Universal Composability from Essentially Any Trusted Setup

Mike Rosulek*

May 14, 2011

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).

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

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 clearly highly desirable, so it is of utmost importance to understand how it can be achieved.

Unfortunately, the news is bleak for those hoping to achieve universal composability. The requirement is simply too demanding to be achieved for every task. Canetti's UC framework [C05] 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, 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 by Prabhakaran & Rosulek [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 almost-total characterization for when a 2-party trusted setup is complete. Our condition is relatively easy to check for a candidate setup, and it applies uniformly to completely arbitrary functionalities (i.e., possibly randomized, reactive, with behavior depending on the security parameter).

When our new classification is compared with the existing characterization of uselessness, we see that **nearly every 2-party setup is either useless or complete**. We can explicitly identify those trusted setups with intermediate power, and we argue that such setups are highly unnatural.

Outline. We revisit the complete characterization of uselessness from Prabhakaran & Rosulek [PR08]. In that work, a property of functionalities called **splittability** was introduced, and it was proved that $\mathcal{F}$ is useless if and only if $\mathcal{F}$ is splittable. To get an idea of the splittability definition,

imagine a party accessing two independent instances of a functionality $\mathcal{F}$ — one in the role of Alice and the other as Bob. Then, informally, $\mathcal{F}$ is splittable if there is a "synchronization strategy" for this party to make these independent instances together appear like a single instance.

Our characterization is based not on splittability, but on a natural variant that we call **strong unsplittability** (defined formally in Section 3). Briefly, $\mathcal{F}$ is strongly unsplittable if there is a single way for reliably distinguishing a true single instance of $\mathcal{F}$ from *any* "synchronization strategy" carried out between two instances of $\mathcal{F}$. Importantly, both of these splittability definitions apply to *arbitrary* functionalities in the UC framework; in particular, no special form is required of the functionality. Then our main technical contribution is the following:

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

A precise statement of the theorem is given in Section 3; it involves a slight technical modification to the strong unsplittability defined above. 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). This intractability assumption is necessary for a completeness characterization such as ours (Section 7).

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 leverage strong unsplittability to construct a commitment protocol in two steps. Recall that the simulator for a UC-secure commitment protocol has two main tasks: to extract the committed value from a corrupt sender, and to give an equivocal commitment to a corrupt receiver. As the first step in our proof, we use the strong unsplittability property to construct a commitment protocol that satisfies the UC-equivocation property but is otherwise only standalone secure (Sections 4–5). Then in Section 6 we "compile" this intermediate protocol into a fully UC-secure commitment protocol.

In Section 7 we show that a different slight variant of strong unsplittability is also *necessary* for completeness. Thus we provide an almost-total characterization[1] of which setups are complete. We see that the only setup functionalities whose power is unclassified are those which are neither splittable nor (basically) strongly unsplittable. As the names suggest, the gap between these two properties is narrow. In Section 3 and Appendix A we argue that functionalities in this gap are highly unnatural. Thus, we informally summarize the power of trusted setups by saying that every "natural" functionality 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].

[1] We make every effort to avoid cumbersome phrases like "[almost-]complete characterization of completeness."

1.2 Related Work

In a similarly motivated work, Maji, Prabhakaran, and Rosulek [MPR10] showed that — among *deterministic* functionalities whose internal state and input/output alphabets were finite — every functionality is either useless or complete. Their approach relied heavily on deriving combinatorial criteria for such functionalities, expressed as a finite automata, whereas the statement of our classification treats functionalities as black-boxes and therefore applies to essentially any functionality that can be expressed in the UC framework. Another closely related work is that of Lin, Pass, and Venkitasubramaniam [LPV09], who develop a framework that unifies many previously-known completeness results. Their completeness condition requires one to develop a "puzzle interaction" with an explicit *trapdoor* and satisfying a *statistical simulation* property. In contrast, our completeness result requires an interaction needing only a *distinguisher*. Furthermore, since our work focuses exclusively on setup functionalities (and not more general relaxations of the UC framework), our criterion is much less open-ended and arguably easier to apply to a candidate setup.

Dichotomies in the cryptographic power of functionalities have been demonstrated previously in other security settings or restricted to small classes of functionalities [CK91, BMM99, HNRR06, MPR10, K11]; however ours is the first work to consider the full range of arbitrary functionalities allowed by the UC framework. In particular, completeness of reactive functionalities was previously considered only in [MPR10], and only for a very restricted class of reactive functionalities.

2 Preliminaries

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

2.1 Universal Composability

We assume some familiarity with the framework of universally composable (UC) security; for a full treatment, see [C05]. 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.[2] 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 as-is 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] It is challenging to formulate a definition of polynomial runtime for single components of a complex interaction. See [HUMQ09] for a full discussion of the issues involved.

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.[3] For simplicity, we assume that C_k receives k bits of the functionality's internal state (including random tape), 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.[4]

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 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 $\mathcal{A}$:

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

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 random 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*, first introduced by Prabhakaran & Rosulek [PR08]. They showed that splittability completely characterizes uselessness (i.e., the existence of a UC-secure protocol in the plain model). Our main result is to show that a natural variant of the splittability condition also provides an almost-total characterization of completeness.

[3] This requirement is not without loss of generality (see Appendix A), but is also implicit in [CLOS02].

[4] 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.

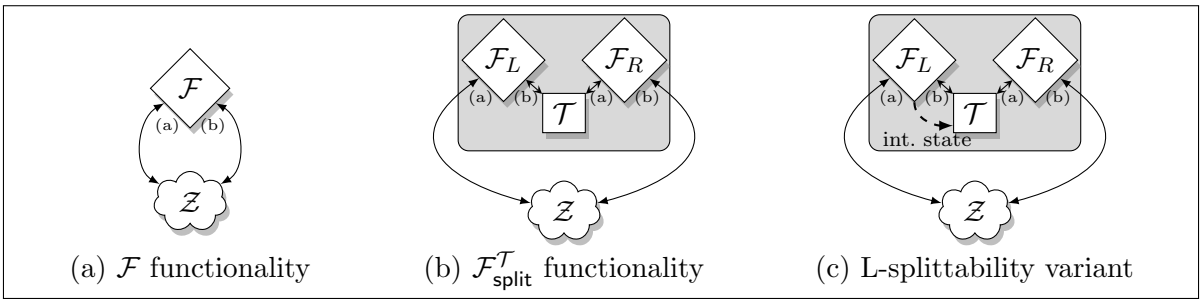
3.1 Splittability and the Statement of our Main Theorem

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 $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ to be the 2-party functionality which behaves as follows (Figure 1b):

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

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

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



(a) $\mathcal{F}$ functionality (b) $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ 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.

Definition 1 ([PR08])**.** *Call an environment* **suitable** *if it does not interact with the adversary except to immediately deliver all outputs from the functionality.*[5]

Then a functionality $\mathcal{F}$ is **splittable** *if there exists an admissible $\mathcal{T}$ such that for all suitable environments $\mathcal{Z}$, $\Delta_{\mathsf{split}}(\mathcal{Z}, \mathcal{F}, \mathcal{T}, k)$ is negligible in k.*

Splittability provides a complete characterization of *uselessness:*

Theorem 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.*

Our classification of complete functionalities is based on the following variant of splittability:

Definition 3. *A functionality $\mathcal{F}$ is* **strongly unsplittable** *if there exists a suitable, uniform*[6] *environment $\mathcal{Z}$ such that for all admissible $\mathcal{T}$, $\Delta_{\mathsf{split}}(\mathcal{Z}, \mathcal{F}, \mathcal{T}, k)$ is noticeable in k.*

[5]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.

[6]Eventually in our result, this environment is used as a subroutine in a protocol. In the UC framework, protocols are typically restricted to be uniform machines, but if this restriction is relaxed then we can allow $\mathcal{Z}$ in this definition to also be non-uniform.

Ideally, our main result would be that every strongly unsplittable functionality is complete. However, imagine a protocol designed to access an ideal instance of $\mathcal{F}$ in a way that mimics the splittability interaction, so that $\mathcal{F}$'s strong unsplittability can be applied in the security proof. In a protocol, each activation of $\mathcal{F}$ is *decoupled* from everything else; whereas in the splittability interaction, the three sub-components of $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ all execute as one *atomic* action from the environment's point of view. This technical complication requires us to consider the following slightly stronger definition (see also the discussion in Section 4.1):

If in $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ we let $\mathcal{T}$ receive the internal state of $\mathcal{F}_L$ (resp. $\mathcal{F}_R$) after every activation and before the first activation (Figure 1c), then we obtain the notions of **L-splittability** and **L-strong-unsplittability** (resp. R-splittability, R-strong-unsplittability). While this difference seems like a significant advantage for $\mathcal{T}$, 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-unsplittability. For example, the 3 notions are equivalent for the standard formulation of secure function evaluation (SFE) functionalities (see Appendix A.1). As another example, the ideal commitment functionality $\mathcal{F}_{\mathsf{com}}$ is R-splittable (assuming we identify Alice as the sender), since the sender already knows the entire internal state of $\mathcal{F}_{\mathsf{com}}$ at all times.

We can now formally state our main result in terms of these splittability notions:

Main Theorem (formal). *$\mathcal{F}$ is complete if the* SHOT *assumption is true and any one 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.[7]

To show that $\mathcal{F}$ is complete, we show that there is a UC-secure commitment protocol in the $\mathcal{F}$-hybrid model. Then the completeness of $\mathcal{F}$ follows from the well-known completeness of the ideal commitment functionality $\mathcal{F}_{\mathsf{com}}$ [CLOS02]. We construct such a commitment protocol in two steps.[8] First, we use $\mathcal{F}$ to construct an intermediate protocol we call a *UC-equivocal* commitment (we do so in Section 4 for case 1, and in Section 5 for case 2 of the main theorem). Then in Section 6 we prove that these UC-equivocal commitment protocols can be used to construct a fully UC-secure commitment protocol.

3.2 Interpreting Splittability & Strong Unsplittability

In Appendix B we show that the previous completeness results involving setups falling within the class of functionalities we consider can all be understood using our new characterization, by showing that the setup in question is L- or R-strongly-unsplittable.

There is a gap in our understanding of functionalities, consisting of functionalities which are neither splittable nor strongly unsplittable; and indeed, there exist (admittedly contrived) functionalities in that gap which are neither useless nor complete.[9] We give concrete examples of such functionalities in Appendix A.2, but briefly describe the two primary reasons a functionality can fall into this gap:

[7] We can allow both $\mathcal{T}$ machines to be non-uniform if we permit UC protocols to be non-uniform.

[8] For convenience we describe our constructions as single-bit commitment protocols, but all are easily extended to multi-bit constructions by letting the underlying standalone-secure commitment protocol be multi-bit.

[9] There is also an apparent gap corresponding to strongly unsplittable functionalities which are neither L-splittable nor L-strongly-unsplittable, etc. However, we are unsure whether such functionalities exist.

- 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).[10] An analogous gap also appears in an otherwise complete dichotomy of Harnik et al. [HNRR06]. Such fluctuating behavior could be explicitly linked to the security parameter (say, in a functionality that implements coin-tossing for even values of the security parameter and does nothing for odd values), but it can also manifest in a functionality whose code ignores the security parameter (an example is given in Appendix A.2). Thus it would not suffice to restrict our result to functionalities whose code ignores the security parameter.

- 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. An example is given in Appendix A.2 of such a functionality, which essentially gives an advantage to whichever player can supply a longer input. However, that example is outside the class of functionalities considered in this work (we require a functionality to have an *a priori* upper bound on the size of inputs it accepts from the parties). We do not know whether similar examples exist within the scope of our results.

We argue that any "natural" functionality will be either splittable or strongly unsplittable (similarly, either L-splittable or L-strongly-unsplittable, and so on). If this is indeed a correct interpretation, then when combined with the characterization of uselessness from [PR08], we have that **every "natural" functionality is either useless or complete**, under the SHOT assumption.

4 UC-Equivocal Commitment from L/R-Strong-Unsplittability

In this section we show that any R- (or, by symmetry, L-) strongly-unsplittable functionality $\mathcal{F}$ can be used to construct a certain kind of commitment protocol. The resulting protocol only has a UC simulation 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 $\mathcal{F}_{\mathsf{com}}$. From this it follows that $\mathcal{F}$ is complete.

4.1 Overview

In a UC-equivocal commitment, an honest sender must be able to convince the receiver to accept its decommitment in the reveal phase. Similarly, the simulator must also be allowed to convince the receiver to accept a (possibly equivocated) decommitment. All this must be possible while still preventing a cheating sender (in the real-process interaction) from forcing the receiver to accept an equivocated decommitment.

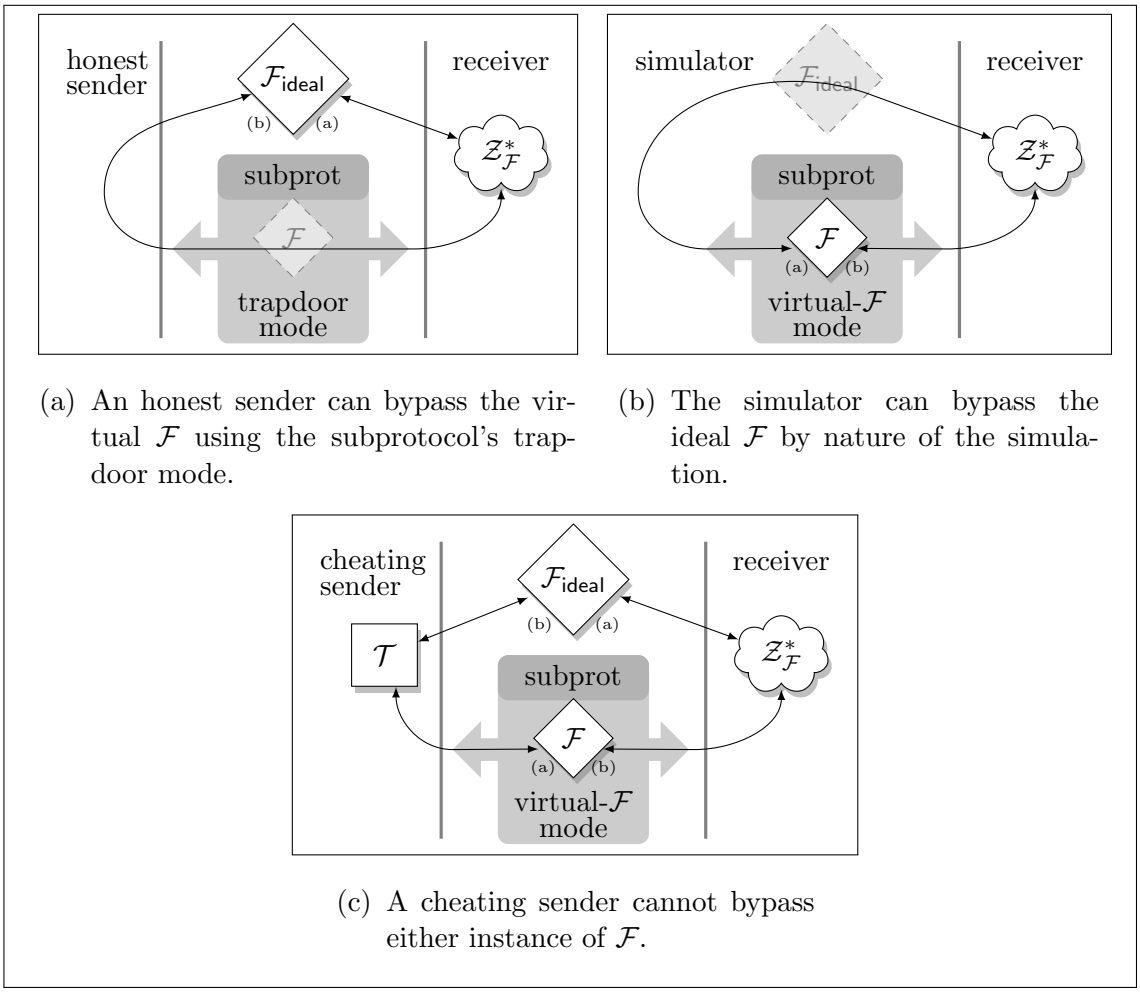
We use strong unsplittability to give an honest sender and the simulator an advantage over a cheating sender. The commit phase of our UC-equivocal protocol is essentially just a commitment under a statistically binding, standalone-secure commitment protocol. The key part of our protocol's reveal phase is the following subprotocol:

[10] We note that this gap may be essentially mitigated by considering relaxed notions of *infinitely-often* splittability and *infinitely-often* strong unsplittability, which yield corresponding notions of infinitely-often useless and infinitely-often complete.

Virtual-$\mathcal{F}$ subprotocol. This subprotocol is essentially a standalone-secure protocol for $\mathcal{F}$. The subprotocol is nominally in **virtual-$\mathcal{F}$ mode**, where it carries out activations of a virtual instance of $\mathcal{F}$, taking inputs and giving outputs to/from the two parties.

However, the subprotocol has an additional mode which we call its **trapdoor mode.** Let C denote the transcript of the standalone commitment in the commit phase, and σ denote the (non-interactive) decommitment value. Suppose the sender is trying to decommit to the value b. Both parties provide C and b as input to the subprotocol, and the sender further provides an input σ. If σ is a valid opening of C to b, then the subprotocol allows the sender to *bypass* the virtual instance of $\mathcal{F}$. Namely, the sender can directly obtain the receiver's inputs to the virtual $\mathcal{F}$, and the sender can force outputs of its choice for the receiver on behalf of the virtual $\mathcal{F}$.

We use the virtual-$\mathcal{F}$ subprotocol in the following way. Suppose $\mathcal{F}$ is strongly unsplittable, with $\mathcal{Z}^*_{\mathcal{F}}$ the fixed distinguishing environment guaranteed from the definition. The parties access an ideal instance of $\mathcal{F}$, which we denote $\mathcal{F}_{\mathsf{ideal}}$ to avoid confusion. The parties also run the virtual-$\mathcal{F}$ subprotocol. The receiver simulates an instance of $\mathcal{Z}^*_{\mathcal{F}}$. Conceptually, these components correspond to the splittability interaction, where $\mathcal{F}_{\mathsf{ideal}}$ corresponds to $\mathcal{F}_L$, the virtual $\mathcal{F}$ corresponds to $\mathcal{F}_R$, and the receiver intuitively corresponds to a machine $\mathcal{T}$. We observe the following properties (summarized in Figure 2):



(a) An honest sender can bypass the virtual $\mathcal{F}$ using the subprotocol's trapdoor mode.

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

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

Figure 2: Intuition behind the reveal phase of $\Pi^{\mathcal{F}}_{\mathsf{EqCom1}}$ and its security properties.

- A sender who can open the standalone commitment C to the claimed value b can activate the trapdoor mode to "bypass" the virtual $\mathcal{F}$. Then by relaying inputs/outputs through the trapdoor mode of the subprotocol between $\mathcal{F}_{\mathsf{ideal}}$ and the receiver ($\mathcal{Z}_{\mathcal{F}}^*$), it can achieve the effect of $\mathcal{Z}_{\mathcal{F}}^*$ interacting with a *single instance* of $\mathcal{F}$.

- A simulator by rule is allowed to bypass the ideal instance $\mathcal{F}_{\mathsf{ideal}}$. By routing inputs/outputs around $\mathcal{F}_{\mathsf{ideal}}$ between the receiver ($\mathcal{Z}_{\mathcal{F}}^*$) and the virtual $\mathcal{F}$ instance, it can also achieve the effect of $\mathcal{Z}_{\mathcal{F}}^*$ interacting with a single instance of $\mathcal{F}$. Because the virtual-$\mathcal{F}$ subprotocol is operating in its normal virtual-$\mathcal{F}$ mode, the standalone commitment C does not need to be openable to the claimed value b.

- A cheating sender cannot bypass the $\mathcal{F}_{\mathsf{ideal}}$ instance, as it must operate within the real-process interaction. If the standalone commitment C cannot be opened to the claimed value b, then the sender cannot activate the trapdoor mode of the subprotocol. Intuitively, the sender is forced to play the role of a synchronizing strategy $\mathcal{T}$ in the definition of $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$.

The definition of $\mathcal{Z}_{\mathcal{F}}^*$ is that it can distinguish between the two cases of interacting with a single instance of $\mathcal{F}$, or interacting with $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$ for any $\mathcal{T}$. 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 $\mathcal{F}_{\mathsf{split}}^{\mathcal{T}}$.

Technical considerations. We outline some important technical considerations that affect the design of our UC-equivocal commitment protocol. First, our virtual-$\mathcal{F}$ subprotocol only has standalone security, so to apply any of its properties may require using a rewinding simulation. If the virtual-$\mathcal{F}$ 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. For this purpose we have the virtual-$\mathcal{F}$ subprotocol share $\mathcal{F}$'s internal state between the two parties using a non-malleable secret sharing (NMSS) scheme, which was proposed for exactly this purpose [IPS08].

The last important technicality is one we alluded to earlier. The activations of the virtual-$\mathcal{F}$ 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 "bypassing" one of the instances of $\mathcal{F}$ ($\mathcal{F}_{\mathsf{ideal}}$ or the virtual $\mathcal{F}$) more subtle. For instance, when the receiver gives an input to the virtual-$\mathcal{F}$ 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 virtual-$\mathcal{F}$ 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 (see Figure 3c,d).

Similarly, consider the case where the simulator is attempting to "bypass" $\mathcal{F}_{\mathsf{ideal}}$. 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 opportunity to make a "round-trip" to the virtual-$\mathcal{F}$ subprotocol to determine the appropriate response to simulate on behalf $\mathcal{F}_{\mathsf{ideal}}$. The only way for the simulator to *immediately* give the correct response is if it already knows the internal state of the virtual $\mathcal{F}$ (see Figure 3a,b). For this reason, we let the subprotocol give this information to the sender, even in its normal virtual-$\mathcal{F}$ mode. Since this internal state

information is also available to a cheating sender, we require $\mathcal{F}$ to be not just strongly unsplittable but R-strongly-unsplittable.

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

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

The functionality $\widetilde{\mathcal{F}}_{\mathsf{virt\text{-}1}}$ for simulating activations of $\mathcal{F}$, with trapdoor mode. Let $\mathsf{NMSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ be a non-malleable secret sharing scheme that can support messages of length $2k$, and let COM be a standalone-secure, plain-model commitment protocol with non-interactive opening phase. Then $\widetilde{\mathcal{F}}_{\mathsf{virt\text{-}1}}$ is the non-reactive, 2-party ideal functionality defined as follows, with global security parameter k:

1. $\widetilde{\mathcal{F}}_{\mathsf{virt\text{-}1}}$ waits for an input of the form $(a, x, q, b, C, \sigma, S_1)$ from the **sender**, and an input of the form (a', y, b', C', S_2) from the **receiver**.

2. If S_1 and S_2 are not both empty and $\mathsf{Reconstruct}(S_1, S_2) = \perp$, then output $\perp$ to both parties and halt. Similarly, if $b' \neq b$ or $C' \neq C$ or $a \neq a'$, then output $\perp$ to both parties and halt.

3. If C is a transcript of the commitment phase of COM, and σ is a valid opening of C to the value b, then we say that the functionality is in **trapdoor mode.**

   (a) If S_1 and S_2 are both empty, then let $(S_1', S_2') \leftarrow \mathsf{Share}(0^{2k})$. Give output (S_1', S_2') to the sender and S_2' to the receiver.

   (b) Otherwise, let $(S_1', S_2') \leftarrow \mathsf{Share}(0^{2k})$. If $a = $ B-IN, then give output (y, S_1', S_2') to the sender and output S_2' to the receiver. If $a \in \{$B-OUT, A-IN$\}$ then give output (S_1', S_2') to the sender and output (q, S_2') to the receiver.

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

   (a) If S_1 and S_2 are both empty, then sample an initial state S for $\mathcal{F}$ (including its random tape). Let $(S_1', S_2') \leftarrow \mathsf{Share}(S \| 0^k)$, and give output (S_1', S) to the sender and S_2' to the receiver.

   (b) Otherwise, let $S \| \hat{y} = \mathsf{Reconstruct}(S_1, S_2)$.

   (c) 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.

   (d) 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 of $\mathcal{F}$ 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.

Standalone-secure protocol. Under the SHOT assumption, there exists a standalone-secure protocol $\Pi_{\mathsf{virt\text{-}}\mathcal{F}\text{-}1}$ for $\widetilde{\mathcal{F}}_{\mathsf{virt\text{-}1}}$.

4.3 UC-Equivocal Commitment

Let COM be a statistically-binding, standalone-secure commitment protocol with non-interactive opening phase. Let $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$ 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 $\Pi^{\mathcal{F}}_{\mathsf{EqCom1}}$ 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. Both parties initialize a virtual $\mathcal{F}$, by the sender providing input $(\text{B-IN}, \perp, \perp, b, C, \sigma, \epsilon)$ and receiver providing input $(\text{B-IN}, \perp, b, C, \epsilon)$ to an instance of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$. The receiver gets output (S'_1, S'_2) and sender gets output S'_2. Both parties update local variables $S_1 := S'_1$ or $S_2 := S'_2$, appropriately.
   (b) Whenever $\mathcal{Z}^*_{\mathcal{F}}$ generates a command x to send to Alice, the parties do the following ($\mathsf{IdealActivation}$, 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 $(\text{A-IN}, \perp, q, b, C, \sigma, S_1)$ and the receiver provides input $(\text{A-IN}, \perp, b, C, S_2)$ to a fresh instance of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$. If the instance aborts or outputs $\perp$, or if $\mathcal{F}_{\mathsf{ideal}}$ is activated during the execution of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$, then both parties abort.
      iii. The $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$ 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) Whenever $\mathcal{Z}^*_{\mathcal{F}}$ generates a command y to send to Bob, the receiver notifies the sender to initiate an instance of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$ as follows ($\mathsf{VirtActivation}$, see Figure 3c):
      i. The sender provides input $(\text{B-IN}, \perp, \perp, b, C, \sigma, S_1)$ and the receiver provides input $(\text{B-IN}, y, b, C, S_2)$ to a fresh instance of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$. If the instance aborts or outputs $\perp$, or if $\mathcal{F}_{\mathsf{ideal}}$ is activated during the execution of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$, then both parties abort.

ii. The $\Pi_{\text{virt-}\mathcal{F}\text{-1}}$ 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: Using $\Pi_{\text{virt-}\mathcal{F}\text{-1}}$, 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 (REVEAL, b). Otherwise the receiver aborts.

4.4 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 $\Pi_{\mathsf{EqCom1}}^{\mathcal{F}}$) is that of interacting with a single instance of $\mathcal{F}$, just as in the splittability definition.[11] 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 (REVEAL, b).

We formalize the security of $\Pi_{\mathsf{EqCom1}}^{\mathcal{F}}$ in the following lemmas.

Lemma 4. *$\Pi_{\mathsf{EqCom1}}^{\mathcal{F}}$ has a UC simulator in the case that the receiver is corrupt (i.e., an equivocating simulator).*

Proof. Consider an interaction in $\Pi_{\mathsf{EqCom1}}^{\mathcal{F}}$ 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.

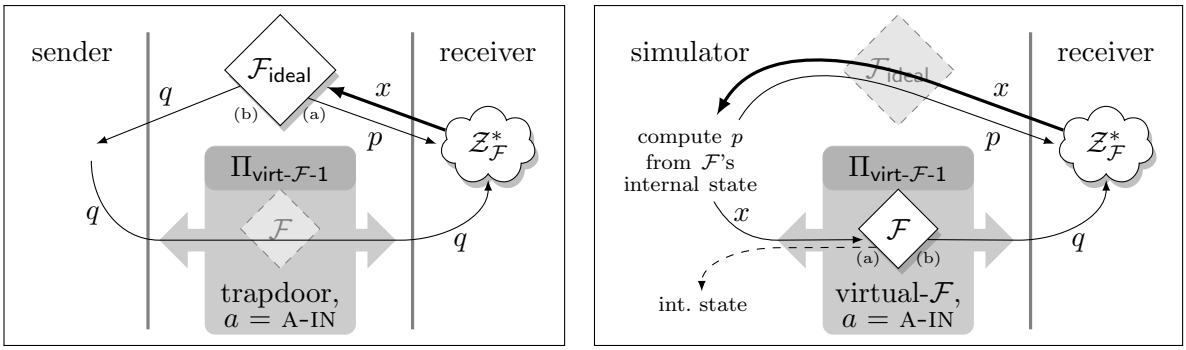
Suppose the ith invocation of the $\Pi_{\text{virt-}\mathcal{F}\text{-1}}$ subprotocol does not abort or result in output of $\perp$. Then by the standalone security of $\Pi_{\text{virt-}\mathcal{F}\text{-1}}$, there is a (possibly rewinding) simulator that can extract an *effective* input of the adversary, $(\hat{a}, \hat{y}, \hat{b}, \hat{C}, \hat{S}_2)$. The honest party's output from $\Pi_{\text{virt-}\mathcal{F}\text{-1}}$ is computationally indistinguishable from the prescribed output of $\widetilde{\mathcal{F}}_{\text{virt-1}}$ 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 $\Pi_{\text{virt-}\mathcal{F}\text{-1}}$ 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.[12] Similarly from the definition of $\widetilde{\mathcal{F}}_{\text{virt-1}}$ we see that $\hat{b} = b$ and $\hat{C} = C$ and $\hat{a} = a$. We conclude that the adversary's *effective input* has the form $(a, \cdot, b, C, S_2)$ — 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 $\mathsf{IdealActivation}$ and $\mathsf{VirtActivation}$ as part of the prescribed protocol. Our simulation differs from the real interaction in that these subprotocols are carried out as follows:

$\mathsf{SimVirtActivation}$**:** (Figure 3d) Same as $\mathsf{VirtActivation}$, except that the simulator uses input $(\text{B-IN}, \perp, \perp, b, C, \perp, S_1)$ in the first execution of $\Pi_{\text{virt-}\mathcal{F}\text{-1}}$ (so $\perp$ is given instead of σ). Then the simulator receives

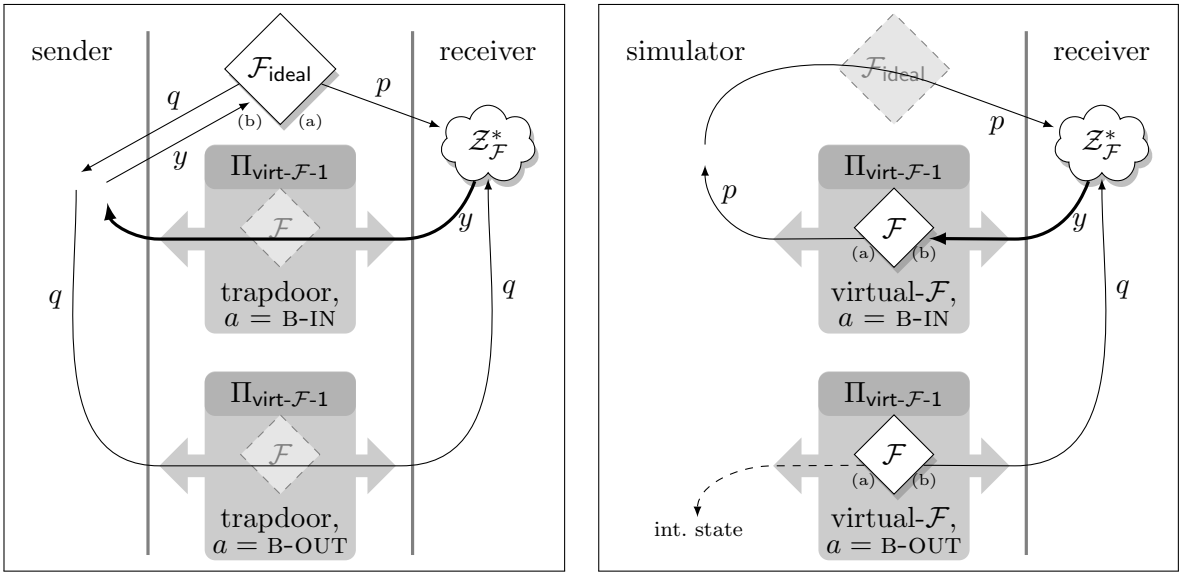
[11] Both parties are honest, but an external adversary may still interact with $\mathcal{F}_{\mathsf{ideal}}$. This is the step where we must use the fact that $\mathcal{F}_{\mathsf{ideal}}$ ignores all activations from the adversary.

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



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

(b) Simulator behavior, when receiver activates $\mathcal{F}_{\text{ideal}}$ 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 $\Pi^{\mathcal{F}}_{\text{EqCom1}}$ and its security proof. Grayed-out instances of $\mathcal{F}$ are being "bypassed" (by the trapdoor mode of $\Pi_{\text{virt-}\mathcal{F}\text{-1}}$ or by virtue of the simulation).

output p (among others) from $\Pi_{\text{virt-}\mathcal{F}\text{-1}}$, which it immediately delivers to the receiver on behalf of $\mathcal{F}_{\text{ideal}}$. The simulator uses input $(\text{B-OUT}, \bot, \bot, b, C, \bot, S_1)$ in the second execution of $\Pi_{\text{virt-}\mathcal{F}\text{-1}}$. The simulator no longer receives S'_2 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 $(\text{A-IN}, x, \bot, b, C, \bot, S_1)$ to the execution of $\Pi_{\text{virt-}\mathcal{F}\text{-1}}$ (so $\bot$ 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 $\mathcal{F}_{\mathsf{ideal}}$. 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 $\mathcal{F}_{\mathsf{ideal}}$. 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:

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 output of Alice, output of 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 $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$ in trapdoor 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 $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$ 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 $\widetilde{\mathcal{F}}_{\mathsf{virt\text{-}1}}$ 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 $\widetilde{\mathcal{F}}_{\mathsf{virt\text{-}1}}$ to output $\perp$, with overwhelming probability. Thus the views of the receiver in the two hybrids are indistinguishable, by the standalone security of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$.

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 trapdoor invocation of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$. 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 trapdoor mode of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$ 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 $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$, which it immediately delivers to the receiver on behalf of $\mathcal{F}_{\mathsf{ideal}}$. In the second invocation of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$ 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 $\widetilde{\mathcal{F}}_{\mathsf{virt\text{-}1}}$ is identical in the two hybrids, for any (effective) receiver input including the correct S_2 value. As above, the hybrids are indistinguishable.[13]

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 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 5. $\Pi^{\mathcal{F}}_{\mathsf{EqCom1}}$ *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* $(\text{REVEAL}, 1-b)]$ *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

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

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 $\Pi^{\mathcal{F}}_{\mathsf{EqCom1}}$, 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 $\Pr[\text{receiver outputs } (\textsc{reveal}, 1-b)]$ 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 $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$ protocol is replaced by the receiver simulating an ideal instance of $\widetilde{\mathcal{F}}_{\mathsf{virt}\text{-}1}$ and running the (possibly rewinding) simulator with the sender. The two hybrids are indistinguishable by the standalone security of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$. We are trying to bound $\Pr[\text{receiver outputs } (\textsc{reveal}, 1-b)]$, 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 $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$ instance. Thus, without loss of generality we assume that the sender does not interact with $\mathcal{F}_{\mathsf{ideal}}$ during any instance of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$.[14] Similarly, the honest receiver's simulated instance of $\mathcal{Z}^*_{\mathcal{F}}$ is not activated at any time during the execution of a $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}1}$ 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 $\widetilde{\mathcal{F}}_{\mathsf{virt}\text{-}1}$ in each main loop of $\Pi^{\mathcal{F}}_{\mathsf{EqCom1}}$ 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 $\widetilde{\mathcal{F}}_{\mathsf{virt}\text{-}1}$ 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).

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 $(\textsc{reveal}, 1-b)$, 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.

5.1 Overview

Our approach is similar to that of the previous section — 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

[14] 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.

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 "trapdoor mode" of the virtual-$\mathcal{F}$ subprotocol. In this subprotocol, the trapdoor 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. With this trapdoor, we have the following observations (Figure 4):

- An honest sender who can activate the trapdoor mode of the virtual-$\mathcal{F}$ 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 trapdoor) 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 "bypass" an instance of $\mathcal{F}$. The technical complications described in the previous section 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

Similar to the previous section, we define an ideal functionality to specify the properties required of 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, and let COM be a standalone-secure, plain-model commitment protocol with non-interactive opening phase. Then $\widetilde{\mathcal{F}}_{\mathsf{virt\text{-}2}}$ is the non-reactive, 2-party ideal functionality defined as follows, with global security parameter k:

1. $\widetilde{\mathcal{F}}_{\mathsf{virt\text{-}2}}$ waits for an input of the form $(a, x, b, C, \sigma, S_1)$ from the **sender**, and an input of the form (a', y, b', C', S_2) from the **receiver**.
2. If S_1 and S_2 are empty, then sample an initial state S for $\mathcal{F}$ (including random tape). Set $(S'_1, S'_2) \leftarrow \mathsf{Share}(S)$, then halt and give output $(\perp, S'_1, S)$ to the sender and $(\perp, S'_2)$ to the receiver.
3. If $\mathsf{Reconstruct}(S_1, S_2) = \perp$ or $b \neq b'$ or $C \neq C'$ or $a \neq a'$, then halt and give output $\perp$ to both parties. Otherwise, set $S = \mathsf{Reconstruct}(S_1, S_2)$.
   (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 **trapdoor mode**. In trapdoor 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 $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}2}$ for $\widetilde{\mathcal{F}}_{\mathsf{virt}\text{-}2}$.

5.3 UC-Equivocal Commitment

Let COM be a statistically-binding, standalone-secure commitment protocol with non-interactive opening phase. Let $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}2}$ 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 $\Pi^{\mathcal{F}}_{\mathsf{EqCom2}}$ proceeds as follows, with security parameter k:

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

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) Both parties initialize a virtual $\mathcal{F}$, by the sender providing input $(\textsc{a-in}, \perp, b, C, \sigma, \epsilon)$ and receiver providing input $(\textsc{a-in}, \perp, b, C, \epsilon)$ to an instance of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}2}$. The receiver gets output $(\perp, S'_1, S)$ and sender gets output $(\perp, S'_2)$. Both parties update local variables $S_1 := S'_1$ or $S_2 := S'_2$, appropriately. The sender gives S to $\mathcal{T}_2$ as the internal state from $\mathcal{F}_R$.

   (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 $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}2}$. The sender provides input $(\textsc{a-in}, x', b, C, \sigma, S_1)$ and the receiver provides input $(\textsc{a-in}, \perp, b, C, S_2)$. If $\mathcal{F}_{\mathsf{ideal}}$ is activated during the execution of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}2}$, both parties abort.

iii. The $\Pi_{\text{virt-}\mathcal{F}\text{-2}}$ 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:

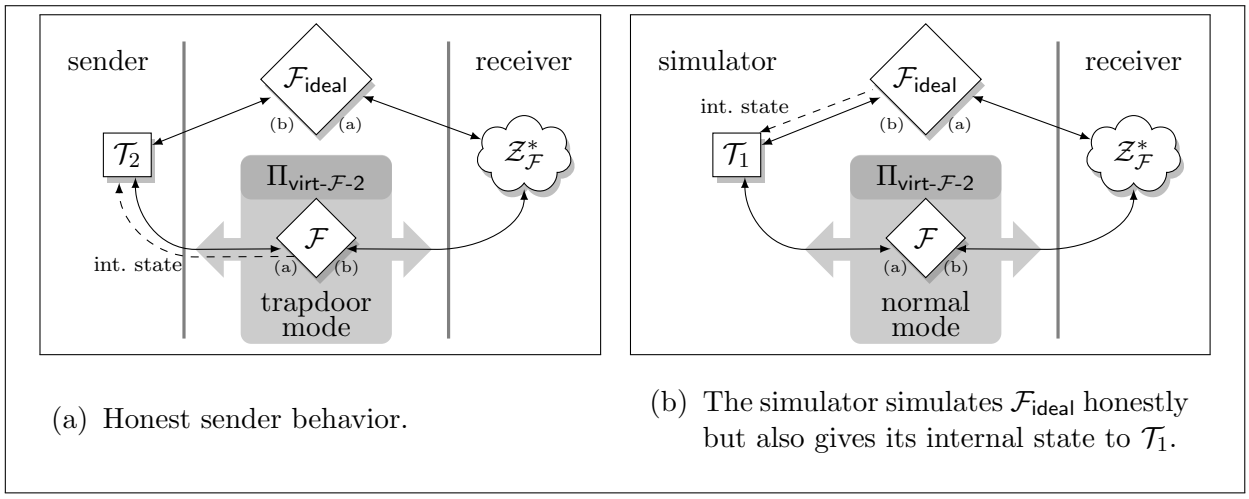
i. The parties initiate a new instance of $\Pi_{\text{virt-}\mathcal{F}\text{-2}}$. The sender provides input $(\text{B-IN}, \perp, b, C, \sigma, S_1)$ and the receiver provides input $(\text{B-IN}, y, b, C, S_2)$. If $\mathcal{F}_{\text{ideal}}$ is activated during the execution of $\Pi_{\text{virt-}\mathcal{F}\text{-2}}$, both parties abort.

ii. The $\Pi_{\text{virt-}\mathcal{F}\text{-2}}$ 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 $\mathcal{F}_{\text{ideal}}$, 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.



(a) Honest sender behavior.

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

Figure 4: Interactions in $\Pi_{\text{EqCom2}}^{\mathcal{F}}$ and its security proof.

5.4 Security Properties

Correctness of the protocol can be seen by inspection, similar to the previous section. We now sketch the security of $\Pi_{\text{EqCom2}}^{\mathcal{F}}$ in the following lemmas.

Lemma 6. *$\Pi_{\text{EqCom2}}^{\mathcal{F}}$ 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 $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}2}$ protocol, the sender gives its input to a simulated instance of $\widetilde{\mathcal{F}}_{\mathsf{virt}\text{-}2}$, and then runs the (possibly rewinding) simulator for the $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}2}$ 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 $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}2}$.

Hybrid 2: Same as Hybrid 1, except that each time through the main loop of $\Pi^{\mathcal{F}}_{\mathsf{EqCom2}}$, the multiple activations of $\widetilde{\mathcal{F}}_{\mathsf{virt}\text{-}2}$ 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 $\widetilde{\mathcal{F}}_{\mathsf{virt}\text{-}2}$.

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 $\Pi^{\mathcal{F}}_{\mathsf{EqCom2}}$, 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 $\mathcal{F}$.[15]

Hybrid 4: Same as Hybrid 3, except that the plain (virtual) instance of $\mathcal{F}$ is replaced with successive executions of the $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}2}$ 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 $\widetilde{\mathcal{F}}_{\mathsf{virt}\text{-}2}$ is identical whether $\widetilde{\mathcal{F}}_{\mathsf{virt}\text{-}2}$ is in trapdoor 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 $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}2}$) is our final UC simulation, since it does not require the value b until the reveal phase. □

Lemma 7. $\Pi^{\mathcal{F}}_{\mathsf{EqCom2}}$ *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 trapdoor mode of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}2}$. As before, we repeatedly apply the standalone security of $\Pi_{\mathsf{virt}\text{-}\mathcal{F}\text{-}2}$ and the specification of $\widetilde{\mathcal{F}}_{\mathsf{virt}\text{-}2}$ 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

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

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 $(\text{REVEAL}, 1-b)$] is negligible. □

6 Full-Fledged UC Commitment from Equivocal Commitment

We now show how our UC-equivocal commitment protocols can be "compiled" into a full-fledged UC commitment protocol. In this protocol we use the following additional properties of the $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ protocols from the previous sections:

- Both $\Pi^{\mathcal{F}}_{\mathsf{EqCom}}$ protocols use the same commit phase; namely, commit to b under COM and give a zero-knowledge proof of knowledge of the opening to the commitment. Therefore, for both of these protocols 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.
- For both protocols, 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 a commitment protocol as in the previous sections. Our final commitment protocol $\Pi^{\mathcal{F}}_{\mathsf{com}}$ is defined as follows:

Commit phase: When the sender receives input (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 the sender providing input $(x, C_1, \ldots, C_k, \sigma_1, \ldots, \sigma_k)$ and the receiver providing input $(C_1, \ldots, C_k)$:
   (a) On input $(x, C_1, \ldots, C_k, \sigma_1, \ldots, \sigma_k)$ from the sender and $(C'_1, \ldots, C'_k)$ from the receiver, verify that for all i, $C_i = C'_i$ and σ_i is a valid decommitment of C_i to value x_i. If not, output $\perp$ to both parties.
   (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 the sender providing input (r, α), and the receiver providing input (r, β):

   (a) On input (r, α) from the sender and (r', β) from the receiver: if $\mathsf{Reconstruct}(\alpha, \beta) = \perp$ or $r \neq r'$ 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 8. *$\Pi^{\mathcal{F}}_{\mathsf{com}}$ 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 (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 5. 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 4 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 at the beginning of the interaction the sender chooses a random value $i^* \leftarrow \{1, \ldots, k\}$. 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 4 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 the SHOT Assumption and 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 used strong unsplittability (and several minor variants) as the basis to show that certain functionalities are complete. Ideally, we would like to prove that strong unsplittability is necessary for completeness; that is, if $\mathcal{F}$ is not strongly unsplittable then $\mathcal{F}$ is not complete. However, the condition "$\mathcal{F}$ is not strongly unsplittable" is not easily amenable to such an approach.

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 negligible *for infinitely many* values of the security parameter. A typical approach to showing that $\mathcal{F}$ is not complete would be to consider a hypothetical protocol for, say, oblivious transfer in the $\mathcal{F}$-hybrid model and then derive a contradiction. But such a hypothetical protocol can invoke many instances of $\mathcal{F}$, and one must presumably apply the aforementioned condition to each of them. To do so would require a hybrid argument, with slightly different environments — and thus potentially a different $\mathcal{T}$ and subset of security value parameters — in each hybrid. There may not be an infinite number of security parameter values for which *every* step of the hybrid argument succeeds.

We can, however, prove the necessity of two very slight variants of strong unsplittability:

Lemma 9. *If the multi-session version of $\mathcal{F}$ is not strongly unsplittable, then $\mathcal{F}$ is not complete.*

Proof. Let $\widehat{\mathcal{F}}$ denote the multi-session version of $\mathcal{F}$. 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 $\widehat{\mathcal{F}}^{\mathcal{T}}_{\mathsf{split}}$. 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 10. *If $\mathcal{F}$ is not strongly unsplittable with respect to environments with multi-bit output, then $\mathcal{F}$ is not complete.*

Proof. We show that if the condition in the lemma is met, then the multi-session version of $\mathcal{F}$ is not strongly unsplittable (with respect to environments with single-bit output), as in Lemma 9. 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. □

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. Electronic Colloquium on Computational Complexity (ECCC) TR01-016, 2001. Previous version "A unified framework for analyzing security of protocols" availabe at the ECCC archive TR01-016. Extended abstract in FOCS 2001.

[C05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Revised version of [C01].

[C07] Ran Canetti. Obtaining universally compoable 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 *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.

[HMQU07] 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.

[HUMQ09] Dennis Hofheinz, Dominique Unruh, and Jrn Mller-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.

[KL11] Dafna Kidron and Yehuda Lindell. Impossibility results for universal composability in public-key models and with fixed inputs. *Journal of 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 Understanding and Interpreting our Splittability Definitions

A.1 L/R-splittability

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

Definition 11. *A functionality $\mathcal{F}$ is a* **secure function evaluation (SFE)** *if it does the following. It waits for an input x from Alice and an input y from Bob, and then gives output $f_A(x, y)$ to Alice and $f_B(x, y)$ to Bob, where f_A and f_B are deterministic functions.*

Lemma 12. *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 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 $\mathcal{F}$. 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-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 13. *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.

A.2 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 strongly unsplittable 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 $\mathcal{Z}$.[16]

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).

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

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.

B 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 $\mathcal{F}^{\mathcal{T}}_{\mathsf{split}}$, 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). Of course, read-only access to the functionality's internal state does not give $\mathcal{T}$ any advantage in making $\mathcal{F}_L$ and $\mathcal{F}_R$ generate the same output. Thus, the CRS functionality is both L- and R-strongly-unsplittable as well.[17]

It is desirable for a CRS to be used only with *static* access — i.e., in an "offline" phase the beginning of the protocol. Our constructions do not satisfy this additional requirement. Indeed, it is important to our results that the setup functionality $\mathcal{F}$ is not activated until after the virtual-$\mathcal{F}$ subprotocol is initialized in the reveal phase of our commitment protocol.

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.

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

[17] 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.

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 $M(x)$.[18] 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 $M'(x) = M(x)$. Note that Bob cannot influence the internal state of the functionality (it can forget his inputs after each activation), so the functionality is also R-strongly-unsplittable.

In a related work, Hofheinz et al. show the completeness of a special kind of hardware token called a **signature card** [HMQU07]. 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 is finite 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 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.

[18] 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, suppose $\mathcal{F}$ does not necessarily inform Alice of every activation. If $\mathcal{F}$ is R-strongly unsplittable even when $\mathcal{F}_R$ informs both parties of every activation, then our protocol construction is still secure in the $\mathcal{F}$-hybrid model.