrow X : x \neq x']$ is noticeable.

Trusted hardware tokens & signature cards. Katz [K07] proposed a variant of the UC framework in which parties have access to trusted hardware tokens. Although not often grouped with other setup assumptions, these tokens can be modeled via an ideal functionality and thus be considered within our classification.

A simplified formulation of the trusted hardware token functionality is as follows: It takes as input the description of a Turing machine M from Alice and notifies Bob that the token is ready. Thereafter, Bob can repeatedly give inputs x and receive the corresponding value $M(x)$. In [K07], the functionality even allows M to have a persistent state, but we do not need to exploit that capability to demonstrate strong unsplittability.

To see that this hardware-token functionality is strongly unsplittable, consider an environment $\mathcal{Z}^*_{\mathcal{F}}$ that chooses random string s and lets $M(\cdot) = F(s, \cdot)$, where F is a pseudorandom function. It gives input M to Alice, waits for a notification from Bob, then chooses a random string x as input for Bob and simply checks whether Bob receives the correct output $F(s, x)$. In any splitting strategy, $\mathcal{T}$ can query the pseudorandom function (sent to $\mathcal{F}_L$) on polynomially many inputs of its choice. With overwhelming probability, $\mathcal{T}$ will never query the function on the input x chosen by the environment. By the pseudorandomness of F, the correct value $M(x) = F(s, x)$ is pseudorandom given $\mathcal{T}$'s view. When $\mathcal{T}$ sends a Turing machine M' to $\mathcal{F}_R$, it is therefore only with negligible probability then that $M'(x) = M(x)$.[15]

In a related work, Hofheinz *et al.* show the completeness of a special kind of hardware token called a **signature card** [HMU07]. The analysis of the signature card functionality is even easier since the functionality itself generates verification keys that are publicly announced. Following the same argument as for the CRS functionality, we see that these tokens are strongly unsplittable (surprisingly, for reasons unrelated to their ability to compute signatures).

Unsplittable deterministic finite functionalities. Maji *et al.* [MPR10] show that every deterministic, finite-memory functionality that is not useless (*i.e.*, not splittable) is in fact complete. Intuitively, a functionality whose input/output alphabet and internal memory is of constant size cannot be in the space between splittable and strongly unsplittable. More formally, in the case of non-reactive functionalities (*i.e.*, secure function evaluation), a complete characterization of splittability is given by Prabhakaran & Rosulek [PR08]. In fact, the characterization is proven in a way that is quite amenable to the strong unsplittability definition. For all unsplittable non-reactive functionalities, they demonstrate a *fixed* environment that can distinguish between $\mathcal{F}$ and *any* $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$. In the case where the functionality is finite, one can easily see that the environment has a constant distinguishing bias.

For the case of reactive functionalities, [MPR10] explicitly contains an argument reminiscent of strong unsplittability. They essentially show that if a finite $\mathcal{F}$ is not splittable, then an environment which sends inputs uniformly at random can determine (with bias $\Theta(1)$) a predicate $P(x)$ on the first input used by Alice in the first activation. There also exist inputs x_0 and x_1 satisfying $P(x_0) \neq P(x_1)$ which induce identical outputs for Bob in the first activation. The argument in [MPR10] is combinatorial, using an understanding of such functionalities as finite automata. From this we can see that $\mathcal{F}$ is strongly unsplittable by an environment $\mathcal{Z}^*_{\mathcal{F}}$ that chooses random $b \leftarrow \{0, 1\}$, instructs Alice to send x_b in the first activation, and thereafter chooses random inputs

[15] In [K07], Alice is not notified of Bob's accesses to the functionality, to fully model the fact that a hardware token is a physical object that cannot "phone home" to its creator. Technically, this puts it outside of the class of functionalities we consider in this work. Still, our construction works for the $\mathcal{Z}^*_{\mathcal{F}}$ we demonstrate here. More formally, we can consider a hybrid interaction where instead of accessing $\mathcal{F}_{\mathsf{ideal}}$, the adversary can answer his own queries with random responses. Then the adversary never interacts with $\mathcal{F}_{\mathsf{ideal}}$ during the execution of the virtual-$\mathcal{F}$ subprotocol, so the security argument goes through.

for all the parties. Because the view of $\mathcal{T}$ is independent of b in the first activation, the environment will detect the splitting strategy with constant bias.