# Universally Composable Security with Global Setup

Ran Canetti* Yevgeniy Dodis† Rafael Pass‡ Shabsi Walfish§

October 3, 2007

**Abstract**

Cryptographic protocols are often designed and analyzed under some *trusted setup* assumptions, namely in settings where the participants have access to global information that is trusted to have some basic security properties. However, current modeling of security in the presence of such setup falls short of providing the expected security guarantees. A quintessential example of this phenomenon is the *deniability* concern: there exist natural protocols that meet the strongest known composable security notions, and are still vulnerable to bad interactions with rogue protocols that use the same setup.

We extend the notion of universally composable (UC) security in a way that re-establishes its original intuitive guarantee even for protocols that use globally available setup. The new formulation prevents bad interactions even with adaptively chosen protocols that use the same setup. In particular, it guarantees deniability. While for protocols that use no setup the proposed requirements are the same as in traditional UC security, for protocols that use global setup the proposed requirements are significantly stronger. In fact, realizing Zero Knowledge or commitment becomes provably impossible, even in the Common Reference String model. Still, we propose reasonable alternative setup assumptions and protocols that allow realizing practically any cryptographic task under standard hardness assumptions *even against adaptive corruptions*.

*canetti@csail.mit.edu. IBM Research, 19 Skyline Dr., Hawthorne, NY 10532 USA.

†dodis@cs.nyu.edu. New York University, Department of Computer Science, 251 Mercer St., New York, NY 10012 USA.

‡rafael@cs.cornell.edu. Cornell University, Department of Computer Science, Ithaca, NY 14853 USA.

§walfish@cs.nyu.edu. New York University, Department of Computer Science, 251 Mercer St., New York, NY 10012 USA.

# Contents

# 1 Introduction

The trusted party paradigm is a fundamental methodology for defining security of cryptographic protocols. The basic idea (which originates in [28]) is to say that a protocol securely realizes a given computational task if running the protocol amounts to "emulating" an ideal process where all parties secretly hand their inputs to an imaginary "trusted party" who locally computes the desired outputs and hands them back to the parties. One potential advantage of this paradigm is its strong "built in composability" property: The fact that a protocol $\pi$ emulates a certain trusted party $\mathcal{F}$ can be naturally interpreted as implying that any system that includes calls to protocol $\pi$ should, in principle, behave the same if the calls to $\pi$ were replaced by ideal calls to the trusted party $\mathcal{F}$.

Several formalizations of the above intuitive idea exist, e.g. [26, 33, 3, 11, 22, 38, 12, 37]. These formalizations vary in their rigor, expressibility, generality and restrictiveness, as well as security and composability guarantees. However, one point which no existing formalism seems to handle in a fully satisfactory way is the security requirements in the presence of "global trusted setup assumptions", such as a public-key infrastructure (PKI) or a common reference string (CRS), where all parties are assumed to have access to some global information that is trusted to have certain properties. Indeed, as pointed out in [35], the intuitive guarantee that "running $\pi$ has the same effect as having access to the trusted party" no longer holds.

As a first indication of this fact, consider the "deniability" concern, namely, allowing party $A$ to interact with party $B$ in a way that prevents $B$ from later "convincing" a third party $C$ that the interaction took place. Indeed, if $A$ and $B$ interact via an idealized "trusted party" that communicates only with $A$ and $B$ then deniability is guaranteed in a perfect, idealized way. Thus, intuitively, if $A$ and $B$ interact via a protocol that emulates the trusted party, then deniability should hold just the same. When the protocol in question uses no global setup, this intuition works, in the sense that emulating a trusted party (in most existing formalisms) automatically implies deniability. However, when global setup is used, this is no longer the case: There are protocols that emulate such a trusted party but do *not* guarantee deniability.

For instance, consider the case of Zero-Knowledge protocols, *i.e.* protocols that emulate the trusted party for the "Zero-Knowledge functionality": Zero-Knowledge protocols in the plain model are inherently deniable, but most Zero-Knowledge protocols in the CRS model are completely *un*deniable whenever the reference string is public knowledge (see [35]). Similarly, most authentication protocols (i.e., most protocols that emulate the trusted party that provides ideally authenticated communication) that use public key infrastructure are not deniable, in spite of the fact that ideal authenticated communication via a trusted party is deniable.

One might think that this "lack of deniability" arises only when the composability guarantees provided by the security model are weak. However, even very strong notions of composability do not automatically suffice to ensure deniability in the presence of global setup. For example, consider the Universal Composability (UC) security model of [12], which aims to achieve the following, very strong composability guarantee:

> *A UC-secure protocol $\pi$ implementing a trusted party $\mathcal{F}$ does not affect any other protocols more than $\mathcal{F}$ does — even when protocols running concurrently with $\pi$ are maliciously constructed.*

When $\mathcal{F}$ is the Zero-Knowledge functionality, this property would seem to guarantee that deniability will hold even when the protocol $\pi$ is used in an arbitrary manner. Yet, even UC-secure ZK protocols that use a CRS are *not* deniable whenever the reference string is globally available. This demonstrates that the UC notion, in its present formulation, does *not* protect a secure protocol $\pi$ from a protocol $\pi'$ that was maliciously designed to interact badly with $\pi$, in the case where $\pi'$ can use the *same setup* as $\pi$.

Deniability is not the only concern that remains un-captured in the present formulation of security in the CRS model. For instance, even UC-secure Zero-Knowledge proofs in the CRS model may not be "adaptively sound" (see [24]), so perhaps a malicious prover can succeed in proving false statements after seeing the CRS, as demonstrated in [1]. As another example, the protocol in [17] for realizing the single-instance commitment functionality becomes *malleable* as soon as *two* instances use the same reference string (indeed, to avoid this weakness a more involved protocol was developed, where multiple commitments can explicitly use the same reference string in a specific way). Note that here, a UC-secure protocol can even affect the security of another UC-secure protocol if both protocols make reference to the same setup.

This situation is disturbing, especially in light of the fact that *some* form of setup is often *essential* for cryptographic solutions. For instance, most traditional two-party tasks cannot be UC-realized with no setup [17, 12, 18], and authenticated communication is impossible without some sort of setup [14]. Furthermore, providing a *globally available* setup that can be used throughout the system is by far the most realistic and convenient way to provide setup.

**A new formalism.** This work addresses the question of how to formalize the trusted-party definitional paradigm in a way that preserves its intuitive appeal even for those protocols that use globally available setup. Specifically, our first contribution is to generalize the UC framework to deal with global setup, so as to explicitly guarantee that the original meaning of "emulating a trusted party" is preserved, even when the analyzed protocol is using the *same setup* as other protocols that may be maliciously and adaptively designed to interact badly with it. In particular, the new formalism called simply generalized UC (GUC) security guarantees deniability and non-malleability even in the presence of global setup. Informally,

> *A GUC-Secure protocol $\pi$ implementing a trusted party $\mathcal{F}$ using some global setup does not affect any other protocols more than $\mathcal{F}$ does — even when protocols running concurrently with $\pi$ are maliciously constructed, and even when all protocols use the same global setup.*

In a nutshell, the new modeling proceeds as follows. Recall that the UC framework models setup as a "trusted subroutine" of the protocol that uses the setup. This implicitly means that the setup is local to the protocol instance using it, and cannot be safely used by any other protocol instance. That modeling, while mathematically sound, certainly does not capture the real-world phenomenon of setup that is set in advance and publicly known throughout the system. The UC with joint state theorem ("JUC Theorem") of [20] allows several instances of specifically-designed protocols to use the same setup, but it too does not capture the case of public setup that can be used by arbitrary different protocols at the same time.

To adequately capture global setup our new formalism models the setup as an additional (trusted) entity that interacts not only with the parties running the protocol, but also with other parties (or, in other words, with the external environment). This in particular means that the setup entity exists not only as part of the protocol execution, but also in the *ideal process,* where the protocol is replaced by the trusted party. For instance, while in the current UC framework the CRS model is captured as a trusted setup entity that gives the reference string only to the adversary and the parties running the actual protocol instance, here the reference string is globally available, i.e. the trusted setup entity also gives the reference string directly to other parties and the external environment. Technically, the effect of this modeling is that now the simulator (namely, the adversary in the ideal process) cannot choose the reference string or know related trapdoor information.

In a way, proofs of security in the new modeling, even with setup, are reminiscent of the proofs of security without setup, in the sense that the only freedom enjoyed by the simulator is to control the local random choices of the uncorrupted parties. For this reason we often informally say that GUC-secure protocols that use only globally available setup are "fully simulatable". We also remark that this modeling is in line with the "non-programmable CRS model" in [35].

One might thus suspect that achieving GUC-security "collapses" down to UC-security *without any setup* (and its severe limitations). Indeed, as a first result we extend the argument of [17] to show that no two-party protocol can GUC-realize the ideal commitment functionality $\mathcal{F}_{com}$ (namely, emulate the trusted party that runs the code of $\mathcal{F}_{com}$ according to the new notion), *even in the CRS model, or in fact with any global setup that simply provides public information.* On the one hand this result is reassuring, since it means that those deniable and malleable protocols that are secure in the (old) CRS model can no longer be secure according to the new notion. On the other hand, this result brings forth the question of whether there exist protocols for commitment (or other interesting primitives) that meet the new notion under *any* reasonable setup assumption. Indeed, the analyses of all existing UC-secure commitment protocols seem to use in an essential way the fact that the simulator has control over the value of the setup information.

**New setup and constructions.** Perhaps surprisingly, we answer the realizability question in the affirmative, in a strong sense. Recall that our impossibility result shows that a GUC protocol for the commitment functionality must rely on a setup that provides the parties with some *private* information. We consider two alternative setup models which provide such private information in a *minimal* way, and show how to GUC-realize practically any ideal functionality in any one of the two models.

The first setup model is reminiscent of the "key registration with knowledge (KRK)" setup from [6], where each party registers a public key with some trusted authority in a way that guarantees that the party can access the corresponding secret key. However, in contrast to [6] where the scope of a registered key is only a single protocol instance (or, alternatively, several instances of specifically designed protocols), here the registration is done once per party throughout the lifetime of the system, and the public key can be used in all instances of all the protocols that the party might run. In particular, it is directly accessible by the external environment.

We first observe that one of the [6] protocols for realizing $\mathcal{F}_{com}$ in the KRK model can be shown to satisfy the new notion, even with the global KRK setup, as long as the adversary is limited to *non-adaptive* party corruptions. (As demonstrated in [19], realizing $\mathcal{F}_{com}$ suffices for realizing *any* "well-formed" multi-party functionality.) However, when adaptive party corruptions are allowed, and the adversary can observe the past internal data of corrupted parties, this protocol becomes insecure. In fact, the problem seems inherent, since the adversary is now able to eventually see *all* the secret keys in the system, even those of parties that were uncorrupted when the computation took place.

Still, we devise a new protocol that realizes $\mathcal{F}_{com}$ in the KRK model even in the presence of adaptive party corruptions, and without any need for data erasures. The high level idea is to use the [17] commitment scheme with a new CRS that is chosen by the parties per commitment. The protocol for choosing the CRS will make use of the public keys held by the parties, in a way that allows the overall simulation to go through even when the same public keys are used in multiple instances of the CRS-generation protocol. Interestingly, our construction does not realize a CRS that is "strong" enough for the original analysis to go through. Instead, we provide a "weaker" CRS, and provide a significantly more elaborate analysis. The protocol is similar in spirit to the coin-tossing protocol of [21], in that it allows the generated random string to have different properties depending on which parties are corrupted. Even so, their protocol is not adaptively secure in our model.

**Augmented CRS.** Next we formulate a new setup assumption, called "augmented CRS (ACRS)" and demonstrate how to GUC-realize $\mathcal{F}_{com}$ in the ACRS model, in the presence of adaptive adversaries. As the name suggests, ACRS is reminiscent of the CRS setup, but is somewhat augmented so as to circumvent the impossibility result for plain CRS. That is, as in the CRS setup, all parties have access to a short reference string that is taken from a pre-determined distribution. In addition, the ACRS setup allows corrupted parties to obtain "personalized" secret keys that are derived from the reference string, their public identities, and some "global secret" that's related to the public string and remains unknown. It is stressed that *only corrupted parties* may obtain their secret keys. This means that the protocol may not include instructions that require knowledge of the secret keys and, therefore, the protocol interface tn the ACRS setup is identical to that of the CRS setup.

The main tool in our protocol for realizing $\mathcal{F}_{com}$ in the ACRS model is a new *identity-based trapdoor commitment (IBTC)* protocol. IBTC protocols are constructed in [2, 39], in the Random Oracle model. Here we provide a construction in the standard model based on one way functions. The construction is secure against adaptive corruptions, and is based on the Feige construction of commitment from Sigma protocols [23], where the committer runs the *simulator* of the Sigma protocol.

**Realizing the setup assumptions.** "Real world implementations" of the ACRS and KRK setups can involve a trusted entity (say, a "post office") that only publicizes the public value. The trusted entity will also agree to provide the secret keys to the corresponding parties upon request, with the understanding that once a party gets hold of its key then it alone is responsible to safeguard it and use it appropriately (much as in the case of standard PKI). In light of the impossibility of a completely non-interactive setup (CRS), this seems to be a minimal "interactiveness" requirement from the trusted entity.

Another unique feature of our commitment protocol is that it guarantees security even if the "global secret" is compromised, as long as this happens *after the commitment phase is completed.* In other words, in order to compromise the overall security, the trusted party has to be *actively malicious during the commitment phase.* This point further reduces the trust in the real-world entity that provides the setup.

Despite the fact that the trusted entity need not be constantly available, and need not remain trustworthy in the long term, it may still seem difficult to provide such an interactive entity in many real-world settings. Although it is impossible to achieve true GUC security with a mere CRS, we observe that the protocols analyzed here do satisfy some notion of security even if the setup entity remains non-interactive (*i.e.* when our ACRS setup functionality is instead collapsed to a standard CRS setup). In fact, although we do not formally prove a separation, protocols proven secure in the ACRS model seem intuitively *more secure* than those of [17, 19] *even when used in the CRS model*! Essentially, in order to simulate information that could be obtained via a real attack on the protocols of [17, 19], knowledge of a "global trapdoor" is required. This knowledge enables the simulator to break the security *of all parties* (including their privacy). On the other hand, simulating the information obtained by real attacks on protocols that are proven secure in the ACRS model merely requires some specific "identity-based trapdoors". These specific trapdoors used by the simulator allow it to break only the security *of corrupt parties who deviate from the protocol*. Of course, when using a CRS setup in "real life" none of these trapdoors are available to anyone, so one cannot actually simulate information obtained by an attacker. Nevertheless, it seems that the actual advantage gained by an attack which *could* have been simulated using the more minimal resources required by protocol simulators in the ACRS model (*i.e.* the ability to violate the security only of corrupt parties, as opposed to all parties) is intuitively smaller.

**A New Composition Theorem.** We present two formulations of GUC security: one formulation is more general and more "intuitively adequate", while the other is simpler and easier to work with. In particular, while the general notion directly considers a multi-instance system, the simpler formulation (called EUC) is closer to the original UC notion that considers only a single protocol instance in isolation. We then demonstrate that the two formulations are equivalent. As may be expected, the proof of equivalence incorporates much of the argumentation involved in the proof of the universal composition theorem. We also demonstrate that GUC security is preserved under universal composition.

**Related work.** Relaxed variants of UC security are studied in [37, 10]. These variants allow reproducing the general feasibility results without setup assumptions other than authenticated communication. However, these results provide significantly weaker security properties than UC-security. In particular, they do not guarantee security in the presence of arbitrary other protocols, which is the focus of this work.

Alternatives to the CRS setup are studied in [6]. As mentioned above, the KRK setup used here is based on the one there, and the protocol for GUC-realizing $\mathcal{F}_{com}$ for non-adaptive corruptions is taken from there. Furthermore, [6] informally discuss the deniability properties of their protocol. However, that work does not address the general concern of guaranteeing security in the presence of global setup. In particular, it adopts the original UC modeling of setup as a construct that is internal to each protocol instance.

In a concurrent work, Hofheinz et. al [30] consider a notion of security that is reminiscent of EUC, with similar motivation to the motivation here. They also formulate a new setup assumption and show how to realize any functionality given that setup. However, their setup assumption is considerably more involved than ours, since it requires the trusted entity to interact with the protocol in an on-line, input-dependent manner. Also, they do not consider adaptive corruptions.

**Future work.** This work develops the foundations necessary for analyzing security and composability of protocols that use globally available setup. It also re-establishes the feasibility results for general computation in this setting. Still, there are several unexplored research questions here.

One important concern is that of guaranteeing *authenticated communication* in the presence of global PKI setup. As mentioned above, this is another example where the existing notions do not provide the expected security properties (e.g., they do not guarantee deniability, whereas the trusted party solution is expressly deniable). We conjecture that GUC authentication protocols (namely, protocols that GUC-realize ideally authentic communication channels) that use a global

PKI setup can be constructed by combining the techniques of [31, 17]. However, we leave full exploration of this problem out of scope for this work.

The notions of key exchange and secure sessions in the presence of global PKI setup need to be re-visited in a similar way. How can universal composability (and, in particular, deniability) be guaranteed for such protocols? Also, how can existing protocols (that are not deniable) be proven secure with globally available setup?

**Organization.** Section 2 outlines the two variants of GUC security, states their equivalence, and re-asserts the UC theorem with respect to GUC secure protocols. Section 3 presents the formal impossibility of realizing $\mathcal{F}_{com}$ in the presence of a globally available common reference string, and highlights the need for alternative setup assumptions. Sections 4 and 5 present new globally available setup assumptions, as well as our protocols for realizing any well-formed functionality. Finally, Section 5.2 describes the efficient construction of a useful tool employed in our protocols.

# 2 Generalized UC Security

Before providing the details for our new security framework, we begin with a high-level overview.

## 2.1 Overview of Generalized UC Security

We now briefly review the concepts behind the original UC framework of [12] (henceforth referred to as "Basic UC") before proceeding to outline our new security frameworks. To keep our discussion at a high level of generality, we will focus on the notion of protocol "emulation", wherein the objective of a protocol $\pi$ is to emulate another protocol $\phi$. Here, typically, $\pi$ is an implementation (such as the actual "real world" protocol) and $\phi$ is a specification (where the "ideal functionality" $\mathcal{F}$ that we wish to implement is computed directly by a trusted entity). Throughout our discussion, all entities and protocols we consider are "efficient" (*i.e.* polynomial time bounded Interactive Turing Machines, in the sense detailed in [13]).

**The Basic UC Framework.** At a very high level, the intuition behind security in the basic UC framework is that any adversary $\mathcal{A}$ attacking a protocol $\pi$ should learn no more information than could have been obtained via the use of a simulator $\mathcal{S}$ attacking protocol $\phi$. Furthermore, we would like this guarantee to be maintained even if $\phi$ were to be used a subroutine of (*i.e.* composed with) arbitrary other protocols that may be running concurrently in the networked environment, and we plan to substitute $\pi$ for $\phi$ in all instances. Thus, we may set forth a challenge experiment to distinguish between actual attacks on protocol $\pi$, and simulated attacks on protocol $\phi$ (referring to these protocols as the "challenge protocols"). As part of this challenge scenario, we will allow adversarial attacks to be orchestrated and monitored by a distinguishing environment $\mathcal{Z}$ that is also empowered to control the inputs supplied to the parties running the challenge protocol, as well as to observe the parties' outputs at all stages of the protocol execution. One may imagine that this environment represents all other activity in the system, including the actions of other protocol sessions that may influence inputs to the challenge protocol (and which may, in turn, be influenced by the behavior of the challenge protocol). Ultimately, at the conclusion of the challenge, the environment $\mathcal{Z}$ will be tasked to distinguish between adversarial attacks perpetrated by $\mathcal{A}$ on the challenge protocol $\pi$, and attack simulations conducted by $\mathcal{S}$ with protocol $\phi$ as the challenge protocol instead. If no environment can successfully distinguish these two possible scenarios, then protocol $\pi$ is said to "UC emulate" the protocol $\phi$.

Specifying the precise capabilities of the distinguishing environment $\mathcal{Z}$ is crucial to the meaning of this security notion. The environment *must* be able to choose the challenge protocol inputs and observe its outputs, in order to enable the environment to capture the behavior of other activity in the network that interacts with the challenge protocol (which may even be used as a subroutine of another network protocol). Of course, we *must* also grant $\mathcal{Z}$ the ability to interact with the attacker (which will be either the adversary, or a simulation), which models the capability of the attacker to coordinate attacks based on information from other network activity in the environment. As demonstrated in [12], granting precisely these capabilities to $\mathcal{Z}$ (even if we allow it to invoke only a *single session* of the challenge protocol) is sufficient to achieve the strong guarantees of *composition theorem*, which states that any arbitrary instances of the $\phi$ that may be running in the network can be safely substituted with a protocol $\pi$ that UC emulates $\phi$. Thus, even if we *constrain* the distinguisher $\mathcal{Z}$ to such interactions with the adversary and a *single session* of the challenge protocol (without providing the ability to invoke other protocols at all), we can already achieve the strong security guarantees we intuitively desired. Notably, although the challenge protocol may invoke subroutines of its own, it was not necessary to grant $\mathcal{Z}$ any capability to interact with such subroutines.

In order to conceptually modularize the design of protocols, the notion of "hybrid models" is often introduced into the basic UC framework. A protocol $\pi$ is said to be realized "in the $\mathcal{G}$-hybrid model" if $\pi$ invokes the ideal functionality $\mathcal{G}$ as a subroutine (perhaps multiple times). (As we will soon see below, the notion of hybrid models greatly simplifies the discussion of UC secure protocols that require "setup".) A high-level conceptual view of UC protocol emulation in a hybrid model is shown in Figure 1.

**Limitations of Basic UC.** Buried inside the intuition behind the basic UC framework is the critical notion that the environment $\mathcal{Z}$ is capable of utilizing its input/output interface to the challenge protocol to mimic the behavior of other (arbitrary) protocol sessions that may be running in a computer network. Indeed, as per the result of [12] mentioned in our discussion above, this would seem to be the case when considering challenge protocols that are essentially "self-contained". Such self-contained protocols, which do not make use of any "subroutines" (such as ideal functionalities) belonging to other protocol sessions, are called *subroutine respecting* protocols – and the basic UC framework models these protocols directly. On the other hand, special considerations would arise if the challenge protocol utilizes (or transmits) information that is also shared by other network protocol sessions. An example of such information would be the use of a global setup, such as a public "common reference string" (CRS) that is reused from one protocol session to the next, or a standard Public Key Infrastructure (PKI). Such shared state is not directly modeled by the basic UC framework discussed above. In fact, the composition theorem of [12] only holds when considering instances of subroutine respecting protocols (which *do not* share any state information with other protocol sessions). Unfortunately, it is impossible to produce UC secure realizations of most useful functionalities without resorting to some setup. However, to comply with the requirements of the UC framework, the setup would have to be done on a per-instance basis. This does not faithfully represent the common realization, where the same setup is shared by all instances. Therefore, previous works handled such "shared state" protocol design situations via a special proof

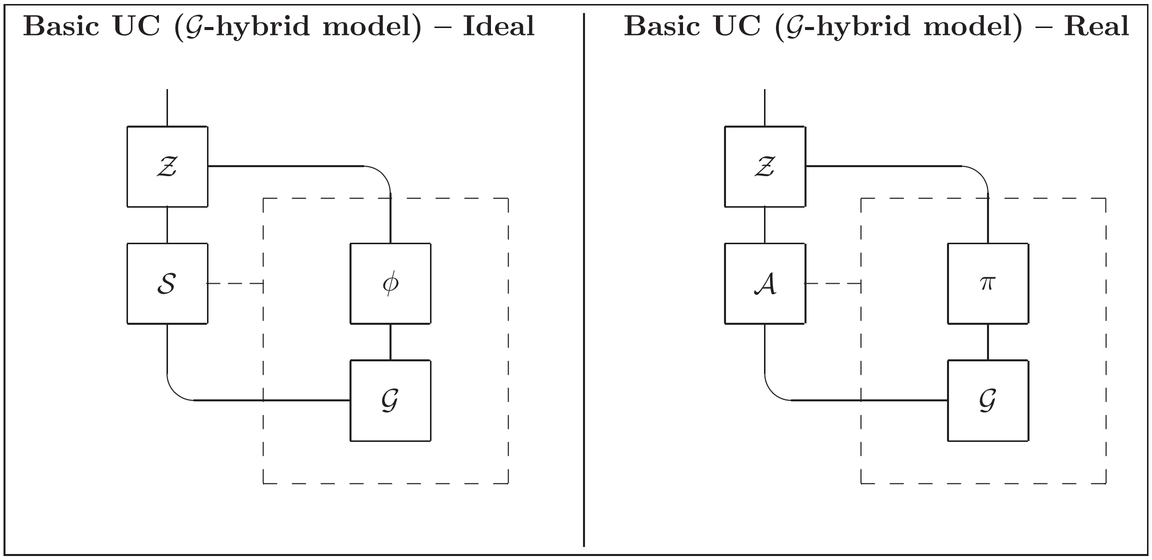


Figure 1: The Basic UC Experiment in the $\mathcal{G}$-hybrid model. A simulator $\mathcal{S}$ attacks a single session of protocol $\phi$ running with an ideal subroutine $\mathcal{G}$, whereas an arbitrary "real" adversary $\mathcal{A}$ attacks a session of $\pi$ running with an ideal subroutine $\mathcal{G}$. The dashed box encloses protocols where $\mathcal{S}$ or $\mathcal{A}$ control the network communications, whereas the solid lines represent a direct Input/Output relationship. (In a typical scenario, $\phi$ would be the ideal protocol for a desired functionality $\mathcal{F}$, whereas $\pi$ would be a practical protocol realizing $\mathcal{F}$, with $\mathcal{G}$ modeling some "setup" functionality required by $\pi$. Observe that the environment can never interact directly with $\mathcal{G}$, and thus, in this particular scenario, $\mathcal{G}$ is never invoked at all in the ideal world since we are typically interested in the case where ideal protocol for $\mathcal{F}$ does not make use of $\mathcal{G}$.)

technique, known as the JUC Theorem [20].

Yet, even the JUC Theorem does not accurately model truly *global* shared state information. JUC Theorem only allows for the construction of protocols that share state *amongst themselves.* That is, an *a-priori* fixed set of protocols can be proven secure if they share state information *only* with each other. No security guarantee is provided in the event that the shared state information is also used by other protocols which the original protocols were not specifically designed to interact with. Of course, malicious entities may take advantage of this by introducing new protocols that use the shared state information if the shared state is publicly available. In particular, protocols sharing global state (*i.e.* using global setups) which are modeled in this fashion may not resist adaptive chosen protocol attacks, and can suffer from a lack of deniability, as we previously mentioned regarding the protocols of [17], [19], and as is discussed in further detail in Section 3.2.

**The Generalized UC Framework.** To summarize the preceding discussion, the environment $\mathcal{Z}$ in the basic UC experiment is unable to invoke protocols that share state in any way with the challenge protocol. This limitation is unrealistic in the case of global setup, when protocols share state information with each other (and indeed, it was shown to be impossible to realize UC-secure protocols without resort to such tactics [17, 12, 18]). To overcome this limitation, we propose the

Generalized UC (GUC) framework. The GUC challenge experiment is similar to the basic UC experiment, only with an *unconstrained* environment. In particular, we will allow $\mathcal{Z}$ to actually invoke and interact with arbitrary protocols, and even multiple sessions of its challenge protocol (which may be useful to $\mathcal{Z}$ in its efforts to distinguish between the two possible challenge protocols). Some of the protocol sessions invoked by $\mathcal{Z}$ may share state information with challenge protocol sessions, and indeed, they can provide $\mathcal{Z}$ with information about the challenge protocol that it could not have obtained otherwise. The only remaining limitation on $\mathcal{Z}$ is that we prevent it from directly observing or influencing the network communications of the challenge protocol sessions, but this is naturally the job of the adversary (which $\mathcal{Z}$ directs). Thus, the GUC experiment allows a very powerful distinguishing environment capable of truly capturing the behavior of arbitrary protocol interactions in the network, *even if protocols can share state information with arbitrary other protocols*. Of course, protocols that are GUC secure are also composable (this fact follows almost trivially from a greatly simplified version of the composition theorem proof of [13], the simplifications being due to the ability of the unconstrained environment to directly invoke other protocol sessions rather than needing to "simulate" them internally).

**The Externalized UC Framework.** Unfortunately, since the setting of GUC is so complex, it becomes extremely difficult to prove security of protocols in our new GUC framework. Essentially, the distinguishing environment $\mathcal{Z}$ is granted a great deal of freedom in its choice of attacks, and any proof of protocol emulation in the GUC framework must hold even in the presence of other arbitrary protocols running concurrently. To simplify matters, we observe that in practice protocols which are designed to share state do so only in a very limited fashion (such as via a single common reference string, or a PKI, etc.). In particular, we will model shared state information via the use of "shared functionalities", which are simply functionalities that may interact with more than one protocol session (such as the CRS functionality). For clarity, we will distinguish the notation for shared functionalities by adding a bar (*i.e.* we use $\bar{\mathcal{G}}$ to denote a shared functionality). We call a protocol $\pi$ that *only* shares state information via a single shared functionality $\bar{\mathcal{G}}$ a $\bar{\mathcal{G}}$-*subroutine respecting* protocol. Bearing in mind that it is generally possible to model "reasonable" protocols that share state information as $\bar{\mathcal{G}}$-subroutine respecting protocols, we can make the task of proving GUC security simpler by considering a compromise between the constrained environment of basic UC and the unconstrained environment of GUC. An $\bar{\mathcal{G}}$-*externally constrained* environment is subject to the same constraints as the environment in the basic UC framework, only it is additionally allowed to invoke a single "external" protocol (specifically, the protocol for the shared functionality $\bar{\mathcal{G}}$). Any state information that will be shared by the challenge protocol must be shared via calls to $\bar{\mathcal{G}}$ (*i.e.* challenge protocols are $\bar{\mathcal{G}}$-subroutine respecting), and the environment is specifically allowed to access $\bar{\mathcal{G}}$. Although $\mathcal{Z}$ is once again constrained to invoking a single instance of the challenge protocol, it is now possible for $\mathcal{Z}$ to internally mimic the behavior of multiple sessions of the challenge protocol, or other arbitrary network protocols, by making use of calls to $\bar{\mathcal{G}}$ wherever shared state information is required. Thus, we may avoid the need for JUC Theorem (and the implementation limitations it imposes), by allowing the environment direct access to shared state information (*e.g.* we would allow it to observe the Common Reference String when the shared functionality is the CRS functionality). We call this new security notion Externalized UC (EUC) security, and we say that a $\bar{\mathcal{G}}$-subroutine respecting protocol $\pi$ $\bar{\mathcal{G}}$-EUC-emulates a protocol $\phi$ if $\pi$ emulates $\phi$ in the basic UC sense with respect to $\bar{\mathcal{G}}$-externally constrained environments.

We show that if a protocol $\pi$ $\bar{\mathcal{G}}$-EUC-emulates $\phi$, then it also GUC emulates $\phi$ (and vice versa, provided that $\pi$ is $\bar{\mathcal{G}}$-subroutine respecting).

**Theorem 2.1.** *Let $\pi$ be any protocol which invokes no shared functionalities other than (possibly) $\bar{\mathcal{G}}$, and is otherwise subroutine respecting (*i.e. *$\pi$ is $\bar{\mathcal{G}}$-subroutine respecting). Then protocol $\pi$ GUC-emulates a protocol $\phi$, if and only if protocol $\pi$ $\bar{\mathcal{G}}$-EUC-emulates $\phi$.*

That is, provided that $\pi$ only shares state information via a single shared functionality $\bar{\mathcal{G}}$, if it merely EUC-emulates $\phi$ with respect to that functionality, then $\pi$ is a full GUC-emulation of $\phi$! As a special case, we obtain that all basic UC emulations (which may not share *any* state information) are also GUC emulations.

**Corollary 2.2.** *Let $\pi$ be any subroutine respecting protocol. Then protocol $\pi$ GUC-emulates a protocol $\phi$, if and only if $\pi$ UC-emulates $\phi$.*

The corollary follows by letting $\bar{\mathcal{G}}$ be the null functionality, and observing that the $\bar{\mathcal{G}}$-externally constrained environment of the EUC experiment collapses to become the same environment as that of the basic UC experiment when $\bar{\mathcal{G}}$ is the null functionality. Thus, it is sufficient to prove basic UC security for protocols with no shared state, or $\bar{\mathcal{G}}$-EUC security for protocols that share state only via $\bar{\mathcal{G}}$, and we will automatically obtain the full benefits of GUC security. The proof of the theorem is given in Section 2.2.

Figure 2 depicts the differences in the experiments of the UC models we have just described, in the presence of a single shared functionality $\bar{\mathcal{G}}$ (of course, the GUC framework is not inherently limited to special case of only one shared functionality). In Section 2.2 we elaborate the technical details of our new models, in addition to proving the equivalence of GUC and EUC security.

We are now in a position to state a strong new composition theorem, which will directly incorporate the previous result (that proving EUC security is sufficient for GUC security). Let $\rho$ be an arbitrary protocol (not necessarily subroutine respecting!) which invokes $\phi$ as a sub-protocol. We will write $\rho^{\pi/\phi}$ to denote a modified version of $\rho$ that invokes $\pi$ instead of $\phi$, wherever $\rho$ had previously invoked $\phi$. We prove the following general theorem in Section 2.2 below:

**Theorem 2.3** (Generalized Universal Composition)**.** *Let $\rho, \pi, \phi$ be PPT multi-party protocols, and such that both $\phi$ and $\pi$ are $\bar{\mathcal{G}}$-subroutine respecting, and $\pi$ $\bar{\mathcal{G}}$-EUC-emulates $\phi$. Then $\rho^{\pi/\phi}$ GUC-emulates protocol $\rho$.*

We stress that $\pi$ must merely $\bar{\mathcal{G}}$-EUC-emulate $\phi$, but that the resulting composed protocol $\rho^{\pi/\phi}$ fully GUC-emulates $\rho$, even for a protocol $\rho$ that is not subroutine respecting.

## 2.2 Details of the Generalized UC Framework

We now present more formal details of our new generalized UC (GUC) framework, and discuss its relationship to basic UC Security. (Here we will refer to the formulation of UC security in [13]). We also present a simplified variant of the new notion called Externalized UC (EUC), and prove its equivalence. Finally, we re-assert the universal composition theorem with respect to the new notion. Many of the low-level technical details, especially those that are essentially identical to those of the basic UC framework, are omitted. A full treatment of these details can be found in [13]. In particular, we do not discuss the proper modeling of polynomial runtime restrictions, the

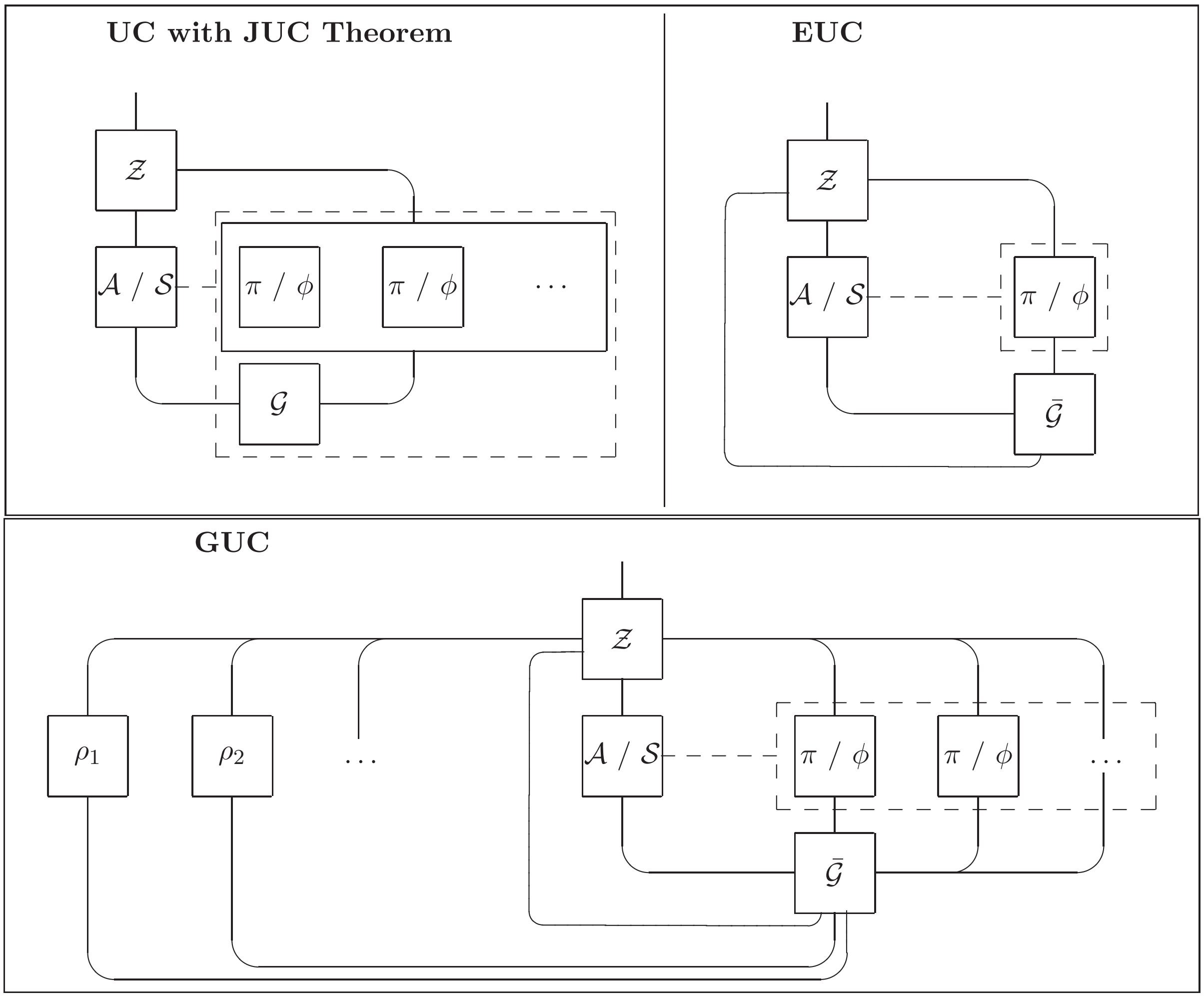


Figure 2: Comparison of models. Using Basic UC with JUC Theorem to share state, only copies of the challenge protocol (or other protocols which may be jointly designed a priori to share $\mathcal{G}$) are allowed to access the common subroutine $\mathcal{G}$, and $\mathcal{Z}$ may only interact with the "multi-session" version of the challenge protocol. In the EUC paradigm, only a single session of the challenge protocol is running, but the shared functionality $\bar{\mathcal{G}}$ it uses is accessible by $\mathcal{Z}$. Finally, in the GUC setting, we see the full generality of arbitrary protocols $\rho_1, \rho_2, \ldots$ running in the network, alongside multiple copies of the challenge protocol. Observe that both $\mathcal{Z}$, and any other protocols invoked by $\mathcal{Z}$ (such as $\rho_1$), have direct access to $\bar{\mathcal{G}}$ in the GUC setting. Intuitively, the GUC modeling seems much closer to the actual structure of networked protocol environments.

order of activations, etc. These issues are handled as in the basic UC framework, which we now briefly review.

### 2.2.1 Basic UC Security

The basic UC framework is built around the notion of *UC emulation*. A protocol is a UC secure *realization* of an *ideal functionality* (which models the security goal), if it UC-emulates the ideal functionality, in the sense that executing the protocol is indistinguishable for an external environment from an interaction with a trusted party running the ideal functionality. Before reviewing the actual "UC experiment" that defines the notion of UC emulation, we first briefly review the basic model of distributed computation, using interactive Turing machines (ITMs). While we do not modify this model, familiarity with it is important for understanding our generalization.

**Systems of ITMs.** To capture the mechanics of computation and communication in computer networks, the UC framework employs an extension of the Interactive Turing Machine (ITM) model [27] (see [13] for precise details on the additional extensions). A computer program (such as for a protocol, or perhaps program of the adversary) is modeled in the form of an ITM (which is an abstract notion). An execution experiment consists of a *system* of ITMs which are instantiated and executed, with multiple instances possibly sharing the same ITM code. (More formally, a system of ITMs is governed by a *control function* which enforces the rules of interaction among ITMs as required by the protocol execution experiment. Here we will omit the full formalisms of the control function, which can be found in [13], and which require only minor modifications for our setting.)

A particular executing ITM instance running in the network is referred to as an ITI (or "ITM Instance"), and we must have a means to distinguish individual ITIs from one another even if they happen to be running identical ITM code. Therefore, in addition to the program code of the ITM they instantiate, individual ITIs are parameterized by a *party ID* ($\mathsf{pid}$) and a *session ID* ($\mathsf{sid}$). We require that each ITI can be uniquely identified by the *identity* pair $\mathsf{id} = (\mathsf{pid}, \mathsf{sid})$, irrespective of the code it may be running. All ITIs running with the same code and session ID are said to be a part of the same *protocol session*, and the party IDs are used to distinguish among the various ITIs participating in a particular protocol session. (By the uniqueness of ITI identities, no party is allowed to participate in more than one protocol session using the same session ID.) For simplicity of exposition, we assume that all $\mathsf{sid}$s are unique, i.e. no two sessions have the same SID. (The treatment can be generalized to the case where the same SID is used by different protocol codes, at the price of somewhat more complicated formalism.) We also refer to a protocol session running with ITM code $\pi$ as an *instance* of protocol $\pi$. ITMs are allowed to communicate with each other via the use of three kinds of I/O tapes: local *input tapes*, local *subroutine output* tapes, and *communication tapes*. The input and subroutine output tapes model "trusted communication", say communication within a single physical computer. The communication tape models "untrusted communication", say communication over an open network. Consequently, writes to the local input tapes of a particular ITI must include both the identity *and the code* of the intended target ITI, and the ITI running with the specified identity must also be running the specified code, or else an error condition occurs. Thus, input tapes may be used to invoke local "trusted" subroutines, and indeed, new ITIs must be introduced into the currently executing system by means of such invocations. That is, if a target ITI with the specified identity does not exist, it is created ("invoked"), and given the specified code. We also require that when an ITI writes to the local subroutine output tape of another ITI, it must provide its own code, and thus these tapes are useful for accepting output from such local "trusted" subroutines. Finally, all "untrusted" communications are passed via the communication tapes, which guarantee neither the code of the intended recipient ITI, nor the code of the sending ITI (but merely their identities).

**The UC Protocol Execution Experiment.** The UC protocol execution experiment is defined as a system of ITMs that's parameterized by three ITMs. An ITM $\pi$ specifies the code of the *challenge protocol* for the experiment, an ITM $\mathcal{A}$ specifies the code of the *adversary*, and an ITM $\mathcal{Z}$ provides the code of the *environment*. The protocol execution experiment places precise conditions on the order in which ITIs are activated, and which ITIs are allowed to invoke or communicate with each other. The precise formal details of how these conditions are defined and imposed (*i.e.* the control function and related formalisms) can be found in [13], but we shall describe some of the relevant details informally. The experiment initially launches only an ITI running $\mathcal{Z}$. In turn, $\mathcal{Z}$ is permitted to invoke only a single ITI running $\mathcal{A}$, followed by (multiple) ITIs running the "challenge protocol" $\pi$ *provided that those ITIs running* $\pi$ *all share the same* $\mathsf{sid}$. This $\mathsf{sid}$, along with the $\mathsf{pid}$s of all the ITIs running $\pi$, may be chosen arbitrarily by $\mathcal{Z}$.

It is stressed that the environment *may not* invoke any additional ITIs, and it is only allowed to write to the input tapes of ITIs which it has directly invoked (or to receive outputs from those ITIs via its subroutine output tape). The environment may not interact with *any* of the communication tapes, nor the tapes of ITIs that it did not directly invoke. In summary, the environment can communicate only with the ITI running the code of the adversary $\mathcal{A}$, and ITIs participating in a <u>single session</u> of protocol $\pi$. We thus refer to the execution experiment as being a *constrained* one, and, in particular, the environment $\mathcal{Z}$ as being a *constrained environment*. The output of the environment $\mathcal{Z}$ in this basic UC protocol execution experiment is denoted by $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

**Ideal Functionalities.** We say an ITM $\mathcal{F}$ is an *ideal functionality* if its code represents a desired (interactive) function to be computed by parties or other protocols which may invoke it *as a subroutine* (and thus, in a perfectly secure way). The $\mathsf{pid}$ of any ITI running $\mathcal{F}$ is set to the special value $\perp$, indicating that the ITI is an ideal functionality. $\mathcal{F}$ accepts input from other ITIs that have the same $\mathsf{sid}$ as $\mathcal{F}$, and may write outputs to multiple ITIs as well.

Every ideal functionality $\mathcal{F}$ also induces an *ideal protocol* $\mathrm{IDEAL}_{\mathcal{F}}$. Parties running $\mathrm{IDEAL}_{\mathcal{F}}$ with the session ID $\mathsf{sid}$ act as *dummy parties*, simply forwarding their inputs to the input tape of an ITI running $\mathcal{F}$ with the same $\mathsf{sid}$, and copying any subroutine output received from $\mathcal{F}$ to the subroutine output tape of the ITI which invoked the party (typically, the environment $\mathcal{Z}$).

**UC Emulation and Realizations.** In the basic UC framework, a protocol $\pi$ is said to *UC-emulate* another protocol $\phi$ if, for any adversary $\mathcal{A}$, there exists a *simulator* $\mathcal{S}$ such that for all environments $\mathcal{Z}$ it holds that $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. That is, no environment behaves significantly differently in the protocol execution experiment when interacting with the challenge protocol $\pi$, under any given attack, than it does when interacting with the challenge protocol $\phi$, under a simulation of that same attack.Intuitively, this means that the protocol $\pi$ is "at least as secure as" the protocol $\phi$, since the effect of any attack on $\pi$ can also be emulated by attacking $\phi$. A protocol $\pi$ is said to *UC-realize* an ideal functionality $\mathcal{F}$ if $\pi$ UC-emulates $\mathrm{IDEAL}_{\mathcal{F}}$. Furthermore, if the protocol $\pi$ is a $\mathcal{G}$-hybrid protocol, then we say that $\pi$ is a UC-secure realization of $\mathcal{F}$ in the $\mathcal{G}$-hybrid model.

#### 2.2.2 Generalized UC Security

We present the generalized variant of UC security. Technically, the difference is very small; however, its effect is substantial. The essential difference here from the basic UC security notion is that here the environment $\mathcal{Z}$ is allowed to invoke ITIs with arbitrary code and arbitrary SIDs, including multiple concurrent instances of the challenge protocol $\pi$ and other protocols. We stress that these ITIs are even allowed to *share state* with each other across multiple sessions (which is a significant departure from prior models). To simplify the presentation and analysis, we will still assume that $\mathcal{Z}$ invokes only a single instance of the adversary $\mathcal{A}$.[1] We call such an environment *unconstrained*, since it need not obey any constraints in regards to which protocols it may invoke.[2] To distinguish from the basic UC experiment, we denote the output of an unconstrained environment $\mathcal{Z}$ attempting to distinguish a challenge protocol $\pi$ in the GUC protocol execution experiment, with an adversary $\mathcal{A}$, as $\text{GEXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. GUC emulation, is now defined as follows, analogously to the definition of basic UC emulation outlined above:

**Definition 1** (GUC-Emulation)**.** *Let $\pi$ and $\phi$ be PPT multi-party protocols. We say that $\pi$ GUC-emulates $\phi$ if, for any PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any (unconstrained) PPT environment $\mathcal{Z}$, we have:*

$$\text{GEXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{GEXEC}_{\pi,\mathcal{A},\mathcal{Z}}$$

As long as the protocol in question makes sure that no instance shares any subroutines with other protocol instances, GUC security is equivalent to basic UC security. This statement, which intuitively follows from the universal composition theorem, will be formalized later. However, we are primarily interested in protocols that *do* share some modules, or subroutines, with other protocol instances. For such protocols the generalized formulation differs radically from the basic one. Specifically, we are interested in modeling "shared trusted modules", which are captured via the shared functionality construct.

**Shared Functionalities.** In addition to the notion of ideal functionalities inherited from the basic UC security setting, we coin the notion of *shared functionalities*. A shared functionality $\bar{\mathcal{G}}$ is completely analogous to an ideal functionality, only it may additionally accept inputs from ITIs with *arbitrary* session IDs. Thus, a shared functionality is just an ideal functionality that may communicate with more than one protocol session. In order to distinguish shared functionalities from ideal functionalities, we require the sid of a shared functionality to begin with the special

[1] Although it is conceptually interesting to consider scenarios where the environment may invoke separate adversaries to attack separate instances of the challenge protocol, particularly when there is some shared state, it can be shown that this notion is equivalent to our simplified single adversary model.

[2] More formally, the control function for the GUC protocol execution experiment allows $\mathcal{Z}$ to invoke ITIs running arbitrary ITM code, and with arbitrary (but unique) identities. In particular, the environment may invoke many ITIs with a special code $\perp$ (even with different sids), which the control function will substitute for ITIs running $\pi$. Thus, the reason for our departure with the convention of [13] (which replaces the code of all ITIs invoked by $\mathcal{Z}$ with instances of $\pi$) is to provide $\mathcal{Z}$ with the ability to invoke ITIs running arbitrary code (other than $\pi$), yet still enable it to invoke instances of the challenge protocol without having access to its code.

symbol # (which is used exclusively in the sid of shared functionalities). As a shorthand notation, we denote the portion of the sid of a shared functionality which follows the # symbol by shsid (and thus, shared functionalities have sid = #‖shsid). Similarly to ideal functionalities, shared functionalities also have # as their fixed pid, and thus all shared functionalities have fixed identities (and can be invoked only by specifying the code of their corresponding ITM).

**Discussion.** Recall that in the basic UC definition, $\mathcal{Z}$ is constrained in its ability to invoke protocols: it may only invoke precisely those parties participating in the one single session of the challenge protocol it is attempting to distinguish, and aside from its ability to invoke and communicate with an adversary it may not invoke any other ITIs (*e.g.* representing other parties and protocol sessions concurrently running in the network) of any sort. Intuitively, it would seem that if this constraint were removed and $\mathcal{Z}$ were allowed to invoke arbitrary ITIs running arbitrary protocols (including multiple concurrent sessions of the challenge protocol itself), that $\mathcal{Z}$ would become a more powerful distinguisher (strengthening the security requirements for protocols to remain indistinguishable). As we will see, in reality, since basic UC security does not allow protocols to *share state*[3] with each other, any concurrent protocol executions that $\mathcal{Z}$ might wish to run can simply be simulated by $\mathcal{Z}$ internally with no need to actually invoke the ITIs. Thus, the constraint that $\mathcal{Z}$ may only invoke parties running a single instance of the protocol it is attempting to distinguish is not a true limitation, and indeed this is where the power of UC security comes from (*e.g.* the ability for $\mathcal{Z}$ to conduct this kind of internal simulation is why UC security holds even in the presence of concurrent protocol executions).

Unlike this basic UC security setting, we wish to consider definitions of security even for protocols that may share state information externally with other (concurrently executing) sessions of the same protocol, or even with other (independently designed) protocols. In such a setting, it is no longer possible for $\mathcal{Z}$ to simulate other protocol executions internally, since some of those protocols may share state with the protocol that $\mathcal{Z}$ is attempting to distinguish. Thus, the constraints placed on $\mathcal{Z}$ in the basic UC setting are of great concern for us, since they would prevent $\mathcal{Z}$ from ever seeing the effects of other protocol executions that share state externally with the protocol $\mathcal{Z}$ is attempting to distinguish, whereas protocol executions in the real world would certainly involve such effects. In order to properly capture interactions between protocols which share state information externally, we introduce the notion of Generalized UC (GUC) security, which builds off the basic UC security concepts outlined above.

Here we note that the differences between GUC-emulation and basic UC-emulation are in the use of an unconstrained environment, and the ability of $\pi$ and $\phi$ to invoke shared functionalities (which is not allowed in the basic UC setting). As an important intuition, we observe that since $\mathcal{Z}$ may invoke ITIs with arbitrary code, it may invoke ITIs which communicate with any shared functionalities invoked by $\pi$ (or $\phi$). Thus, $\mathcal{Z}$ may essentially access shared functionalities in an arbitrary manner (the primary restriction being the uniqueness of the identities of the ITIs which $\mathcal{Z}$ may invoke).

Intuitively, we call this security notion Generalized UC since the challenge protocol may interact with *external* protocols in an arbitrary manner. Whereas in the basic UC security setting, external

[3]The typical example of protocols sharing state occurs when using the CRS model. Multiple instances of a protocol may all share the same CRS, which certainly implies a relationship between those protocol executions which is not captured by standard UC security.

protocols were viewed as being independent of the challenge protocol itself, in the GUC setting shared functionalities may link the "state" of the challenge protocol to the state of other protocols running in the network which may seem completely external to the challenge protocol session under consideration.

### 2.2.3 Externalized UC Security

Since the unconstrained environment in GUC security setting we have just described is able to invoke arbitrary ITIs (and thus cause arbitrary interactions with shared functionalities, etc.), it becomes difficult to directly prove that a protocol GUC-emulates another protocol, *i.e.* to show that a simulated adversary $\mathcal{S}$ attacking a protocol $\phi$ behaves indistinguishably from an actual adversary $\mathcal{A}$ attacking protocol $\pi$. In particular, such analysis seems to directly involve arguing about systems where multiple instances of multiple protocols run concurrently. This stands in contrast to the situation with basic UC security, where it suffices to analyze a single instance of the protocol in isolation, and security in a multi-instance system follows from a general composition theorem.

We alleviate this situation in two steps: As a first step, we formulate another notion of protocol emulation, called externalized UC emulation, which is considerably simpler and in particular considers only a single instance of the protocol in question. We then show that this simplified notion is equivalent to the above general notion. In a second step, we re-assert the universal composition theorem with respect to GUC-emulation.

We remark that in the basic UC framework these two conceptual steps are demonstrated via the same technical theorem (namely, the UC theorem). We find that in the present framework it is clearer to separate the two issues.

**Subroutine respecting protocols.** Before proceeding to define externalized UC emulation, we coin the following terminology. We say that an ITI $M$ is a *subroutine* of another ITI $M'$ if $M$ either receives inputs on its input tape from $M'$ (and does not explicitly ignore them), or writes outputs to the subroutine output tape of $M'$. Recursively, we also say that if $M$ is a subroutine of a party (ITI) running protocol $\pi$ or a *sub-party* of protocol $\pi$, then $M$ is a *sub-party* of protocol $\pi$. By uniqueness of session identifiers, if there is an instance of protocol $\pi$ running with session ID $\mathsf{sid}$, all ITIs running with session ID $\mathsf{sid}$ are running $\pi$ or are sub-parties of $\pi$.

A protocol $\pi$ is said to be $\bar{\mathcal{G}}$*-subroutine respecting* if none of the sub-parties of an instance of $\pi$ provides output to or receives input from any ITI that is not also party/sub-party of that instance of $\pi$, *except* for communicating with a *single instance* of the shared ITI $\bar{\mathcal{G}}$. In other words, an instance of a $\bar{\mathcal{G}}$-subroutine respecting protocol $\pi$ has the property that all sub-parties of this instance of $\pi$ are only allowed to communicate with parties or sub-parties of this same instance of $\pi$ (they do not share themselves with other protocol instances in any way), with the sole exception that calls to a shared functionality $\bar{\mathcal{G}}$ are allowed. Using this terminology, we can now define externalized UC emulation.

**The Externalized UC Protocol Execution Experiment.** Rather than allowing the environment to operate completely unconstrained as in the GUC experiment, we constrain the environment so that it may only invoke particular types of ITIs. Specifically, the environment is only allowed to invoke a single instance of the challenge protocol (as in the constrained environment of basic UC), plus a *single* ITI running the code of a shared functionality (i.e., a shared subroutine) $\bar{\mathcal{G}}$. In other words, the EUC experiment is the same as the basic UC experiment, except the (otherwise constrained) environment is also allowed to provide input to and obtain output from a single instance of a shared functionality (which is specified by the challenge protocol under consideration). We say that such an environment is $\bar{\mathcal{G}}$-*externally constrained* if it is allowed such extra access to a shared functionality $\bar{\mathcal{G}}$. (Note that although we consider only one shared functionality at a time for the sake of simplicity, it is also reasonable to define the notions of "subroutine respecting" and "EUC security" with respect to multiple shared functionalities.) Given a $\bar{\mathcal{G}}$-subroutine respecting protocol $\pi$, we denote the output of the environment in the $\bar{\mathcal{G}}$-EUC protocol experiment by $\mathrm{EXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A},\mathcal{Z}}$. EUC-emulation is defined analogously to the notion of GUC-emulation:

**Definition 2** (EUC-Emulation)**.** *Let $\pi$ and $\phi$ be PPT multi-party protocols, where $\pi$ is $\bar{\mathcal{G}}$-subroutine respecting. We say that $\pi$ EUC-emulates $\phi$ with respect to shared functionality $\bar{\mathcal{G}}$ (or, in shorthand, that $\pi$ $\bar{\mathcal{G}}$-EUC-emulates $\phi$) if for any PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any $\bar{\mathcal{G}}$-externally constrained environment $\mathcal{Z}$, we have:*

$$\mathrm{EXEC}^{\bar{\mathcal{G}}}_{\phi,\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A},\mathcal{Z}}$$

**$\bar{\mathcal{G}}$-EUC Secure Realization.** We say that a protocol $\pi$ *realizes* an ideal functionality $\mathcal{F}$ if $\pi$ $\bar{\mathcal{G}}$-EUC-emulates $\mathrm{IDEAL}_F$. Notice that the formalism implies that the shared functionality $\bar{\mathcal{G}}$ exists both in the model for executing $\pi$ *and also in the model for executing the ideal protocol for $\mathcal{F}$, $\mathrm{IDEAL}_F$.*

We remark that the notion of $\bar{\mathcal{G}}$-EUC-emulation can be naturally extended to protocols that use several different shared functionalities (instead of only one).

#### 2.2.4 Equivalence of GUC to EUC and a generalized UC theorem

We show that $\bar{\mathcal{G}}$-EUC-emulation is, surprisingly, equivalent to full GUC-emulation for any $\bar{\mathcal{G}}$-subroutine respecting protocol. Perhaps unsurprisingly, the proof of the equivalence theorem incorporates most of the arguments of the universal composition theorem. In particular, the "quality" of security degrades linearly with the number of instances of $\pi$ invoked by $\mathcal{Z}$ in the GUC experiment.

The formal statement of this equivalence is given in Theorem 2.1 above. The proof of the theorem, which we now give, makes use of a hybrid argument (akin to that in the universal composition theorem of [13]) to show that security for the single-instance setting of EUC is sufficient to ensure security under the more strenuous multi-instance setting of GUC.

*Proof of Theorem 2.1.* It is trivial to observe that protocols which GUC-emulate each other also $\bar{\mathcal{G}}$-EUC-emulate each other (since any simulation that is indistinguishable to unconstrained environments is certainly indistinguishable to the special case of $\bar{\mathcal{G}}$-externally constrained environments as well), but the other direction is non-obvious. The basic idea of the proof is that an $\bar{\mathcal{G}}$-externally constrained environment can simulate the same information available to an unconstrained environment, even while operating within its constraints. The proof technique is essentially the same as the proof of composition theorem, only in this case multiple sessions must be handled directly without going through an intermediate protocol. (That is, the proof of composition theorem considers a protocol $\pi$ which emulates a protocol a $\phi$, and shows that a protocol $\rho$ which may invoke multiple copies of $\pi$ emulates a protocol $\rho$ which invokes $\phi$ instead. Here, we essentially need to demonstrate that if a single copy of $\pi$ emulates $\phi$ then multiple copies of $\pi$ emulate multiple copies of $\phi$.)

Applying the technique of [13], we observe that there are equivalent formulations of protocol emulation with respect to dummy adversaries (this needs to be proven separately, but the proofs are identical to those for the original notion of UC emulation), and we will use those formulations here to simplify the proof. Let $\mathcal{D}$ denote the fixed "dummy adversary" (which simply forwards messages to and from the environment). For the remainder of the proof, we shall refer to the "simulator $\mathcal{S}$" as the "adversary", in order to avoid confusion (roughly speaking, $\mathcal{S}$ attempts to simulate the attack of an adversary, so this terminology is appropriate).

Suppose that $\pi$ $\bar{\mathcal{G}}$-EUC-emulates $\phi$. This means there exists some adversary $\mathcal{S}$ that will satisfy $\mathrm{EXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{D},\mathcal{Z}} \approx \mathrm{EXEC}^{\bar{\mathcal{G}}}_{\phi,\mathcal{S},\mathcal{Z}}$ for any $\bar{\mathcal{G}}$-externally constrained environment $\mathcal{Z}$. To prove our claim, we will need to show that the existence of such $\mathcal{S}$ is sufficient to construct a new adversary $\tilde{\mathcal{S}}$ such that $\mathrm{GEXEC}_{\pi,\mathcal{D},\tilde{\mathcal{Z}}} \approx \mathrm{GEXEC}_{\phi,\tilde{\mathcal{S}},\tilde{\mathcal{Z}}}$ holds for any unconstrained environment $\tilde{\mathcal{Z}}$.

We construct an adversary $\tilde{\mathcal{S}}$ in a similar fashion to the construction of $\mathcal{A}_\pi$ in the proof of composition theorem in [13], using multiple instances of $\mathcal{S}$ that are simulated by $\mathcal{A}_\pi$ internally. That is, to ensure that each instance of $\phi$ mimics the corresponding instance of $\pi$, $\tilde{\mathcal{S}}$ will run separate copies of $\mathcal{S}$ for each instance of $\pi$ (and $\tilde{\mathcal{S}}$ then simply forwards messages between $\tilde{\mathcal{Z}}$ and the corresponding copy of $\mathcal{S}$ for that instance when necessary). We now prove that $\tilde{\mathcal{S}}$ satisfies the requirement for GUC-emulation via a hybrid argument (again, as is done in the proof of composition theorem).

Assume for the purpose of contradiction that $\mathrm{EXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{D},\tilde{\mathcal{Z}}} \not\approx \mathrm{EXEC}^{\bar{\mathcal{G}}}_{\phi,\tilde{\mathcal{S}},\tilde{\mathcal{Z}}}$ (in particular, assume the distinguishing advantage of $\tilde{\mathcal{Z}}$ is $\epsilon$). Let $m$ be an upper bound on the number of instances of $\pi$ which are invoked by $\tilde{\mathcal{Z}}$. For $l \leq m$, let $\tilde{\mathcal{S}}_l$ denote the adversary for a modified execution $\mathrm{EX}_l = \mathrm{EXEC}(l,\phi)^{\bar{\mathcal{G}}}_{\pi,\tilde{\mathcal{S}}_l,\tilde{\mathcal{Z}}}$ in which the first $l$ instances of $\pi$ are simulated[4] using instances of $\mathcal{S}$ and $\phi$ (as would be done by $\tilde{\mathcal{S}}$), but the remaining invocations of $\pi$ by $\tilde{\mathcal{Z}}$ are in fact handled by genuine instances of $\pi$ (with $\tilde{\mathcal{S}}_l$ simply forwarding messages directly to and from those instances, as $\mathcal{D}$ would). In particular, we observe that the modified interaction $\mathrm{EX}_0$ is just the interaction with $\pi$, and $\mathrm{EX}_l$ is the unmodified interaction with $\tilde{\mathcal{S}}$ and $\phi$ replacing $\mathcal{D}$ and $\pi$. Then, by our assumption that the interactions with $\mathrm{EX}_0$ and $\mathrm{EX}_m$ are distinguishable, there must be an $0 < l \leq m$ such that $\tilde{\mathcal{Z}}$ distinguishes between the modified interactions with $\mathrm{EX}_l$ and $\mathrm{EX}_{l-1}$ with advantage at least $\epsilon/m$. We can construct an $\bar{\mathcal{G}}$-externally constrained environment $\mathcal{Z}^*$ from such a $\tilde{\mathcal{Z}}$ which succeeds in distinguishing the ensembles $\mathrm{EXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{D},\mathcal{Z}^*}$ and $\mathrm{EXEC}^{\bar{\mathcal{G}}}_{\phi,\mathcal{S},\mathcal{Z}^*}$ with probability at least $\epsilon/m$, contradicting the fact that $\pi$ $\bar{\mathcal{G}}$-EUC-emulates $\phi$.

The construction of $\mathcal{Z}^*$ is slightly involved, but on a high level, $\mathcal{Z}^*$ internally simulates the actions of $\tilde{\mathcal{Z}}$ including all ITIs activated by $\tilde{\mathcal{Z}}$ (other than those for $\bar{\mathcal{G}}$ and the $l$-th instance of $\pi$), but forwards all communications sent to the $l$-th instance of $\pi$ to its own external interaction instead (which is either with a single instance of $\pi$ and $\mathcal{D}$, or $\phi$ and $\mathcal{S}$). We observe that since $\pi$ is subroutine respecting, the only way ITIs activated by the simulated $\tilde{\mathcal{Z}}$ may somehow share state information with the challenge instance of $\pi$ is via access to the shared functionality $\bar{\mathcal{G}}$. Whenever

[4]Technically, we *must* modify the execution experiment here, since it is the environment which attempts to invoke the challenge protocol $\pi$, which is beyond the control of the adversary $\tilde{\mathcal{S}}$. Thus $l$ and $\phi$ need to be specified as part of the execution experiment itself.

an ITI invoked by the internal simulation of $\tilde{\mathcal{Z}}$ wishes to communicate with $\bar{\mathcal{G}}$, $\mathcal{Z}^*$ invokes a corresponding dummy party with the same pid and sid, and then forwards communications between the internally simulated ITI and the actual shared functionality $\bar{\mathcal{G}}$ in $\mathcal{Z}^*$'s external interaction via the dummy party. $\mathcal{Z}^*$ then outputs whatever the internally simulated copy of $\tilde{\mathcal{Z}}$ outputs. This interaction results in the simulated $\tilde{\mathcal{Z}}$ operating with a view that corresponds to either $\mathrm{EX}_{l-1}$ or $\mathrm{EX}_l$ (which $\tilde{\mathcal{Z}}$ can distinguish with probability at least $\epsilon/m$), and thus $\mathcal{Z}^*$ successfully distinguishes with probability at least $\epsilon/m$, as claimed, completing the contradiction. $\square$

We observe that if a protocol $\pi$ does not use any shared functionalities (i.e., $\pi$ is $\bar{\mathcal{G}}$-subroutine respecting for a null functionality that generates no output) then a corollary of the above claim states that $\pi$ UC-emulates $\phi$ if and only if $\pi$ GUC-emulates $\phi$. This equivalence shows the power of the basic UC emulation security guarantee, since it is indeed equivalent to the seemingly stronger notion of GUC emulation (for any protocols which exist in the more limited basic UC setting).

**Universal Composition.** Finally, we generalize the universal composition theorem to hold also with respect to GUC-emulation. That is, consider a $\bar{\mathcal{G}}$-subroutine respecting protocol $\phi$ that is being used as a subroutine in some (arbitrary) larger protocol $\rho$. The new composition theorem guarantees that it is safe to replace a protocol $\phi$ with a different protocol $\pi$ that merely $\bar{\mathcal{G}}$-EUC-emulates $\phi$, and yet the resulting implementation of $\rho$ (which now invokes $\pi$ instead of $\phi$) will fully GUC-emulate the original version (which had invoked $\phi$).

The formal composition theorem is stated in Theorem 2.3 above, which we now prove. The proof is similar in spirit to the proof of universal composition theorem (but here we no longer require the hybrid argument, since multiple protocol instances are already taken care of by the GUC setting).

*Proof of Theorem 2.3.* Since the notions of $\bar{\mathcal{G}}$-EUC-emulation and GUC-emulation are equivalent for subroutine respecting protocols which do not use shared functionalities other than $\bar{\mathcal{G}}$, it suffices to prove that if $\pi$ GUC-emulates $\phi$ then $\rho^{\pi/\phi}$ GUC-emulates $\rho$ (of course, there is a corresponding loss of exact security as per Theorem 2.1). Thus, it suffices to prove that the composition theorem holds for subroutine respecting protocols that GUC-emulate each other. For the remainder of the proof, we shall refer to the "simulator $\mathcal{S}$" as the "adversary", in order to avoid confusion (roughly speaking, $\mathcal{S}$ attempts to simulate the attack of an adversary, so this terminology is appropriate).

The proof that GUC-emulation is composable follows the same general approach as the composition theorem for basic UC in [13], with some simplifications resulting from the use of unconstrained environments. We begin by noting that there is an equivalent formulation of GUC-emulation with respect to dummy adversaries (the proof of this claim is entirely analogous to the proof of the same statement for basic UC security). Thus, denoting the dummy adversary by $\mathcal{D}$, we wish to construct an adversary $\mathcal{A}_\rho$ such that

$$\mathrm{GEXEC}_{\rho^{\pi/\phi},\mathcal{D},\mathcal{Z}} \approx \mathrm{GEXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}} \tag{1}$$

for any unconstrained environment $\mathcal{Z}$.

Since $\pi$ GUC-emulates $\phi$ there is an adversary $\mathcal{S}$ such that $\mathrm{GEXEC}_{\pi,\mathcal{D},\mathcal{Z}_\pi} \approx \mathrm{GEXEC}_{\phi,\mathcal{S},\mathcal{Z}_\pi}$ for any unconstrained environment $\mathcal{Z}_\pi$. That is, $\mathcal{S}$ expects to interact with with many instances of $\phi$, with the goal of translating them to mimic the action of corresponding instances of $\pi$ from the viewpoint of any environment $\mathcal{Z}_\pi$. We will use $\mathcal{S}$ to construct $\mathcal{A}_\rho$ satisfying (1) above. Unlike the construction in the basic UC composition theorem, it is not necessary for $\mathcal{A}_\rho$ to run multiple copies of $\mathcal{S}$ (one for each session of $\pi$), since the GUC adversary $\mathcal{S}$ already deals with the scenario where multiple sessions of $\pi$ are executing (as unconstrained environments may invoke multiple instances of the challenge protocol). Thus the construction of $\mathcal{A}_\rho$ here is simpler.

$\mathcal{A}_\rho$ will simply run a single copy of $\mathcal{S}$ internally, forwarding all messages intended for instances of $\pi$ (which are sub-parties to instances of the challenge protocol $\rho$) sent by $\mathcal{A}_\rho$'s environment $\mathcal{Z}$ to its internal simulation of $\mathcal{S}$, as well as forwarding any messages from $\mathcal{S}$ back to $\mathcal{Z}$ as is appropriate. (Note that $\mathcal{A}_\rho$ need not simulate any interactions with instances of $\pi$ that are invoked directly by $\mathcal{Z}$ rather than an instance of $\rho$, since those are not associated with the challenge protocol.) Additionally, $\mathcal{A}_\rho$ forwards $\mathcal{S}$'s interactions with instances of $\phi$ between the external instances of $\phi$ (again, only those which are sub-parties to instances of the challenge protocol $\rho$) and its internal copy of $\mathcal{S}$ as well. (Intuitively, $\mathcal{A}_\rho$ acts as the environment for $\mathcal{S}$ by forwarding some of its own interactions with $\mathcal{Z}$ concerning instances of $\pi$, and also copying its own interactions with external instances of $\phi$. The reason $\mathcal{S}$ is not employed directly in place of $\mathcal{A}_\rho$ is that $\mathcal{A}_\rho$ must translate the "challenge protocol" sessions $\rho$ to isolate their sub-protocol invocations of $\phi$, which is the challenge protocol $\mathcal{S}$ expects to interact with. Thus, effectively, the instances of $\rho$ itself simply become part of the environment for $\mathcal{S}$. Note that there may be many instances of $\rho$ which are being translated, and each of those instances may invoke many instances of $\phi$.)

In order to prove that $\mathcal{A}_\rho$ satisfies (1) we perform a standard proof by contradiction. Assume there exists an environment $\mathcal{Z}$ capable of distinguishing the interaction with $\mathcal{A}_\rho$ and $\rho$ from the interaction with $\mathcal{D}$ and $\rho^{\pi/\phi}$. We show how to construct an environment $\mathcal{Z}_\pi$ capable of distinguishing an interaction between $\mathcal{S}$ and $\phi$ from an interaction with $\mathcal{D}$ and $\pi$, contradicting the fact that $\pi$ GUC-emulates $\phi$.

The construction of $\mathcal{Z}_\pi$ is again analogous to the technique applied in [13], with some additional simplification (since there is no hybrid argument here, we may simply treat all instances of $\phi$ the same way). As a useful tool in describing the construction of $\mathcal{Z}_\pi$, we briefly define an "internal simulation adversary" $\hat{\mathcal{A}}_\rho$, which will be run internally by $\mathcal{Z}_\pi$ alongside an internally simulated copy of $\mathcal{Z}$. Whenever $\hat{\mathcal{A}}_\rho$ receives a message, it performs the same function as $\mathcal{A}_\rho$, only replacing $\mathcal{A}_\rho$'s communications with its internal simulation of the adversary $\mathcal{S}$ (along with its corresponding challenge protocol $\phi$) by communications with the external adversary for $\mathcal{Z}_\pi$ (and its corresponding challenge protocol, which will either be $\pi$ if the adversary is $\mathcal{D}$ or $\phi$ if the adversary is $\mathcal{S}$). We observe that if $\mathcal{Z}_\pi$'s adversary is $\mathcal{D}$, then $\hat{\mathcal{A}}_\rho$ also acts like $\mathcal{D}$, since it will merely forward all messages. Similarly, if $\mathcal{Z}_\pi$'s external adversary is $\mathcal{S}$, then $\hat{\mathcal{A}}_\rho$ will function identically to $\mathcal{A}_\rho$.

On a high level, the environment $\mathcal{Z}_\pi$ will internally simulate the environment $\mathcal{Z}$ and adversary $\hat{\mathcal{A}}_\rho$, externally invoking copies of any ITIs that are invoked by the simulations (with the exception of instances of $\mathcal{Z}$'s challenge protocol $\pi$, and any invocations of $\mathcal{Z}_\pi$'s challenge protocol made by $\hat{\mathcal{A}}_\rho$), appropriately forwarding any messages between those ITIs and its internal copies of $\mathcal{Z}$ and $\mathcal{A}_\rho$. Whenever the internal copy of $\mathcal{Z}$ wishes to invoke an instance of the challenge protocol $\rho$, the environment $\mathcal{Z}_\pi$ internally simulates the instance (by modeling an ITI running $\rho$ with the specified identity), forwarding any communications between $\rho$ and shared functionalities to external instances of those shared functionalities (such forwarding may be accomplished by $\mathcal{Z}_\pi$ externally invoking dummy parties the with same identities as the sub-parties of $\rho$ that wish to communicate with the shared functionalities, and then forwarding communications through the dummy parties). Because $\rho$ is subroutine respecting, it is safe to conduct such an internal simulation, as instance of $\rho$ do not share state with any ITIs external to $\mathcal{Z}_\pi$ except via the shared functionalities (which are handled appropriately). Of course, whenever the internal simulation of $\hat{\mathcal{A}}_\rho$ wish to communicate with an instance of its challenge protocol, $\mathcal{Z}_\pi$ will forward the communications to the correct instance of its own challenge protocol, as described above. When the internally simulated $\mathcal{Z}$ halts and provides output, $\mathcal{Z}_\pi$ similarly halts, copying the same output.

Now, we can observe that $\mathrm{GEXEC}_{\pi,\mathcal{D},\mathcal{Z}_\pi} = \mathrm{GEXEC}_{\rho^{\pi/\phi},\mathcal{D},\mathcal{Z}}$ by considering that the internal simulation conducted by $\mathcal{Z}_\pi$ will be a faithful recreation of the latter experiment. In particular, by its construction, the simulation of $\hat{\mathcal{A}}_\rho$ will simply act as the dummy adversary $\mathcal{D}$. Furthermore, the internal simulation of $\rho$ is correctly replacing all invocations of $\phi$ by invocations of $\pi$ (and thus is a perfect simulation of $\rho^{\pi/\phi}$), while the rest of the experiment proceeds identically. A similar argument yields that $\mathrm{GEXEC}_{\phi,\mathcal{S},\mathcal{Z}_\pi} = \mathrm{GEXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}}$. Previously, we assumed for the sake of contradiction that there exists an environment $\mathcal{Z}$ such that $\mathrm{GEXEC}_{\rho^{\pi/\phi},\mathcal{D},\mathcal{Z}} \not\approx \mathrm{GEXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}}$. Combining this statement with the previous two equations yields the result that there exists an environment $\mathcal{Z}_\pi$ such that $\mathrm{GEXEC}_{\pi,\mathcal{D},\mathcal{Z}_\pi} \not\approx \mathrm{GEXEC}_{\phi,\mathcal{S},\mathcal{Z}_\pi}$, contradicting the fact that $\pi$ GUC-emulates $\phi$, completing our proof. □

# 3 Insufficiency of the Global CRS Model

In this section we demonstrate that a global CRS setup is *not* sufficient to GUC-realize even the basic two-party commitment functionality. We then further elaborate the nature of this insufficiency by considering some weaknesses in the security of previously proposed constructions in the CRS model. Finally, we suggest a new "intuitive" security goal, dubbed *full simulatability*, which we would like to achieve by utilizing the GUC-security model (and which was not previously achieved by any protocols in the CRS model).

## 3.1 Impossibility of GUC-realizing $\mathcal{F}_{com}$ in the $\bar{\mathcal{G}}_{crs}$ model

This section shows that the simple CRS model is insufficient for GUC-realizing $\mathcal{F}_{com}$. Let us elaborate.

Recall that many interesting functionalities are unrealizable in the UC framework without any setup assumption. For instance, it is easy to see that the ideal authentication functionality, $\mathcal{F}_{auth}$, is unrealizable in the plain model. Furthermore, many two party tasks, such as Commitment, Zero-Knowledge, Coin-Tossing, Oblivious Transfer and others cannot be realized in the UC framework by two-party protocols, even if authenticated communication is provided [17, 18, 12].

As a recourse, the common reference string (CRS) model was used to re-assert the general feasibility results of [28] in the UC framework. That is, it was shown that any "well-formed" ideal functionality can be realized in the CRS model [17, 19]. However, the formulation of the CRS model in these works postulates a setting where the reference string is given *only to the participants in the actual protocol execution.* That is, the reference string is chosen by an ideal functionality, $\mathcal{F}_{crs}$, that is dedicated to a given protocol execution. $\mathcal{F}_{crs}$ gives the reference string only to the adversary and the participants in that execution. Intuitively, this formulation means that, while the reference string need not be kept secret to guarantee security, it cannot be safely used by other protocol executions. In other words, no security guarantees are given with respect to executions that use a reference string that was obtained from another execution rather than from a dedicated instance of $\mathcal{F}_{crs}$. (The UC with joint state theorem of [20] allows multiple executions of certain protocols to use the same instance of the CRS, but it requires all instances that use the CRS to be carefully designed to satisfy some special properties.)

In contrast, we are interested in modeling a setting where the same CRS is globally available to all parties and all protocol executions. This means that a protocol $\pi$ that uses the CRS must take into account the fact that the same CRS may be used by *arbitrary* other protocols, even protocols that were specifically designed to interact badly with $\pi$. Using the GUC security model defined in Section 2, we define this weaker setup assumption as a *shared* ideal functionality that provides the value of the CRS not only to the parties of a given protocol execution, but rather to all parties, and even directly to the environment machine. In particular, this global CRS functionality, $\bar{\mathcal{G}}_{crs}$, exists in the system both as part of the protocol execution and as part of the ideal process. Functionality $\bar{\mathcal{G}}_{crs}$ is presented in Figure 3.

**Functionality $\bar{\mathcal{G}}_{crs}$**

Parameterized by a distribution $\mathcal{PK}$, $\bar{\mathcal{G}}_{crs}$ proceeds as follows, when activated by any party:

1. If no value has been previously recorded, choose a value $MPK \xleftarrow{\$} \mathcal{PK}$, and record the value $MPK$.
2. Return the value $MPK$ to the activating party.

Figure 3: The Global Common Reference String functionality. The difference from the Common Reference String functionality $\mathcal{F}_{crs}$ of [12, 17] is that $\mathcal{F}_{crs}$ provides the reference string only to the parties that take part in the actual protocol execution. In particular, the environment does not have direct access to the reference string.

We demonstrate that $\bar{\mathcal{G}}_{crs}$ is insufficient for reproducing the general feasibility results that are known to hold in the $\mathcal{F}_{crs}$ model. To exemplify this fact, we show that no two-party protocol that uses $\bar{\mathcal{G}}_{crs}$ as its only setup assumption GUC-realizes the ideal commitment functionality, $\mathcal{F}_{com}$ (presented in Figure 4). The proof follows essentially the same steps as the [17] proof of impossibility of realizing $\mathcal{F}_{com}$ in the plain model. The reason that these steps can be carried out even in the presence of $\bar{\mathcal{G}}_{crs}$ is, essentially, that the simulator obtains the reference string from an external entity ($\bar{\mathcal{G}}_{crs}$), rather than generating the reference string by itself. We conjecture that most other impossibility results for UC security in the plain model can be extended in the same way to hold for GUC security in the presence of $\bar{\mathcal{G}}_{crs}$.

**Functionality $\mathcal{F}_{com}$**

**Commit Phase:** Upon receiving a message $(\texttt{commit}, sid, P_c, P_r, b)$ from party $P_c$, where $b \in \{0, 1\}$, record the value $b$ and send the message $(\texttt{receipt}, sid, P_c, P_r)$ to $P_r$ and the adversary. Ignore any future `commit` messages.

**Reveal Phase:** Upon receiving a message $(\texttt{reveal}, sid)$ from $P_c$: If a value $b$ was previously recorded, then send the message $(\texttt{reveal}, sid, b)$ to $P_r$ and the adversary and halt. Otherwise, ignore.

Figure 4: The Commitment Functionality $\mathcal{F}_{com}$ (see [17])

**Theorem 3.1.** *There exists no bilateral, terminating protocol $\pi$ that GUC-realizes $\mathcal{F}_{com}$ and uses only the shared functionality $\bar{\mathcal{G}}_{crs}$. This holds even if the communication is ideally authentic.*

*Proof.* Intuitively, the proof of the impossibility of UC commitments (for the plain model) described in [17] holds here as well, since an $\bar{\mathcal{G}}_{crs}$-externally constrained environment $\mathcal{Z}$ is able to obtain a copy of the global CRS directly from $\bar{\mathcal{G}}_{crs}$ by invoking a separate dummy party specifically to obtain the reference string, preventing the simulator $\mathcal{S}$ from choosing the CRS on its own (in order to arrange knowledge of a trapdoor).

More formally, suppose that there exists a commitment protocol $\pi$ (for a party $P_c$ committing a bit $b$ to a party $P_r$) and a simulator $\mathcal{S}$ such that $\mathrm{EXEC}_{\mathcal{F}_{com},\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ for any adversary $\mathcal{A}$ and any $\bar{\mathcal{G}}_{crs}$-externally constrained environment $\mathcal{Z}$ (here we may even allow $\mathcal{S}$ to depend on the choice of $\mathcal{A}$ and $\mathcal{Z}$). We will arrive at a contradiction.

We accomplish this by constructing a new $\bar{\mathcal{G}}_{crs}$-externally constrained environment $\mathcal{Z}'$ and a new adversary $\mathcal{A}'$ such that there is *no* simulator $\mathcal{S}'$ which can satisfy $\mathrm{EXEC}_{\mathcal{F}_{com},\mathcal{S}',\mathcal{Z}'} \approx \mathrm{EXEC}_{\pi,\mathcal{A}',\mathcal{Z}'}$. Recall that an $\bar{\mathcal{G}}_{crs}$-externally constrained environment may invoke dummy parties running $\mathrm{IDEAL}_{\bar{\mathcal{G}}_{crs}}$ using any unique $(\mathsf{pid}, \mathsf{sid})$, and thus may obtain a copy of the global CRS.

Our $\mathcal{A}'$ is constructed so as to corrupt the recipient $P_r$ at the beginning of the protocol. During the protocol, $\mathcal{A}'$ will run the algorithm for $\mathcal{S}$ using the same CRS as obtained from $\bar{\mathcal{G}}_{crs}$ (via $\mathcal{Z}'$) to respond to all of $\mathcal{S}$'s $\bar{\mathcal{G}}_{crs}$ queries, and using the same party and session identities for $P_c$ and $P_r$ in this "virtual" run of $\mathcal{S}$. Furthermore, while acting as the environment for this copy of $\mathcal{S}$, $\mathcal{A}$ will "corrupt" the party "$P_c$" in the virtual view of $\mathcal{S}$. Whenever $\mathcal{A}'$ receives protocol messages from the honest party $P_c$ in the real protocol execution, it sends the same messages on behalf of the "corrupt party $P_c$" in the virtual view of $\mathcal{S}$. Whatever messages $\mathcal{S}$ would send on behalf of the "honest" virtual recipient "$P_r$", $\mathcal{A}'$ will send on behalf of the real party $P_r$ (which it has previously corrupted). At some point, $\mathcal{S}$ must send the message $(\texttt{commit}, sid, P_c, P_r, b')$ to the commitment functionality. At this point, the adversary $\mathcal{A}'$ will output the bit $b'$, and halt.

We define the environment $\mathcal{Z}'$ to choose a random bit $b$, and provide it as the input for the honest committer $P_c$. If the adversary outputs $b'$ such that $b' = b$, then $\mathcal{Z}'$ outputs 1 (and 0 otherwise). (Additionally, we implement any trivial interface for $\mathcal{Z}'$ to pass a copy of the CRS to $\mathcal{A}'$.) Observe that no decommitment ever occurs, and thus the view of $\mathcal{S}'$ must be independent of the choice of $b$ (meaning that $\mathcal{S}'$ must be correct with probability 1/2). However, as $\mathcal{S}$ must produce a bit $b'$ that matches $b$ with all but negligible probability (since we assumed it simulates the protocol $\pi$ correctly), $\mathcal{A}'$s guess $b'$ must match $b$ with high probability, and thus $\mathcal{Z}'$ will clearly distinguish between the guesses of $\mathcal{A}'$ and those of $\mathcal{S}'$ (which are correct with probability exactly 1/2). $\square$

In fact, it is easy to see that the above impossibility result extends beyond the mere availability of $\bar{\mathcal{G}}_{crs}$ to any circumstance where the shared functionality will only provide information globally (or, yet more generally, the impossibility holds whenever all the shared information available to protocol participants can also be obtained by the environment). For instance, this impossibility will hold even in the (public) random oracle model, which is already so strong that it cannot truly be realized without the use of a fully interactive trusted party. Another interpretation of this result is that no completely *non-interactive* global setup can suffice for realizing $\mathcal{F}_{com}$. The next section studies the problem of realizing $\mathcal{F}_{com}$ using setup assumptions with minimal interaction requirements.

## 3.2 Deniability and Full Simulatability

To demonstrate that the problems with using a global CRS to realize $\mathcal{F}_{com}$, in the fashion of [19], are more than skin deep technicalities that arise only in the GUC framework we now consider the issue of deniability. Intuitively, a protocol is said to be "deniable" if it is possible for protocol participants to deny their participation in a protocol session by arguing that any "evidence" of their participation (as obtained by other, potentially corrupt protocol participants) could have been fabricated.

Recalling the intuition outlined in the introduction, we would like realized protocols to guarantee the same security as the ideal functionalities they realize, meaning that the adversary will learn nothing more from attacking the protocol than could be learned from attacking its corresponding ideal functionality. Protocols realized with such a guarantee are inherently *deniable*, since a protocol participant can accurately argue that any information sent during the protocol session could have been obtained by an adversary using only the output from the ideal functionality[5] in an attack simulation conducted entirely without his or her actual participation.

For instance, if we consider the ideal functionality for Zero Knowledge (ZK), we expect that any secure realization of that functionality should reveal no information to the adversary beyond the output of the ideal functionality (which contains only a single bit). In particular, the ideal functionality output can easily be generated entirely without the help of the prover, and thus the prover should be able to deny his participation in any proof protocols, since they reveal no information that could not have been obtained independently. However, we already know from the result of [35] that it is impossible to achieve such deniability for ZK in the CRS model. Indeed, we may see that the UC simulator for ZK functionality in [19] chooses a fresh CRS, and generates the simulated protocol transcripts with respect to that, instead of the published real-world CRS. Thus, if a protocol transcript makes use of the real-world CRS, it could not have been obtained via simulation (so a successful prover is indeed incriminated by the transcript).

When there is no deniability, the adversary is truly able to obtain valuable information by observing protocol interactions that would not be revealed by the ideal functionality. Thus we have found a practical example of an actual security loss that direct results from the relaxations of UC security inherent in the CRS technique of [19]. We can now clearly see that the impossibility of realizing $\mathcal{F}_{com}$ via the CRS model in the GUC setting is due to a meaningful strengthening of security guarantees (since deniability is guaranteed in the GUC setting, and that guarantee is not achieved by protocols realized in the CRS model).

On an intuitive level, it might be helpful to consider the issue of deniability in light of the "real world" resources required in order to run the GUC simulator to simulate a given protocol session. If the resources required to simulate a protocol session are readily available, then we say the protocol session is *plausibly deniable* (since it is plausible that information obtained from the protocol was the result of a simulation). If the resources required to simulate are difficult or impossible to obtain, then there is no guarantee of plausible deniability (since it will be difficult to convince others that

[5]Of course, if the output of the ideal functionality "incriminates" a user by revealing some of his secrets, the resulting protocol does not meet our intuitive understanding of the word "deniable". Still, the protocol itself may be said to be "as deniable" as the functionality it realizes.

an incriminating protocol transcript was the result of a simulation). We wish to employ simulation techniques that require only minimal resources to conduct a simulation, increasing the plausibility of denials (as well as decreasing the value of any information that an adversary might obtain by attacking a secure protocol). Thus, we use the term *fully simulatable* to refer to any plausibly deniable protocol realized in the GUC framework.

From this vantage point, we observe that the resource required to conduct the protocol simulations in [19] is a "trapdoor" for the CRS. In particular, the CRS must be "rigged" with such a trapdoor *a priori*. Such rigging is certainly not plausible when there is a trusted party choosing the CRS, and this is in fact the root of the deniability problem for the CRS model. Furthermore, knowledge of this trapdoor implies the ability to completely violate security of any protocol constructed using the techniques of [19], and thus there would be no security against any entity capable of simulating protocols. Similarly, in the "imaginary angel" model of [37], the simulator requires access to super-polynomial time functionalities that are certainly not plausibly available in the real world (and thus, the deniability problem arises there as well). Indeed, if the "imaginary angels" of [37] were to somehow be made practical in the real world, all security would be lost.

We comment that, although we do not make any attempt to formalize a "general" notion of deniability here, the guarantee we seek to provide is that protocols are "as deniable" in the real world as they would have been in the ideal world (past works did not satisfy even this basic requirement). In fact, as we will see, our particular realization of fully simulatable security will guarantee that even "on line" (interactive) deniability is preserved, since the simulator can very practically be run in real time. Indeed, as long as an honest party $P$ never deviates from the protocol, it is not possible for other (even corrupt) protocol participants to conclusively demonstrate $P$'s participation in the fully simulatable protocol session to a third party, *even while the protocol is ongoing*!

# 4 Fully Simulatable General Computation

We now turn our attention to the problem of *constructing* fully simulatable GUC-secure protocols. That is, we would like it to be possible for a real-world adversary to simulate the effects of any attack on a protocol (in a computationally indistinguishable manner), without actually conducting the attack on the protocol (instead utilizing only the information that would be revealed by an ideally secure realization). The impossibility result of Section 3 implies that we cannot do this in the standard CRS model (if we correctly model the global availability of the CRS). Thus, we must consider alternatives to the CRS model if we hope to achieve our goal.

Since we must somehow avoid the impossibility result of Section 3 for the CRS model, we would like to find reasonable alternative global setup assumptions that allow for realizing interesting tasks. That is, we are looking for shared functionalities $\bar{\mathcal{G}}$ (as defined in Section 2), so that on the one hand $\bar{\mathcal{G}}$ will be implementable in reality with reasonable trust assumptions, and on the other hand we will have protocols that GUC-realize interesting functionalities and still use no setup (i.e., no ideal functionalities) other than $\bar{\mathcal{G}}$. We say that such GUC-secure protocols are "fully simulatable" since the GUC-simulator for attacking the ideal protocol can, in a very practical sense, be run directly by the adversary. This allows the adversary to simulate *the same information* that can be gotten by attacking any session of the real protocol, without the need to actually perform an attack. (Of course, this simulated information is inherently useless to the adversary, since the ideal protocol attacked by the simulation is secure by definition.)

We first observe that if the system is equipped with a "fully interactive trusted party" that realizes, say, $\mathcal{F}_{mcom}$, the multi-session variant of $\mathcal{F}_{com}$, by interacting separately and privately with each session, then we can directly use the protocol of [19] to GUC-realize any "well-formed" functionality. However, we would like to find more reasonable global setup assumptions, and in particular assumptions that require less interaction from the trusted entity. (Indeed, this realization requires the trusted party to perform strictly more work than it would by directly computing the desired functionalities, *i.e.* the trivial realization of ideal model functionalities). Although it is clear that we can achieve fully simulatable protocols by using highly interactive trusted parties to compute functionalities, it seems to be a more difficult problem to realize GUC-secure protocols using an "offline" shared functionality. Indeed, by our earlier impossiblity results, *some* degree of interaction would seem to be essential, so we begin by considering the idea of limiting the interaction to a "registration phase".

## 4.1 The KRK Model

We observe that the "key registration with knowledge (KRK)" setup of [6], can be modified to serve as a shared functionality, allowing us to realize any "well-formed" ideal functionality against non-adaptive ("static") adversaries using the techniques of that work. Although the setup phase is interactive (parties must register their public keys with registration authorities), it is possible to show (with some minor modifications) that the protocol of [6] can allow the trusted party to remain "offline" for all subsequent protocol activity.

**Functionality $\mathcal{G}_{krk}$**

$\mathcal{G}_{krk}$ proceeds as follows, given a (deterministic) key generation function $\mathsf{Gen}$ (with security parameter $\lambda$), running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$:

**Registration:** When receiving a message $(\texttt{register}, \mathsf{sid}, r)$ from party $P_i$ that has not previously registered, compute $(PK_i, SK_i) \leftarrow \mathsf{Gen}^{\lambda}(r)$ and record the tuple $(P_i, PK_i, SK_i)$.

**Retrieval:** When receiving a message $(\texttt{retrieve}, \mathsf{sid}, P_i)$ from party $P_j$ (where $j \neq i$), if there is a previously recorded tuple of the form $(P_i, PK_i, SK_i)$, then return $(\mathsf{sid}, P_i, PK_i)$ to $P_j$. Otherwise return $(\mathsf{sid}, P_i, \perp)$ to $P_r$. When receiving a message $(\texttt{retrieve}, \mathsf{sid}, P_i)$ from party $P_i$, if there is a previously recorded tuple of the form $(P_i, PK_i, SK_i)$, then return $(\mathsf{sid}, P_i, PK_i, SK_i)$ to $P_i$. Otherwise, return $(\mathsf{sid}, P_i, \perp)$ to $P_i$.

Figure 5: The Knowledge-based Key Registration Functionality (similar to that of [6]). Note that each instance of $\mathcal{G}_{krk}$ can only be invoked by the parties of a single protocol session (*i.e.* with a fixed $\mathsf{sid}$). After converting this ideal functionality to a shared functionality, $\bar{\mathcal{G}}_{krk}$, and restricting retrieval of private keys to *corrupt parties only*, it is possible GUC-realize any functionality using only a single public key per-party.

Recall that the KRK setup of [6] is an ideal functionality $\mathcal{G}_{krk}$ (shown in Figure 5), that chooses a private and public key pair for each registered party (namely, each registered ITI), and lets all parties know the value of the public key. In the natural version of the KRK setup, parties are also allowed to retrieve their own secret keys. Since $\mathcal{G}_{krk}$ is not a shared functionality, we have that each instance of a protocol has its own instance of $\mathcal{G}_{krk}$, which does not lend itself to easy implementation. To fix this, we re-formulate $\mathcal{G}_{krk}$ as a shared functionality, $\bar{\mathcal{G}}_{krk}$, that chooses a public key *per PID* rather than per ITI – that is, all ITIs that have the same PID also have the same public key. (This modeling makes $\bar{\mathcal{G}}_{krk}$ significantly easier to implement in a real system, since only one key is required for each party.) Furthermore, we add a simple modeling restriction: we only allow parties to learn their own secret keys if they are corrupt.[6] Using this new $\bar{\mathcal{G}}_{krk}$ setup, the protocol of [6] works (with minor modifications) even in the GUC-security model, provided that (a) party corruptions are non-adaptive, and (b) all parties with the same PID are corrupted together – we call such corruption pattern PID-wise.

**Theorem 4.1.** *The [6] protocol GUC-realizes $\mathcal{F}_{zk}$, even when given access only to $\bar{\mathcal{G}}_{krk}$, as long as the party corruptions are non-adaptive and PID-wise.*

The proof of this theorem is by a natural extension of the proof in [6] to the EUC framework (which is, of course, equivalent to GUC), but surprisingly, we can achieve a much stronger goal than non-adaptive GUC security with interactive setup.

### 4.2 The Augmented CRS Model

Although it may seem that *at least* an interactive "registration phase" is required in order to avoid our earlier impossibility result, we show that something even *less interactive* will suffice. We propose a further simplification of $\bar{\mathcal{G}}_{krk}$, denoted $\bar{\mathcal{G}}_{acrs}$, and a protocol that GUC-realizes $\mathcal{F}_{com}$ (and thus any well-formed functionality) having access only to $\bar{\mathcal{G}}_{acrs}$. Unlike $\bar{\mathcal{G}}_{krk}$, the $\bar{\mathcal{G}}_{acrs}$ shared functionality does not *require* any interaction (much like $\mathcal{F}_{crs}$), but merely offers a one-time use interactive "key retrieval" service to those who choose to use it. Therefore, we refer to this new setup assumption as the *Augmented CRS* (ACRS) model. In particular, protocols realized in the ACRS model will not actually make use of the key retrieval service, since the model only allows corrupt parties to retrieve their keys. Thus, we are assured that honest parties need never communicate interactively with $\bar{\mathcal{G}}_{acrs}$.

Somewhat counter-intuitively, it is even crucial that uncorrupted parties in ACRS model never "bother" to obtain their secret keys from the trusted authority (since even an honest party may inadvertently execute a rogue protocol, which might expose the secret key). Similarly, it is crucial that corrupted parties have access to their secret keys, since otherwise they would be unable to conduct attack simulations. (On a side note, security is still guaranteed to honest parties who obtain their keys and use them to conduct attack simulations *provided that* they only use their keys for simulation purposes. This is a direct consequence of the "equivalence" between the information revealed by attack simulations, and the information that can be obtained via real-world attacks.) To enforce the protocol design criteria that honest parties should not require access to their secret keys, we directly define the $\bar{\mathcal{G}}_{acrs}$ functionality so that it refuses to supply secret keys to honest parties. (Of course, a direct realization of $\bar{\mathcal{G}}_{acrs}$ by a trusted party cannot actually determine which parties are honest, yet intuitively this modeling should still suffice. In fact, it is not problematic even if the real-world trusted party gives keys to honest parties, as long as they are careful to protect their own security by keeping their keys secret.)

More formally, our new shared functionality $\bar{\mathcal{G}}_{acrs}$ is parameterized by two functions, Setup and Extract. It first chooses a random secret value $MSK$ and a public value $PK \leftarrow \mathsf{Setup}(MSK)$, and

[6] The reason for this restriction, and its meaning, will be discussed in further detail in Section 4.2.

publicizes $PK$ (as a CRS). Next, whenever a *corrupted* party with PID *pid* asks for its secret key, $\bar{\mathcal{G}}_{acrs}$ returns the value $SK_{pid} \leftarrow \mathsf{Extract}(PK; pid; MSK)$. The functionality is presented in Figure 6.

**Functionality** $\bar{\mathcal{G}}_{acrs}^{\mathsf{Setup},\mathsf{Extract}}$

**Initialization Phase:** At the first activation, compute a CRS $(PK) \leftarrow \mathsf{Setup}(MSK)$ for a randomly chosen $\lambda$-bit value $MSK$, and record the pair $(PK, MSK)$.

**Providing the public value:** Whenever activated, by any party requesting the CRS, return $PK$ to the requesting party and to the adversary.

**Dormant Phase:** Upon receipt of a message $(\texttt{retrieve}, \mathsf{sid}, P)$ from a *corrupt* party $P$, return the value $SK_P \leftarrow \mathsf{Extract}(PK; P; MSK)$ to $P$. (Receipt of this message from honest parties is ignored.)

Figure 6: The Identity-Based Augmented CRS Functionality

**Comparing $\bar{\mathcal{G}}_{krk}$ and $\bar{\mathcal{G}}_{acrs}$.** The main difference between $\bar{\mathcal{G}}_{acrs}$ and $\bar{\mathcal{G}}_{krk}$ (the global variant of $\mathcal{G}_{krk}$) is that in $\bar{\mathcal{G}}_{acrs}$ there is a single public value, whereas in $\bar{\mathcal{G}}_{krk}$ an extra public value must be given per party identity. Using a paradigm analogous to the identity-based encryption of [7], we avoid the use of per-party public keys and replace them with a single *short* "master public key" (and indeed our constructions use short public keys that depend only on the security parameter). This property, combined with the fact that the parties who follow their protocols never obtain their secret keys, makes $\bar{\mathcal{G}}_{acrs}$ very close in spirit to a global CRS setup as in $\bar{\mathcal{G}}_{crs}$. In fact, in light of the far-reaching impossibility result for $\bar{\mathcal{G}}_{crs}$, $\bar{\mathcal{G}}_{acrs}$ can be regarded as a "minimum interaction" global setup.

We note that, as pointed out in [6], $\bar{\mathcal{G}}_{krk}$ can be naturally implemented by multiple "registration authorities", where no single authority needs to be fully trusted by all. (However, we once again stress that $\bar{\mathcal{G}}_{krk}$ requires *all* parties, *even those who honestly follow their protocols*, to interactively register with a *some* authority and obtain a public key.) Similarly, although $\bar{\mathcal{G}}_{acrs}$ with a short public key would naturally seem to call for a realization by a single trusted entity, the same technique applies and several instances of $\bar{\mathcal{G}}_{acrs}$ may be run by different trusted authorities. Unlike $\bar{\mathcal{G}}_{krk}$, however, parties may participate in protocols while placing their trust in an arbitrary trusted authority, without ever having registered with *any* authority. This is extremely useful for settings where PKIs are not desirable or easy to implement, and where no single "global" authority is available (see *e.g.* [5]).[7]

In the next section, we will prove the following result:

[7] In fact, the protocol we will describe in Section 5 can also support a "graceful failure" approach similar to that outlined in [6], in the scenario where protocol participants do not mutually trust any single authority. That is, by using suitable "graceful" tools (in the case of our protocol, a "graceful" IBTC) , we can ensure full GUC security if trustworthy authorities are used by all parties, and ordinary stand-alone security for party $P$ in the case where only party $P$'s authority is trustworthy (even if party $P$'s own authority is made completely unavailable after publishing its reference string and/or is later corrupted subsequent to the completion of the protocol!). The proof of this claim is a straightforward consequence of our protocol in Section 5, combined with the fact that our IBTC construction in Section 5.2 guarantees hiding even for maliciously chosen public keys (as long as it remains possible to verify the existence of the $\Sigma$-protocol with HVZK property for the signature scheme). Of course, all IBTCs addressed to party $P$ should make use of the reference string published by the authority that $P$ trusts.

**Theorem 4.2.** *There exists a protocol that GUC-realizes $\mathcal{F}_{com}$ given access to $\bar{\mathcal{G}}_{acrs}$. Party corruptions can be adaptive (and in the non-erasure model), as long as they are PID-wise.*

Finally, we note that a GUC secure realization of $\mathcal{F}_{com}$ is indeed sufficient to GUC-realize any "well-formed" *multi-party* functionality. This may be accomplished by first using $\mathcal{F}_{com}$ to realize $\mathcal{F}_{zk}$ (as in [19]), and then using $\mathcal{F}_{zk}$ to realize the "one-to-many" Zero-Knowledge functionality, $\mathcal{F}_{zk}^{1:M}$ (via the technique of [36]). The multi-party protocol compiler from [19] can then be used to yield a UC-secure realization of any well-formed multi-party functionality in the $\mathcal{F}_{zk}^{1:M}$-hybrid model, without using any shared state (thus it is also a GUC-secure realization by Corollary 2.2).

# 5 GUC-Realizing $\mathcal{F}_{com}$ using the $\bar{\mathcal{G}}_{acrs}$ Global Setup

We now describe the construction of a protocol satisfying the conditions of Theorem 4.2, above. When combined with the compiler from [19], such a *fully simulatable* realization of $\mathcal{F}_{com}$ yields a fully simulatable realization of any well-formed two-party or multi-party functionality. Furthermore, we show that, in addition to requiring only the more minimal $\bar{\mathcal{G}}_{acrs}$ setup, our protocol achieves significantly stronger properties than the fully simulatable protocol from [6] realized in the $\bar{\mathcal{G}}_{krk}$ model. (Of course, our protocol can also be trivially modified for use in the $\bar{\mathcal{G}}_{krk}$ model, where it will enjoy the same strengthened security guarantees.)

Firstly, our protocol realizing $\mathcal{F}_{com}$ in the $\bar{\mathcal{G}}_{acrs}$ model remains secure even in the presence of adaptive corruptions (whereas the protocol of [6] does not). Intuitively, adaptive security seems to be difficult to attain in either the $\bar{\mathcal{G}}_{krk}$ or $\bar{\mathcal{G}}_{acrs}$ models, since an adaptive adversary is eventually able to learn nearly all secrets in the system (save only for the random coins of the trusted party). Since the simulator relies on these same secrets to "break" privacy of (corrupt) parties during attack simulations, it would seem that an adversary provided with access to *all* secrets should be able to violate the privacy of past protocol transcripts (allowing it to discern simulated "information-less" interactions from genuine ones). Our protocol manages to avoid this problem through the use of some additional interactivity.

Remarkably, our protocol for realizing $\mathcal{F}_{com}$ will also maintain security of past protocol executions even when the trusted party implementing $\bar{\mathcal{G}}_{acrs}$ is later corrupted (revealing the random coins used to generate the CRS, and leaving the overall system with no secrets at all)! That is, our protocol will guarantee that past transcripts of protocol interactions can *never* be used to compromise the security or deniability of honest parties *even if the trusted party is later corrupted*. Security is only lost when the trusted party acts maliciously *prior to*, or *during* protocol execution. This kind of "forward security" with respect to the trusted party further minimizes the trust assumptions required to realize $\bar{\mathcal{G}}_{acrs}$ in the real-world. For instance, an adversary cannot later coerce the trusted party into breaking the security of an honest party after the completion of the protocol. Such forward security cannot be achieved using the protocol of [6] since knowledge of the secret key allows "extraction" from past commitments, breaking privacy. Similarly, the protocol of [19] also loses all privacy of past transcripts if the trusted party implementing the CRS setup later reveals a trapdoor.

## 5.1 High-level description of the protocol

Our protocol for realizing $\mathcal{F}_{com}$ in the $\bar{\mathcal{G}}_{acrs}$ shared hybrid model, which we call Protocol UAIBC (for UC Adaptive Identity-Based Commitment), relies on two new techniques. First, we construct an *identity-based* trapdoor commitment (IBTC) which enjoys adaptive security. Then we provide a general transformation from any IBTC into a protocol that securely implements $\mathcal{F}_{com}$.

**Constructing IBTC.** In the setting of IBTC a single "master-key" is made public. Additionally, all parties can obtain a private-key that is associated to their party identifier. (Note that this setting corresponds exactly to the interface of $\bar{\mathcal{G}}_{acrs}$.) Intuitively, an IBTC is a commitment scheme with the additional property that a committer who *knows* the receiver's secret-key can *equivocate* commitments (*i.e.*, it can open up commitments to any value, breaking the binding property of the commitment). Furthermore, it should hold that an adversary that obtains the secret-keys of multiple parties, still should not be able to violate the binding property of commitments sent to parties for which it has not obtained the secret-key.

Constructions of IBTCs were previously known in the Random Oracle Model [2, 39]. Here we provide a conceptually simple approach to constructing an adaptively secure IBTC from any one-way function, in the standard model. Our approach relies on the use of Sigma protocols [15], in an approach based on that of [23] (and perhaps surprisingly can result in a very practical protocol). On a very high-level (and very oversimplified) the general idea is as follows: 1) let the master-key be a public-key for a signature scheme, 2) let the secret-key for a party be a signature on its party identifier, and 3) construct a commitment scheme where the reveal phase consists of a "proof" that *either* the revealed value is consistent with the value committed to, *or* the committer knows a signature on the receiver's party identifier (this "proof" must also "hide" which of these two statements actually holds). We mention that the actual instantiation of this idea is somewhat more involved, in order to guarantee adaptive security, and we provide the full details of our construction in Section 5.2.

**From IBTC to GUC Commitments.** Recall that a protocol for realizing $\mathcal{F}_{com}$ must intuitively satisfy two properties (in addition to the traditional binding and hiding properties of any commitment scheme): 1) it must be equivocable, and 2) it must be extractable. We show how to transform any "equivocable" commitment scheme (such as an IBTC) into a protocol for securely realizing $\mathcal{F}_{com}$ (for single bit commitments). Previously similar types of transformations have appeared in the literature (e.g., [19], [8]). Unfortunately all such transformations either require some additional *non-global* setup (and are thus not applicable in out setting), or only work in the case of static security. We now turn our focus to the protocol UAIBC, which GUC-realizes the $\mathcal{F}_{com}$ functionality via a novel transformation of an IBTC from a mere equivocable commitment (in the standard model), to an equivocable *and* extractable commitment secure against adaptive corruptions in the GUC-security model. We remark that our transformation technique can be employed by substituting *any* merely equivocable commitment scheme (such as standard public key based trapdoor commitments) in place of the IBTC in our protocol, and will yield a scheme that is both equivocable and extractable, a general approach that may prove useful in many other contexts.

We now describe some of the details and intuition for our protocol UAIBC. We will show that UAIBC realizes the $\mathcal{F}_{com}$ ideal functionality in a *fully simulatable* manner (even against adaptive adversaries in the non-erasure setting). Furthermore, the protocol UAIBC is *forward secure*, so that past protocol sessions will remain secure even if the trusted party implementing $\bar{\mathcal{G}}_{acrs}$ is later corrupted. We refer to this latter property as "strong full simulatability", since the output transcript of a GUC-simulation will be indistinguishable from a real transcript even to parties in possession of all keys (indeed, even if the parties possess all the random coins used to generate the CRS!). The protocol is given in the combined secure and authenticated channels model (formally denoted as the $\mathcal{F}_{smt}$ and $\mathcal{F}_{auth}$ hybrid models, respectively). In particular, UC-secure channels are used crucially in the proof of security of UAIBC, detailed in Section 5.

On a high-level, protocol UAIBC proceeds as follows. The committer $P_c$ and receiver $P_r$ first perform a coin-tossing to generate a public-key $\rho$ for a dense crypto-system. This coin-tossing requires the receiver to use an IBTC, and has the property that if the committer is corrupted, the outcome of the coin-tossing can be set to any value. After a completed coin-tossing, the committer commits to a single bit $b$ using an IBTC (let $c$ denote this commitment), and additionally sends an auxiliary string $e$: $e$ is either a random string in case $b = 1$, and an encryption to the decommitment information of $c$ if $b = 0$. (We here require that the encryption scheme used has *pseudo-random ciphertexts*.) In the reveal phase, the committer is required to provide correct decommitment information for $c$, and additionally reveal the value encrypted in $e$ in case $b = 0$. We graphically illustrate the operation of this protocol in Figure 7.

We now take a brief detour to specify the security properties of the IBTC scheme and the dense crypto-system required by our protocol, before proceeding with more in-depth discussion of the protocol.

### 5.2 Identity-based Trapdoor Commitments

The tool we now define, which we dub Identity-based Trapdoor Commitments (IBTCs), represents a slight generalization of the identity-based chameleon hash functions first introduced in [2]. Any Identity-Based Chameleon Hash function is sufficient (but not *necessary*) to yield an IBTC. We introduce IBTCs in order to accommodate our suggested constructions in Section 4.

**Definition 3** ( IBTC). *An Identity-based Trapdoor Commitment scheme $IC$ is given by a* 5*-tuple of* PPT *algorithms $IC = (\mathsf{Setup}, \mathsf{Extract}, \mathsf{Com}, \mathsf{ECom}, \mathsf{Eqv})$, with the following basic properties:*

- $\mathsf{Setup}$: A deterministic algorithm which takes as input a (uniformly random) "master secret key" $MSK \in \{0,1\}^\lambda$ (where $\lambda$ is a security parameter), and outputs a public key $PK$.
- $\mathsf{Extract}$: On input $(ID, MSK)$ outputs a trapdoor $SK_{ID}$ for identity $ID$.
- $\mathsf{Com}$: A deterministic algorithm which takes as input a tuple of the form $(PK, ID, d, m)$, and outputs a commitment $\kappa$ for message $m$ under identity $ID$ using random input $d$. The domain from which $d$ is chosen is denoted by $\mathcal{D}$. As a shorthand, we may write $\mathsf{Com}_{ID}(d, m)$ to denote $\mathsf{Com}(PK, ID, d, m)$.
- $\mathsf{ECom}$: On input $(PK, ID, SK_{ID})$ outputs a pair $(\kappa, \alpha)$, to be used with $\mathsf{Eqv}$.
- $\mathsf{Eqv}$: On input $(PK, ID, SK_{ID}, \kappa, \alpha, m)$ produces a value $d \in \mathcal{D}$ such that $\mathsf{Com}_{ID}(d, m) = \kappa$. That is, $d$ makes $\kappa$ appear to be a commitment to $m$.

Following conventional terminology, we refer to $\kappa$ as a "commitment" for identity $ID$ and the pair $(d, m)$ as the corresponding "decommitment". A commitment $c$ for identity $ID$ is said to "open to" a certain message $m$ using the decommitment pair $(d, m)$ if $\mathsf{Com}(PK, ID, d, m)$.

**Remark:** Throughout this work, we focus on obtaining security in the presence of adaptive corruptions. Therefore, we only consider IBTCs that utilize the random input to $\mathsf{Com}$ as the decommitment, rather than a more general notion that allows for $\mathsf{Com}$ to generate a separate decommitment value. Similarly, we require all the random coins input to $\mathsf{Setup}$ to come from $MSK$ in order to enable forward security (*i.e.* to tolerate adaptive corruption of the trusted setup party).

We are now ready to state the formal security properties of Identity-based Trapdoor Commitments. Roughly speaking, they consist of two natural computational security properties known as "binding" and "equivocability". The binding property states that any adversary, given only

| Step # | $P_c(b)$ | | $P_r$ |
|---|---|---|---|
| **Commit Phase** | | | |
| (1) | | $\xrightarrow{\texttt{commit},sid,P_c,P_r}$ | |
| (2) | | $\xleftarrow{\kappa_1}$ | $\rho_1 \xleftarrow{\$} \Phi$<br>$d_1 \xleftarrow{\$} \mathcal{D}$<br>$\kappa_1 \leftarrow \mathsf{Com}_{P_c}(d_1, \rho_1)$ |
| (3) | $\rho_2 \xleftarrow{\$} \Phi$ | $\xrightarrow{\rho_2}$ | |
| (4) | | $\xleftarrow{d_1,\rho_1}$ | $\rho \leftarrow \rho_1 \cdot \rho_2$ |
| (5) | $\kappa_1 \stackrel{?}{=} \mathsf{Com}_{P_c}(d_1, \rho_1)$<br>$\rho \leftarrow \rho_1 \cdot \rho_2$<br>$d \xleftarrow{\$} \mathcal{D}$<br>$\kappa \leftarrow \mathsf{Com}_{P_r}(d, b)$<br>if $b = 0$ $\psi \leftarrow E_\rho(r, d)$<br>if $b = 1$ $\psi \leftarrow$ random | $\xrightarrow{\kappa,\ \psi}$ | |
| **Reveal Phase** | | | |
| (1&2) | if $b = 0$<br>if $b = 1$ | $\xrightarrow{\texttt{b=0},\ d,\ r}$<br>$\xrightarrow{\texttt{b=1},\ d}$ | if $b = 0$ $E_\rho(r, d) \stackrel{?}{=} \psi$<br>$\kappa \stackrel{?}{=} \mathsf{Com}_{P_r}(d, b) \Rightarrow b$ |

Figure 7: Operation of Protocol $\mathsf{UAIBC}$, with party $P_c$ committing bit $b$ to party $P_r$. Note that $\mathsf{Com}$ is the commitment operation of an IBTC (the subscript is the identity of the intended recipient), and $E_\rho$ is a Dense PRC secure encryption using public key $\rho$ (the first input is the random coins fed to the encryption operation, and the second is the plaintext). Steps 2 to 4 of the **Commit** phase are essentially a coin-tossing protocol, whereas the subsequent steps are similar to the protocol of [19].

knowledge of $PK$ and the secret keys of some users *other* than $ID$, cannot produce a commitment for $ID$ with two different openings. The equivocability property guarantees that the output of $\mathsf{Eqv}$ is indistinguishable from a uniformly random choice of $d$, even to an adversary who knows $MSK$ (this is required for forward security reasons). In other words, $\mathsf{Eqv}$ allows one to open a special "equivocable commitment" $\kappa$ produced by $\mathsf{ECom}$ in such a way that $\kappa$ is indistinguishable from a normal commitment to $m$ generated via $\mathsf{Com}(PK, ID, d, m)$ (rather than $\mathsf{ECom}$), even though $m$ had not yet been specified when $\kappa$ was actually produced using $\mathsf{ECom}$.

We give a general construction of IBTCs in Section 5.2.2, and we observe that they can be built from one-way functions (although more efficient number-theoretic constructions are possible as well).

**Security of Identity-based Trapdoor Commitments.** Identity-based Trapdoor Commitment schemes must satisfy the following security requirements.

- **Binding -** Every PPT adversary $\mathcal{A}$ wins the following game with negligible probability:

1. The challenger computes $MSK \xleftarrow{\$} \{0,1\}^\lambda$, $PK \leftarrow \mathsf{Setup}(MSK)$, and then sends $PK$ to $\mathcal{A}$.
2. $\mathcal{A}$ is allowed to query an oracle for $\mathsf{Extract}(PK, \cdot, MSK)$ (many times).
3. $\mathcal{A}$ outputs $(ID, d, m, d', m')$.
4. $\mathcal{A}$ wins if $ID$ was not submitted to the $\mathsf{Extract}$ oracle in Step 2 and $m \neq m'$, but $\mathsf{Com}_{ID}(d, m) = \mathsf{Com}_{ID}(d', m')$.

Attack Game 1: Binding Attack Game (for IBTCs)

- **Equivocability -** Every PPT adversary $\mathcal{A}$ has at most negligible advantage in the following game:

1. The challenger computes $MSK \xleftarrow{\$} \{0,1\}^\lambda$, $PK \leftarrow \mathsf{Setup}(MSK)$, and then sends $MSK$ to $\mathcal{A}$ (unlike the previous game, where only $PK$ was provided to $\mathcal{A}$).
2. $\mathcal{A}$ chooses an identity $ID$ and a message $m$, and sends $(ID, m)$ to the challenger.
3. The challenger chooses $b \in \{0,1\}$ at random, and computes $d \in \mathcal{D}$ in one of two ways, depending on $b$:
   - if $b = 0$, then $d \xleftarrow{\$} \mathcal{D}$;
   - if $b = 1$, then $SK_{ID} \leftarrow \mathsf{Extract}(PK, ID, MSK)$, $(\kappa, \alpha) \leftarrow \mathsf{ECom}(PK, ID, SK_{ID})$, $d \leftarrow \mathsf{Eqv}(PK, ID, SK_{ID}, \kappa, \alpha, m)$.

   The challenger then sends $d$ to $\mathcal{A}$.
4. $\mathcal{A}$ outputs a "guess" $\hat{b} \in \{0,1\}$.
5. $\mathcal{A}$'s advantage is defined to be $|\Pr[\,\hat{b} = b\,] - 1/2|$.

Attack Game 2: Equivocability Attack Game (for IBTCs)

**Constructions of Identity-based Trapdoor Commitments.** The following theorem summarizes the results of our IBTC construction technique:

**Theorem 5.1.** *There exists an efficient construction of an IBTC from any signature scheme with an "augmented Σ-protocol". In particular, such signature schemes exist if one-way functions exist, and can be efficiently constructed under the Strong RSA assumption.*

The proof follows directly from the details of the construction, which we now outline.

### 5.2.1 Augmented Σ-Protocols

Our approach is based on the technique from [23] for constructing commitment schemes from "Σ-protocols". In particular, the basic tool we use to construct IBTCs is a modified form of standard Σ-protocols (see [34, 21]) that we call augmented Σ-protocols. Intuitively, Σ-protocols are three move protocols for proving some relation in zero knowledge. The augmented definition adds a "reverse state construction" property that enables the commitment schemes we construct to tolerate adaptive adversaries (who are able to learn the internal state of corrupted parties). We do this by constructing "fake" state information for honest runs of the Σ-protocol (which require the prover to know the witness), in order to make them appear as if they were instead output by the zero-knowledge simulator (which does not require knowledge of the witness).Augmented Σ-protocols are closely related to the "non-erasure" Σ-protocols of [21], but require a slightly different state construction property (one that is roughly the "reverse" of the corresponding property in [21]).

**Definition 4.** *Let $X, W$, and $L \subset X$ be finite sets. We refer to an element $x \in X$ as an "instance", an element $w \in W$ as a "witness", and the set $L$ as a "language". Furthermore, assume there is an efficiently testable binary relation $R$, such that for all $x \in L$ we have $(x, w) \in R$ for some $w \in W$. (Here we define "efficiently testable" to mean $R$ runs in time polynomial in $|x|$, and therefore we must also have that $|w| \leq \mathsf{poly}(|x|)$.)*

*Consider a three move protocol run between a prover $P$, with input $(x, w) \in R$, and a verifier $V$ with input $x$, executed as follows. $P$ chooses some random coins $r_a$, computes $a \leftarrow A(x, w, r_a)$, and sends $a$ to $V$. $V$ then chooses a random string $c$ (the "challenge"), and sends it to $P$. Finally, $P$ computes and responds with $z \leftarrow Z(x, w, r_a, c)$. The verifier $V$ then computes and outputs the result of $B(x, a, c, z)$, which returns either* $\mathsf{accept}$ *or* $\mathsf{reject}$. *We require that $A$, $Z$, and $B$ be (deterministic) poly-time algorithms, and that $|c| \leq \mathsf{poly}(|x|)$. Such a protocol is called an* Augmented Σ-Protocol *for the language $L$ if it satisfies the following properties:*

- **Completeness -** If $(x, w) \in R$ then an interaction between $P(x, w)$ and $V(x)$ – following the procedure outlined above – causes the verifier to output $\mathsf{accept}$ with overwhelming probability.

- **Special HVZK (Honest Verifier Zero Knowledge) -** There exists a PPT algorithm $\mathsf{ZKSim}$ (a "Zero-Knowledge simulator"), such that every PPT distinguisher $D$ has at most negligible advantage in the following game:

1. $D$ computes $(x, w) \in R$ along with a challenge $c$, and sends $(x, w, c)$ to the challenger.
2. The challenger chooses $b \in \{0, 1\}$ and some coins $r$ at random, and computes the following:

   if $b = 0$: $a \leftarrow A(x, w, r)$, $z \leftarrow Z(x, w, r, c)$

   if $b = 1$: $(a, z) \leftarrow \mathsf{ZKSim}(x, c, r)$

   The challenger then sends $(a, z)$ to $D$.
3. $D$ outputs a guess $\hat{b} \in \{0, 1\}$.
4. $D$'s advantage is defined to be $|\Pr[\, b = \hat{b} \,] - 1/2|$.

Attack Game 3: Special HVZK Attack Game

To simplify analysis, we additionally require that $\mathsf{ZKSim}$ must always output either a "valid" pair $(a, z)$ (*i.e.* such that $B(x, a, c, z)$ outputs $\mathsf{accept}$), or an error condition.[8]

**Intuition:** Essentially, this means that for any efficiently computable challenge $c$, it is possible to produce a protocol transcript computationally indistinguishable from an actual run with a prover (who knows $w$), even without knowledge of $w$. Furthermore, we require that the indistinguishability holds even if the distinguisher knows the witness $w$. (This additional requirement is slightly stronger than the usual HVZK property.) We chose to define the HVZK security property in the form of an interactive game in order to ensure that the challenge is generated computationally, which facilitates the use of certain number theoretic assumptions (such as Strong RSA) for constructing Augmented $\Sigma$-protocols.

- **Reverse State Construction -**

  There exists a PPT algorithm $\mathsf{RSC}$ such that every PPT distinguisher $D$ has at most negligible advantage in the following game:

1. $D$ computes $(x, w) \in R$ along with a challenge $c$, and sends $(x, w, c)$ to the challenger.
2. The challenger chooses some coins $r_a$ and $r_s$ at random, and computes $(a, z) \leftarrow \mathsf{ZKSim}(x, c, r_s)$, $a' \leftarrow A(x, w, r_a)$, $z' \leftarrow Z(x, w, r_a, c)$, and $r'_s \leftarrow \mathsf{RSC}(x, w, r_a, c)$. Then the challenger chooses $b \in \{0, 1\}$, and proceeds as follows:

   if $b = 0$: Send $(a, z, r_s)$ to $D$.

   if $b = 1$: Send $(a', z', r'_s)$ to $D$.
3. $D$ outputs a guess $\hat{b} \in \{0, 1\}$.
4. $D$'s advantage is defined to be $|\Pr[\, b = \hat{b} \,] - 1/2|$.

Attack Game 4: Reverse State Construction Attack Game

[8]Note that this requirement does *not* further restrict the class of languages with Augmented $\Sigma$-protocols.

**N.B.:** For the case $b = 1$, the values of $a'$ and $z'$ being supplied to $D$ come from the outputs of $A$ and $Z$, not $\mathsf{ZKSim}$. Therefore, in order for the values of $a'$ and $z'$ observed by $D$ to remain consistent with $r'_s$, it must be true with overwhelming probability that $\mathsf{ZKSim}(x, c, \mathsf{RSC}(x, w, r_a, c))$ (the "simulation" output resulting from the coins produced by the Reverse State Construction procedure) yields the same choice of $a'$ and $z'$ as are produced by $A$ and $Z$ when using the random coins $r_a$. Otherwise, $D$ could easily observe the inconsistency and would correctly guess that $\hat{b} = 1$ whenever a discrepancy occurs, allowing $D$ to achieve a non-negligible advantage.

**Intuition:** This property is not typically required of $\Sigma$-protocols, hence we specify that the protocols are "Augmented" in the nomenclature. This additional requirement ensures that after a standard run of the $\Sigma$-protocol, it is possible for the prover to find a pseudo-random input for the simulator that causes it output the same transcript. Furthermore, even a distinguisher who knows the witness cannot determine whether those pseudo-random coins were chosen using this reconstruction procedure or not (*i.e.* if they are truly random). It is easy to verify that this property actually implies the above Special HVZK property of $\Sigma$-protocols as well, but we state it separately to clarify the relationship with standard $\Sigma$-protocols (which instead feature the Special HVZK property, or perhaps a slightly weaker HVZK property). The reasons for this additional requirement will become clear in Section 5.2.

- **Special Soundness -** There exists a PPT algorithm $\mathcal{E}_{\mathrm{rw}}$ (a "rewinding extractor"), such that every PPT adversary $D$ wins the following game with at most negligible probability:

1. $D$ computes $(x, a, c, z, c', z')$, where $c \neq c'$, and sends it to the challenger.
2. The challenger computes $w \leftarrow \mathcal{E}_{\mathrm{rw}}(x, a, c, z, c', z')$.
3. $D$ wins if both $B(x, a, c, z)$ and $B(x, a, c', z')$ return $\mathsf{accept}$, but $(x, w) \notin R$.

Attack Game 5: Special Soundness Attack Game

### 5.2.2 An IBTC from Signature Schemes with Augmented $\Sigma$-Protocols

Let $\Upsilon = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ denote any signature scheme that is existentially unforgeable against chosen message attack (*i.e.* it is UF-CMA secure). Define a relation $R^{\Upsilon}(x, w)$ such that $(x, w) \in R^{\Upsilon}$ if either 1) $x = \langle VK, m \rangle$ and $w = \sigma$ such that $\mathsf{Ver}(VK, m, \sigma) = \mathsf{accept}$; or 2) $w = r_g$ such that $(VK, SK) = \mathsf{Gen}(r_g, 1^{\lambda})$ and $SK$ is a valid signing key corresponding to $VK$ (that is, $w$ contains all the random coins used by the $\mathsf{Gen}$ algorithm when producing $VK$ and $SK$). We say that $\Upsilon$ has an Augmented $\Sigma$-Protocol if there is an Augmented $\Sigma$-Protocol for the relation $R^{\Upsilon}$.

**Remark:** The second case of $R^{\Upsilon}$, which allows $r_g$ to be used as a witness, is necessary to ensure that the properties of the Augmented $\Sigma$-protocol for the signature scheme are preserved even with respect to adversaries who eventually learn $r_g$. We ultimately rely on this property for "forward security" purposes, in case the trusted setup party who generates $VK$ is later corrupted. Of course, since $\mathsf{Gen}(r_g, 1^\lambda)$ yields the signing key $SK$, if the witness is $r_g$ we can use it to compute a signature witness $\sigma$ satisfying the form of the first case. Therefore, if there is an Augmented $\Sigma$-protocol for signatures, we can easily transform it into an Augmented $\Sigma$-protocol for $R^\Upsilon$ by simply having provers with a witness $r_g$ compute the appropriate signature witness, and then run the original Augmented $\Sigma$-protocol using the signature witness (and, of course, a prover with a signature witness simply runs the original protocol). This transformation does not at all impact the efficiency of the original protocol, and it is easily verified that all properties of Augmented $\Sigma$-protocols hold for the transformed protocol with respect to the relation $R^\Upsilon$. Therefore, we observe that it is sufficient to have an Augmented $\Sigma$-protocol for the signature scheme alone, despite our use of the slightly more involved relation $R^\Upsilon$.

Given such a signature scheme, we can construct an Identity-based Trapdoor Commitment scheme as follows:

- $\mathsf{Setup}(1^\lambda)$**:** Compute $(VK, SK) \leftarrow \mathsf{Gen}(r_g; 1^\lambda)$, where $r_g$ denotes the random coins used by the $\mathsf{Gen}$ algorithm. Return the pair $(PK = VK, MSK = r_g)$.
- $\mathsf{Extract}(PK, ID, MSK)$**:** Run $\mathsf{Gen}(MSK, 1^\lambda)$ to obtain $SK$. Compute $\sigma_{ID} = \mathsf{Sig}(SK, ID)$. Return $SK_{ID} = \sigma_{ID}$.
- $\mathsf{Com}(PK, ID, d, m)$**:** Compute $(a, z) \leftarrow \mathsf{ZKSim}(x = \langle PK, ID\rangle, c = m, r_s = d)$, where $\mathsf{ZKSim}$ is the $\Sigma$-protocol simulator for $R^\Upsilon$. Return the commitment $\kappa = a$.
- $\mathsf{ECom}(PK, ID, SK_{ID})$**:** Choose random coins $r_a$ and compute $a = A(x = \langle PK, ID\rangle, w = SK_{ID}, r_a)$. Return the pair $(\kappa = a, \alpha = r_a)$.
- $\mathsf{Eqv}(PK, ID, SK_{ID}, \kappa, \alpha, m)$**:** Compute $r_s = \mathsf{RSC}(x = \langle PK, ID\rangle, w = SK_{ID}, r_a = \alpha, c = m)$. Return $d = r_s$.

**Intuition:** The message $m$ is used as the "challenge" flow on which to run the Zero-Knowledge simulator for the $\Sigma$-protocol, producing the prover's first flow $a$ and a response $z$. Only the first value, $a$, is used to form the commitment $\kappa$. The decommitment $d$ consists of exactly the random coins $r_s$ used to conduct the simulation. The prover's response flow $z$ is not used at all. This is a slight departure from the technique of [23], which instead uses $z$ as the decommitment and requires the recipient to verify that $B(a, c, z)$ outputs $\mathsf{accept}$. In this work, we allow the eventual (adaptive) corruption of the committer, in which case all random coins used in the HVZK simulation must be revealed. Since security must hold against an adversary who can eventually learn the random coins $r_s$ (used by the HVZK simulation algorithm $\mathsf{ZKSim}$ to produce the pair $(a, z)$ for the $\Sigma$-protocol), we can simplify our protocols by allowing the random coins $r_s$ to serve directly as the decommitment. Never the less, when considering specific instantiations, in practice it may be more efficient to send $z$ as the decommitment (using the approach of [23]) rather than $r_s$ (even though $r_s$ must still be provided to the adversary when the committer is corrupted).

Equivocation of a commitment is achieved by using knowledge of the witness $w$ (*i.e.* the equivocation trapdoor $SK_{ID}$) to "honestly" conduct the $\Sigma$-protocol proof (instead of simulating) so that the prover can later respond to any "challenge" (thus opening the commitment to any message), even though the first flow was already sent as the commitment. Once again, we avoid the need to send the response flow $z$ by sending the random coins $r_s$ as the decommitment instead, and, therefore, we make use of the Reverse State Construction algorithm to produce the decommitment.

**Theorem 5.2.** *The construction described above yields a secure IBTC scheme.*

*Proof.* It is easy to verify the correctness of the above IBTC implementation. We now proceed to prove the security properties.

- **Binding.** Suppose there is an adversary $\mathcal{A}$ breaking the binding property of the commitment scheme. We describe a reduction that forges signatures for $\Upsilon$ (in the UF-CMA attack game) by acting as the challenger for $\mathcal{A}$ in the security game for the binding property of the IBTC as follows:

  1. To forge a signature in a UF-CMA attack game with verification key $VK$, the challenger sets $PK = VK$ and sends $PK$ to $\mathcal{A}$.
  2. The challenger then responds to $\mathcal{A}$'s queries to the $\mathsf{Extract}(PK, \cdot, MSK)$ oracle by simply forwarding them to the signature oracle in the UF-CMA game, and returning the responses. (It is easy to verify that this yields the correct output.)
  3. Wait until $\mathcal{A}$ outputs a tuple $(ID, d, m, d', m')$.
  4. If $\mathcal{A}$ wins the binding attack game, then $ID$ was never queried to the oracle, and $\kappa = \mathsf{Com}_{ID}(d, m) = \mathsf{Com}_{ID}(d', m')$. In this case, we may compute $(\kappa, z) \leftarrow \mathsf{ZKSim}(\langle PK, ID\rangle, m, d)$ and $(\kappa, z') \leftarrow \mathsf{ZKSim}(\langle PK, ID\rangle, m', d')$. That is, we now have two accepting transcripts for the Augmented $\Sigma$-protocol which begin with the same first flow, enabling us to use the rewinding extractor (from the soundness property) to obtain: $w \leftarrow \mathsf{Ext}(\langle PK, ID\rangle, \kappa, m, z, m', z')$. With overwhelming probability, $w$ is a valid witness and therefore either contains a signature $\sigma_{ID}$ such that $\mathsf{Ver}(PK, ID, \sigma_{ID})$ accepts, or the random coins needed to recover the signing key itself (in which case it is easy to produce a forgery). Since $ID$ was never queried to the oracle in the binding attack game, the reduction never queried it to the signing oracle in the UF-CMA game, and thus the reduction may output $\sigma_{ID}$ as a new forgery.

  That is, whenever $\mathcal{A}$ succeeds in breaking the binding property of our IBTC construction, the reduction outputs a forgery with overwhelming probability. Clearly then, if the signature scheme is unforgeable, the binding property must hold.

- **Equivocability.** Equivocability of commitments follows immediately from the Reverse State Construction property of Augmented $\Sigma$-protocols. Clearly, any adversary capable of distinguishing the output of $\mathsf{Eqv}$ from random coins (when given access to $MSK$), can also be used to distinguish the output of $\mathsf{RSC}$ (when using a witness containing $r_g$). That is, set $x = \langle PK, ID\rangle$ and $w = r_g$ in the Reverse State Construction game, and feed the resulting challenge to the adversary who breaks the equivocability of the commitment scheme, then output the same guess as the equivocability adversary. It is easy to verify that the resulting advantage in the Reverse State Construction game is the same as that of the adversary attacking the equivocability property of the commitment scheme.

$\square$

### 5.2.3 Signature Schemes with Augmented $\Sigma$-Protocols

To complete the construction of IBTCs, it remains to construct a signature scheme with an augmented $\Sigma$-protocol for the knowledge of the signature. First, we observe that every $NP$-relation is known to have such a $\Sigma$-protocol if one-way functions exist [25, 29, 4]. Specifically, the protocol of [4] (where the prover commits to a permutation of a graph with a Hamiltonian cycle, and is challenged to reveal either a cycle or the permutation) is easily shown to support the requirements of augmented $\Sigma$-protocols when using pseudo-random commitments. In particular, the reverse state construction property can be achieved by simply producing random coins corresponding to the graph opened by the simulator, along with coins yielding the same random commitment values used by all the unopened commitments to edges in the graph (relying on the pseudo-randomness of the commitments).

Since signature schemes can also be constructed from one-way functions, we immediately obtain a generic construction of IBTCs based on one-way functions. We also notice that we can have more efficient constructions if we use certain signature schemes based on specific number-theoretic assumptions. For example, we can use any "signature scheme with efficient protocols", such as the one of [16] based on strong RSA.

## 5.3 Dense PRC Secure Encryption

The commitment protocol of [19] (which uses a common reference string) requires the use of a CCA secure public key encryption scheme with Pseudo-Random Ciphertexts (PRC). The CCA security requirement arises due to the fixed nature of the global public encryption key, which is reused during every run of the protocol. Our protocol, on the other hand, will not make use of any fixed public keys for encryption. Instead, we will be using a coin-flipping protocol to choose a fresh public key for each run of our commitment protocol (this is discussed in more detail below). One advantage of doing this is that we will require only semantically secure encryption with PRC, since each key is used only once. However, there is also a disadvantage: the public key is chosen as the result of a fair coin-flipping protocol (which will be uniformly random in some finite group), so the encryption scheme must have "dense" public keys. Informally, this means we require public keys to be computationally indistinguishable from random. Fortunately, schemes satisfying these requirements exist under widely used computational assumptions, such as the DDH assumption (in fact, standard El-Gamal encryption suffices). The formal definition of security for Dense PRC encryption provided below has the flavor of "left-or-right" security, which is most convenient for the proof of security of our protocol.

**Definition 5** (Dense PRC Encryption)**.** *A public key encryption scheme* PRC $= (G, E, D)$*, satisfying the standard correctness properties for public key encryption, is said to be* Dense PRC *secure if it satisfies the following additional constraints:*

1. *Public keys produced by $G$ lie within some abelian group $\Phi$, with efficiently computable inverse and group operations.*
2. *Membership in $\Phi$ is efficiently testable.*
3. *The uniform distribution on $\Phi$ is efficiently samplable.*

4. *The distribution of public keys produced by $G$ is computationally indistinguishable from the uniform distribution on $\Phi$.*

5. *For all* PPT *adversaries $\mathcal{A}$:*

$$\Pr[A^{LR(\cdot,0)}(1^\lambda) = 1] - \Pr[A^{LR(\cdot,1)}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

*where the answer to a query of the oracle $LR(m, b)$ is computed by obtaining $(K, K^{-1}) \leftarrow G(1^\lambda)$ and returning the pair $\langle K, E_K(m)\rangle$ if $b = 0$, or returning the uniformly random pair $\langle U, R\rangle$ if $b = 1$ (where $U$ is chosen from a group $\Phi$, and $R$ from $\{0,1\}^{|E_U(m)|}$). We need not allow $\mathcal{A}$ to query the oracle more than once, but this restriction is unimportant as long as fresh keys are computed on each query to the oracle.*

**Intuition:** The final condition in the definition captures the main security property of Dense PRC: an honestly generated public key (output by $G$) and an honestly generated ciphertext (output by $E$) are indistinguishable from a randomly chosen public key and a random bit string of appropriate length. Note that it must be possible to encrypt messages to random public keys, and that the result ciphertexts must still appear to be random (since otherwise we would have a way to distinguish random public keys from honestly generated ones). This is important for our purposes, since our protocol requires parties to encrypt messages under a randomly generated public key. The remaining properties of Dense PRC schemes are merely technical requirements that enable two parties to jointly select a random public key via a standard "coin-flipping protocol".

### 5.4 Details and Design of Protocol UAIBC

A full description of the protocol UAIBC in the $\bar{\mathcal{G}}_{acrs}$ (Augmented CRS) model is shown in Figure 8 below.

Intuitively, the final flow of the commit phase of the UAIBC protocol is very similar to the UAHC protocol of [19] and the CLOS-KR protocol of [6]. The actual structure of the last flow in UAIBC is slightly different from the UAHC and CLOS-KR protocols since it uses a single ciphertext, rather than two ciphertexts as the UAHC and CLOS-KR protocols do. The only purpose of this minor change is to further simplify the presentation of the protocol and the proof, and indeed it would seem that the same change can be safely made to the UAHC and CLOS-KR protocols.

The novelty of protocol UAIBC is its use of a coin-flipping protocol to choose the public key for the encryption scheme (used when producing $\psi$), rather than using a globally fixed key as in [19], or public keys belonging to the individual parties as in [6]. The net result of this coin-flipping protocol is that no party (trusted or otherwise) should ever be in possession of the decryption key, unless perhaps some "cheating" occurred during the coin-flipping process (as we will discuss below).

Of course, the use of this coin-flipping protocol makes the commitment phase of the protocol interactive (whereas in [19] it was not). We remark that some interactivity is unavoidable when constructing adaptively secure GUC commitment schemes in the Augmented CRS model.[9]

Simulation of the UAIBC protocol is slightly involved. On a high level, as with all UC-secure realizations of $\mathcal{F}_{com}$, the simulator must be able to extract the actual committed values being sent

[9]This claim will be proven formally in an extended version of this paper.

**Protocol** $\mathsf{UAIBC}$

Let $(G, E, D)$ be a Dense PRC secure encryption scheme. We denote the key-space of the dense encryption scheme by $\Phi$, and the security parameter by $\lambda$. Then Protocol $\mathsf{UAIBC}$ proceeds as follows, with party $P_c$ committing a bit $b$ to party $P_r$, in the shared $\bar{\mathcal{G}}_{acrs}^{\mathsf{Setup},\mathsf{Extract}}$ hybrid model. (Note, we also assume ideal private and authenticated channels for all communications.)

**Common Reference String:** All parties are initialized with the common reference string produced by $\bar{\mathcal{G}}_{acrs}^{\mathsf{Setup},\mathsf{Extract}}$ (the public key for the IBTC scheme produced by $\mathsf{Setup}$, which is an implicit parameter to the $\mathsf{Com}$ and $\mathsf{Open}$ algorithms). In particular, we note that $\bar{\mathcal{G}}_{acrs}^{\mathsf{Setup},\mathsf{Extract}}$ is a shared subroutine of Protocol $\mathsf{UAIBC}$.

**Commit Phase:**

1. On input $(commit, sid, P_c, P_r, b)$ where $b \in \{0, 1\}$, party $P_c$ sends $(\mathtt{com}_0, sid, P_c, P_r)$ to $P_r$. It then waits to receive a message of the form $(\mathtt{com}_1, sid, P_c, P_r, \kappa_1)$ from $P_r$.
2. $P_r$ receives $(\mathtt{com}_0, sid, P_c, P_r)$ and records it, ignoring any later $\mathtt{com}_0$ messages from $P_c$ with the same $sid$. $P_r$ chooses a random $\rho_1 \in \Phi$ and a decommitment $d_1 \xleftarrow{\$} \mathcal{D}$, and records them. $P_r$ then computes $\kappa_1 \leftarrow \mathsf{Com}_{P_c}(d_1, \rho_1)$, sends $(\mathtt{com}_1, sid, P_c, P_r, \kappa_1)$ to $P_c$ and waits to receive the message of the form $(\mathtt{com}_2, sid, P_c, P_r, \rho_2)$ from $P_c$ in response.
3. $P_c$ receives and records $(\mathtt{com}_2, sid, P_c, P_r, \kappa_1)$, ignoring any later $\mathtt{com}_2$ messages from $P_c$ with the same $sid$. $P_c$ then chooses a random $\rho_2 \in \Phi$, sends the message $(\mathtt{com}_3, sid, P_c, P_r, \rho_2)$ to $P_r$, and waits to receive of a message of the form $(\mathtt{com}_3, sid, P_c, P_r, d_1, \rho_1)$ from $P_r$.
4. $P_r$ receives $(\mathtt{com}_2, sid, P_c, P_r, \rho_2)$ and records it, ignoring any later $\mathtt{com}_2$ messages from $P_c$ with the same $sid$. $P_r$ computes and stores $\rho \leftarrow \rho_1 \cdot \rho_2$, and then sends $(\mathtt{com}_3, sid, P_c, P_r, d_1, \rho_1)$ to $P_c$ and waits to receive a message of the form $(\mathtt{com}_4, sid, P_c, P_r, \kappa, \psi)$ from $P_c$.
5. $P_c$ receives and records $(\mathtt{com}_3, sid, P_c, P_r, d_1, \rho_1)$ from $P_r$, ignoring any later $\mathtt{com}_3$ messages from $P_c$ with the same $sid$. $P_c$ then verifies that $\kappa_1 = \mathsf{Com}_{P_c}(d_1, \rho_1)$, and ignores the message if verification fails. If verification succeeds, $P_c$ computes $\rho \leftarrow \rho_1 \cdot \rho_2$, then chooses $d \leftarrow \mathcal{D}$ and records it along with $\rho$. Additionally, if $b = 0$, $P_c$ chooses and records random coins $r$, and then computes $\psi \leftarrow E_\rho(r, d)$; otherwise, when $b = 1$, $P_c$ chooses $\psi$ at random. $P_c$ then sends $(\mathtt{com}_4, sid, P_c, P_r, \kappa, \psi)$ to $P_r$.
6. $P_r$ receives $(\mathtt{com}_4, sid, P_c, P_r, \kappa, \psi)$ from $P_c$ and records it, ignoring any later $\mathtt{com}_4$ messages from $P_c$ with the same $sid$. $P_r$ then outputs $(\mathtt{receipt}, sid, P_c, P_r)$.

**Reveal Phase:**

1. On input $(reveal, sid)$ party $P_c$ retrieves the necessary records and, if $b = 0$, sends $(\mathtt{rev}, sid, P_c, P_r, d, 0, r)$ to $P_r$; otherwise, $P_c$ sends $(\mathtt{rev}, sid, P_c, P_r, d, 1)$ to $P_r$.

2a. When receiving a message of the form $(\mathtt{rev}, sid, P_c, P_r, d, 0, r)$, $P_r$ records it, ignoring all subsequent messages from $P_c$ with the same $sid$. $P_r$ then retrieves the values of $\rho$, $\kappa$, and $\psi$ corresponding to $sid$ from its records and verifies that $E_\rho(r, d) = \psi$ and $\mathsf{Com}_{P_r}(d, 0) = \kappa$. If the verification succeeds, $P_r$ outputs $(reveal, sid, P_c, P_r, 0)$. Otherwise, $P_r$ ignores the message.

2b. When receiving a message of the form $(\mathtt{rev}, sid, P_c, P_r, d, 1)$, $P_r$ records it, ignoring all subsequent messages from $P_c$ with the same $sid$. $P_r$ then retrieves the value of $\kappa$ corresponding to $sid$ from its records and verifies that $\mathsf{Com}_{P_r}(d, 1) = \kappa$. If the verification succeeds, $P_r$ outputs $(reveal, sid, P_c, P_r, 1)$. Otherwise, $P_r$ ignores the message.

Figure 8: Protocol $\mathsf{UAIBC}$ for realizing $\mathcal{F}_{com}$ in the $\bar{\mathcal{G}}_{acrs}$-Externalized UC setting

by corrupt parties during the commit phase, while simultaneously being capable of equivocating commitments that are sent by honest parties (for which the simulator must generate the protocol flows). On the surface, these properties seem to conflict – if a specific value can be extracted from a flow of the commitment protocol, then the flow must be binding, and yet, the simulator must also be able equivocate flows, so they are not always binding. This conflict is particularly striking in the Augmented CRS model, where the adversary is allowed to learn all the same trapdoor information as the simulator. When no trapdoor information is available exclusively to the simulator, it is difficult to imagine how commitments can be binding for the adversary, but remain equivocable for the simulator. In other words, it would seem that if the simulator has the power to extract committed values and to equivocate commitments, an equally powerful real-world adversary should also have these capabilities. Indeed, that difficulty is precisely what causes the UAHC protocol to fail in our setting. Fortunately, there is one crucial advantage that the simulator is allowed to have that enables us to overcome these conflicts: the simulator is allowed to choose the random coins used to generate the protocol flows of honest parties, whereas the real-world adversary has no such control (although it may learn the random coins upon later corruption of the honest party). The coin-flipping technique is a method for leveraging this advantage.

The coin-flipping protocol executed in Steps 1-4 of the UAIBC commitment protocol is designed so that the simulator can program the outcome of the coin-flip whenever party $P_c$ is corrupt. By using the $P_c$'s IBTC key, $SK_{P_c}$, to equivocate the commitment sent in Step 2, the simulator can arrange the public key $\rho$ to be any string of its choosing. Of course, the simulator will choose a public key $\rho$ for which it knows the corresponding secret key, $\rho^{-1}$. Thus, if party $P_c$ is corrupt, the simulator is able to decrypt the ciphertext sent in Step 5, extracting the committed value.

To equivocate commitments sent to a corrupt party $P_r$, the simulator uses the IBTC key belonging to $P_r$, $SK_{P_r}$, in order to generate an equivocable commitment to be sent in the final flow (Step 5). The ciphertext sent contains a legitimate opening to 0, but will later be passed off as a random ciphertext in the event that the simulator must open to 1. While such an equivocation could be detected by someone with knowledge of $\rho^{-1}$, in this instance, no one (not even the simulator) can have such knowledge, as the coin-flipping process will be fair.

Proving that the simulation described above is indeed indistinguishable from the real-world interaction requires the use of a rewinding technique. We stress that the UC simulation itself does not employ any rewinding (which is of critical importance, since a UC simulator may not rewind the environment). However, the use of a rewinding technique is required whenever we must "program" the coin-flipping results in the case where the adversary corrupts party $P_r$. In particular, this scenario is encountered only in the reduction to the security of the encryption scheme used for one of the simulation indistinguishability hybrid arguments. In order to show that the encryption can be broken if the hybrid experiments can be distinguished, we must arrange for the encryption key being attacked by the reduction to appear in the outcome of the coin-flipping (even when $P_r$ is corrupted).

In summary, protocol UAIBC introduces a technique for transforming trapdoor commitments (which are merely equivocable) to full-fledged (adaptive *and* forward secure!) UC commitments (which must be both equivocable and extractable) by using a simulatable coin-flipping protocol. In particular, the simulatable coin-flipping protocol makes use of both rewinding and trapdoors to achieve simulatability. (We employ only the trapdoor based simulation technique when constructing the UC simulator itself, while relying on the ability to rewind the coin-flipping protocol *only for the* *proof* of the simulation's indistinguishability.) Our general approach to constructing adaptively UC secure commitments from standard trapdoor commitments is of independent interest, and indeed, the problem was implicit in many prior works (*e.g.* [19, 17, 8]) which either resort to impractical techniques (such as per-session CRS) or are forced to settle for static (non-adaptive) security.

## 5.5 Security Proof for Protocol UAIBC

In this section, we will prove that protocol UAIBC is a GUC-secure realization of $\mathcal{F}_{com}$ in the $\bar{\mathcal{G}}_{acrs}$ shared hybrid model. Before proceeding with the details of the proof, we first state and prove a general lemma regarding coin-tossing protocols that is used in our proof of security. Our lemma is analogous to the Reset Lemma of [9], but unlike the Reset Lemma it aims to guarantee not only that certain events will occur after rewindings, but that the output distribution of the entire experiment remains computationally indistinguishable.

**Lemma 5.3** (Coin-tossing Lemma – due to Victor Shoup)**.** Consider a general "indistinguishability" task, $X$. The task is represented as a PPT algorithm that may take as input a system parameter $\Lambda$, a random reference parameter $\rho$, a bit string $x$, and challenge bit $b$. The system parameter $\Lambda$ is generated by some designated PPT parameter generation algorithm. The reference parameter $\rho$ must be drawn from an abelian group $\Phi$ (which possibly depends on the choice of $\Lambda$) with efficiently computable inverse and group operations (which we write multiplicatively). The uniform distribution on $\Phi$ must be efficiently samplable, and membership in $\Phi$ should be efficiently decidable. The task $X$ is said to be computationally indistinguishable if every PPT distinguisher $D$ has at most negligible advantage in the following attack game:

1. The challenger first generates a system parameter $\Lambda$ (in accordance with the parameter generation algorithm). The challenger then samples $\rho \xleftarrow{\$} \Phi$, and sends the pair $(\Lambda, \rho)$ to $D$.
2. $D$ computes an arbitrary value $x$, and sends $x$ to the challenger.
3. The challenger chooses $b \xleftarrow{\$} \{0,1\}$, computes $y \leftarrow X(\Lambda, \rho, x, b)$, and sends $y$ to $D$.
4. $D$ outputs a guess $\hat{b} \in \{0,1\}$.
5. $D$'s advantage is defined to be $|\Pr[b = \hat{b}] - 1/2|$.

Attack Game 6: Task Indistinguishability Attack Game

Suppose that, instead of selecting the reference parameter at random from $\Phi$, we generate the reference parameter using a simple "coin tossing" protocol that takes place between the challenger and the distinguisher. The coin tossing makes use of a string commitment Com that is computationally binding with respect to the distinguisher (this security property is formally described in Section 5.2). For the sake of generality, we allow both Com and $\Phi$ to be parameterized by $\Lambda$. We claim that the computational indistinguishability of $X$ holds, even though $\rho$ is now being chosen with some participation by the distinguisher itself.

More formally, for any computationally indistinguishable task $X$, the advantage of any PPT distinguisher $D'$ is at most negligible in the following attack game:

1. The challenger first generates a system parameter $\Lambda$ (in accordance with the parameter generation algorithm). The challenger then sends $\Lambda$ to $D'$.
2. $D'$ sends a commitment $\kappa_1$ to the challenger.
3. The challenger samples $\rho_2 \stackrel{\$}{\leftarrow} \Phi$, and sends $\rho_2$ to $D'$.
4. $D'$ computes an arbitrary value $x$, along with an opening $(d_1, \rho_1)$, and sends the triple $(d_1, \rho_1, x)$ to the challenger.
5. The challenger verifies that $\kappa_1 = \mathsf{Com}(d_1, \rho_1)$, and that $\rho_1 \in \Phi$. The challenger then computes $\rho \stackrel{\$}{\leftarrow} \rho_1 \cdot \rho_2$. Finally, the challenger chooses $b \stackrel{\$}{\leftarrow} \{0, 1\}$, computes $y \leftarrow X(\Lambda, \rho, x, b)$, and sends $y$ to $D'$. If the verification step fails, the challenger sends the error message $\perp$ to $D'$ instead.
6. $D'$ outputs a guess $\hat{b} \in \{0, 1\}$.
7. $D'$'s advantage is defined to be $|\Pr[b = \hat{b}] - 1/2|$.

Attack Game 7: Coin-tossing Attack Game (for general tasks)

To summarize, subject to some minor and natural technical qualifications, the Coin-tossing Lemma states that: *Experiments that are computationally indistinguishable when fed with a randomly sampled reference parameter will remain indistinguishable when the reference parameter is substituted by the output of a (three round) two-party coin-tossing protocol (where the second flow of the protocol is assumed to be generated honestly).*

*Proof.* Suppose that $D'$ is an efficient distinguisher for the game described in Attack Game 7 that succeeds with a non-negligible advantage $\alpha$. In particular, we have that $\alpha \geq 1/p$ for some polynomial $p$ (at infinitely many values of the security parameter). We show how to construct an efficient distinguisher $D$ that succeeds with non-negligible advantage in Attack Game 6, contradicting the computational indistinguishability of the task. The distinguisher $D$ operates as follows:

1. $D$ runs a copy of $D'$ internally, acting as the challenger for $D'$, and begins by passing along the value of $\Lambda$ it receives in Step 1 of Attack Game 6. The value $\rho$ that $D$ also receives in Step 1 is recorded for later use.
2. After receiving a commitment $\kappa_1$ from the internal copy of $D'$, $D$ samples $\rho_2^* \stackrel{\$}{\leftarrow} \Phi$ and sends it to $D'$.
3. If $D'$ does not respond with a valid opening of $\kappa_1$, then $D$ outputs 0 and halts. Otherwise, if $D'$ responds with a value $x^*$ and a valid opening of $\kappa_1$ to the value $\rho_1$, then $D$ proceeds as follows:

$\omega \xleftarrow{\$} \{1, \ldots, p\}$
for $i \leftarrow 1$ to $p$ do
  if $i = \omega$ // *Pray $\omega$ is the "right" guess*
    then $\rho_2 \leftarrow \rho \cdot \rho_1^{-1}$
    else $\rho_2 \xleftarrow{\$} \Phi$
  rewind $D'$ back to Step 3 of Attack Game 7 and send $\rho_2$ to $D'$
  if $D'$ responds with $x$ and a valid opening of $\kappa_1$ to the value $\rho_1'$, then
    if $i \neq \omega$ then output 0 and halt // *Prayers unanswered*
    if $\rho_1' \neq \rho_1$ then output 0 and halt // *Commitment broken!*
    $D$ passes $x$ to its challenger in Step 2 of Attack Game 6
    $D$ receives a response $y$ from its challenger, and forwards $y$ to $D'$
    when $D'$ outputs $\hat{b}$, $D$ outputs $\hat{b}$ and halts
output 0 and halt // *rewinding failed to produce second opening*

We claim that this $D$ will have advantage at least $1/4p^2$ at game Attack Game 6. To see this, will consider a series of modifications to Attack Game 7.

**Game $G_0$.** This game is modified version of Attack Game 7 that plays out identically unless the verification in Step 5 succeeds. In this case, instead of simply sending the value $y$, the challenger will rewind $D'$ to Step 3 and feed it with another (freshly sampled) value of $\rho_2$, as many times as it takes to get a second run with a successful opening of the commitment. That is, after rewinding, the game proceeds normally to Step 5, only now if the verification fails, we will rewind $D'$ to Step 2 again and repeat until the verification in Step 5 has succeeded for a second time, after which the game continues as in the original version.

What is the advantage of $D'$ in $G_0$? If the advantage of $D'$ was $\alpha$ in Attack Game 7, then the advantage of $D'$ at $G_0$ is also $\alpha$. To see why, let $C$ and $C'$ be random variables representing the value of $\rho_2$ chosen on the first and last runs (respectively), and let $R$ be the random variable representing all the random coins used in Step 1 and the random tape of $D'$ (note that the value of $R$ is unaffected by any number of rewindings). Let $\mathsf{Succ}(r, \rho_2)$ be a predicate denoting whether the verification in Step 5 succeeds when $R = r$ and $C = \rho_2$ (*i.e.* it indicates whether $D'$ gives a valid opening of the commitment with those outcomes). Together, $R$ and $C$ represent all the random coins used in Attack Game 7. We will show that, for any fixed choice of $R$, the distribution of $C'$ is the same as the distribution of $C$, and therefore distribution of the modified experiment is unchanged from that of the original. Fixing our choice of $R$ and expanding the distribution of $C'$ by total probability theorem yields:

$$\begin{aligned}
\Pr[C' = \rho_2 \mid R = r] \quad = \quad & \Pr[C' = \rho_2 \mid R = r, \mathsf{Succ}(r, C)] \cdot \Pr[\mathsf{Succ}(r, C)] \\
& + \Pr[C' = \rho_2 \mid R = r, \neg\mathsf{Succ}(r, C)] \cdot \Pr[\neg\mathsf{Succ}(r, C)] \\
= \quad & \Pr[C = \rho_2 \mid R = r, \mathsf{Succ}(r, C)] \cdot \Pr[\mathsf{Succ}(r, C)] \\
& + \Pr[C = \rho_2 \mid R = r, \neg\mathsf{Succ}(r, C)] \cdot \Pr[\neg\mathsf{Succ}(r, C)]
\end{aligned}$$

The equality of the first terms in the sums follows from simply observing that $C' = C$ in the event that $D'$ fails to open the commitment on the first run, since no rewinding occurs in that case (the first run is the last run). The equality of the second terms follows by observing that, conditioned on $\mathsf{Succ}(r, C)$, the variable $C'$ is uniformly distributed among the values of $\rho_2$ such that $\mathsf{Succ}(r, \rho_2)$ is true, since in this case we will "rewind" by feeding uniformly random choices of $\rho_2$ into the experiment until $\mathsf{Succ}(r, \rho_2)$ holds. (Of course, $C$ is also uniformly distributed among those same values when conditioned on $\mathsf{Succ}(r, C)$.) Applying total probability theorem again to the last line in the equation above, we obtain: $\Pr[C' = \rho_2 \mid R = r] = \Pr[C = \rho_2 \mid R = r]$. Thus, the distributions of the pair $(R, C)$ and the pair $(R, C')$ are identical. We conclude that the output distribution of the original experiment in Attack Game 7 and the output distribution in Game $G_0$ are the same.

**Game** $G_1$**.** This is a minor modification of Game $G_0$. In the event that rewindings occur, $G_1$ is identical to $G_0$ for the first $p$ rewindings. After $p$ rewindings have occurred, $G_1$ proceeds from Step 3 as in the original version of Attack Game 7, without making any further attempts to rewind. $G_1$ is otherwise identical to $G_0$.

To analyze what happens to $G_0$ if we bound the number of rewindings by $p$, we first calculate the expected number of rewindings in $G_0$. Let $N$ denote the number of rewindings that occur in a run of $G_0$. Using our previously established notation, we will analyze the $\mathrm{E}[N]$ by first fixing the choice of $R$. We can then compute the expected number of rewindings $\mathrm{E}[N \mid R = r]$ with a simple application of total probability theorem (combined with the linearity property of expectation):

$$
\begin{aligned}
\mathrm{E}[N \mid R = r] &= \quad \mathrm{E}[N \mid R = r, \mathsf{Succ}(r, C)] \cdot \Pr[\mathsf{Succ}(r, C)] \\
&\quad + \mathrm{E}[N \mid R = r, \neg\mathsf{Succ}(r, C)] \cdot \Pr[\neg\mathsf{Succ}(r, C)] \\
&= \quad (1/\Pr[\mathsf{Succ}(r, C)]) \cdot \Pr[\mathsf{Succ}(r, C)] \\
&\quad + 0 \cdot \Pr[\neg\mathsf{Succ}(r, C)] \\
&= 1 \qquad\qquad\qquad (\text{ for } \Pr[\mathsf{Succ}(r, C)] \neq 0 \text{ )}
\end{aligned}
$$

The second equality follows by observing that, for a fixed value of $R$, the probability that any of the rewindings succeeds is exactly the same as $\Pr[\mathsf{Succ}(r, C)]$ (the initial probability with which a rewinding occurs). Of course, conditioned on $\neg\mathsf{Succ}(r, C)$, there are no rewindings at all. In case $\Pr[\mathsf{Succ}(r, C)] = 0$, we will have that $\mathrm{E}[N \mid R = r] = 0$, since no rewindings can occur. For any fixed choice of $r$ we have shown that $\mathrm{E}[N \mid R = r] \leq 1$, and therefore, we may conclude that $\mathrm{E}[N] \leq 1$.

Now we are in a position to analyze the advantage of $D'$ in $G_1$. Since the expected number of rewinding attempts in $G_0$ is at most 1, the overall probability that there will be more than $p$ rewinding attempts is at most $1/p$, by Markov's inequality. Therefore, with all but probability $1/p$, the rewinding procedure of $G_1$ perfectly simulates $G_0$, wherein $D'$ has the same advantage $\alpha$ as in the original setting of Attack Game 7. In the event that $G_1$ differs from $G_0$, the distinguisher $D'$ learns no information about $b$, so the probability that $\hat{b} = b$ is 1/2 (by independence). Therefore, the overall advantage of $D'$ in Game $G_1$ is at least $\alpha - (1/2)(1/p) \geq 1/p - 1/2p = 1/2p$.

**Game** $G_2$**.** This is the same as Game $G_1$, except that whenever a second opening of the commitment is obtained, it is checked against the original opening. If the commitment has been opened in two different ways, the game is halted. Since opening the commitment in two different ways must occur with negligible probability (by the binding property of the commitment scheme), for sufficiently large choices of the security parameter, this abort happens with probability at most $1/4p$. Therefore, the advantage of $D'$ in $G_2$ differs from that of $G_1$ by at most $1/4p$, so we have that $D'$ has advantage at least $1/2p - 1/4p = 1/4p$ in Game $G_2$.

**Game $G_3$.** Finally, we modify Game $G_2$ to exactly match the procedure we described above for constructed $D$ from $D'$. Namely, $G_3$ differs from $G_2$ in that if we must rewind, we first select a uniformly random "guess" at one of the $p$ possible rewindings, and if the "guess" does not correspond to the rewinding on which the second opening of the commitment occurs, the game is halted (and the advantage is computed as if $D'$ had output 0). Since the "guess" for the rewinding instance is accurate with probability $1/p$, the experiment plays out as in Game $G_2$ with probability at least $1/p$. In the event that a halt occurs, we will have that $\hat{b} = b$ with probability 1/2 (again, by independence). Therefore, the overall advantage of $D'$ in Game $G_3$ will be at least $(1/p)(1/4p) = 1/4p^2$.

This concludes our analysis of the advantage of $D$ in our construction, showing that $D$ indeed has a non-negligible advantage if distinguisher $D'$ can succeed at Attack Game 7 with non-negligible advantage. □

*Proof of Theorem 4.2.* We now prove the security of protocol UAIBC, by showing that it is a GUC-secure realization of $\mathcal{F}_{com}$. We describe a technique for simulating the protocol flows of UAIBC in the ideal-world with $\mathcal{F}_{com}$, and give a proof that the simulation in the ideal-world setting is indistinguishable from a real-world execution of UAIBC, even if the adversary is allowed to corrupt the trusted party implementing $\bar{\mathcal{G}}_{acrs}$ at the end of the game. Recall that we need only prove that UAIBC will $\bar{\mathcal{G}}_{acrs}$-EUC-emulate $\mathcal{F}_{com}$, and thus we will be proving with respect to a $\bar{\mathcal{G}}_{acrs}$-externally constrained environment $\mathcal{Z}$. To model this, we may imagine that $\mathcal{Z}$ is provided with $SK_{P_a}$, for any corrupt party $P_a$, at the moment of corruption. (In reductions to the security of the IBTC, we make an oracle query to obtain the key $SK_{P_a}$, which, as can be seen below, will never correspond to the key under attack by the reduction.)

**Initialization -** All parties are initialized with a copy of the common reference string $PK$ published by $\bar{\mathcal{G}}_{acrs}^{\mathsf{Setup},\mathsf{Extract}}$ during its honestly executed global initialization step. That is, the simulation we describe below merely expects the initialization step to be completed prior to the start of the simulation (no special setup is required on behalf of the simulator itself).

**Simulating Communication with $\mathcal{Z}$ -** $\mathcal{S}$ simply forwards communications between $\mathcal{A}$ and $\mathcal{Z}$.

**Simulating the commit phase for honest parties (Strategy 1) -** Since we are in the secure channels model, the simulator need only send null messages of appropriate length for each of the messages passed in UAIBC. Nothing more need be done unless one of the parties is corrupted at some later time. If the sender is later corrupted, the simulator receives the sender's input and can construct internal state for the sender representing an honestly generated transcript. If the recipient is later corrupted (prior to the reveal phase, which would again provide the simulator with the correct input) then the simulator must follow the strategy we now describe for the case of a recipient corrupted prior to the sender.

**Simulating the commit and reveal phases for corrupt recipients (Strategy 2) -** If the recipient is corrupted at any point during the commit phase while the sender is still honest, the simulator conducts the following strategy. We note that, once again, we depend critically on the secure channels model to allow the simulator to choose the content of the prior transcript only *after* corrupting the recipient in the ideal model. Since only the recipient has been corrupted, the simulator must be able to complete the commit phase of the protocol on behalf of the sender without actually knowing the committed bit $b$ (which $\mathcal{F}_{com}$ will not disclose until the reveal phase is initiated). Thus, the simulator needs to be able equivocate the commitment so that it can be opened to the proper value $b$ in the subsequent reveal phase. Before generating any of the protocol flows, $\mathcal{S}$ issues a `retrieve` request to $\bar{\mathcal{G}}_{acrs}^{\mathsf{Setup},\mathsf{Extract}}$ on behalf of the corrupt $P_r$ in order to obtain $SK_{P_r}$. All flows will be honestly generated except the final message sent by $P_c$ in Step 5 of the commit phase of the protocol. $\mathcal{S}$ computes $(\hat{\kappa}, \alpha) \leftarrow \mathsf{ECom}(P_r; SK_{P_r})$ and $d^{(\hat{b})} \leftarrow \mathsf{Eqv}(P_r, SK_{P_r}, \hat{\kappa}, \alpha, \hat{b})$ for $\hat{b} = 0, 1$ and stores the results. Instead of the honest flow, $\mathcal{S}$ sends $(\mathtt{com}_4, sid, P_c, P_r, \hat{\kappa}, \hat{\psi})$ where $\hat{e}$ is the encryption of $d^{(0)}$ (computed exactly as though committing to $b = 0$ in Step 5).

When performing the reveal phase, $\mathcal{S}$ can simply open to $b = 0$ using the decommitment $d^{(0)}$ honestly (including opening $\hat{\psi}$ correctly). If $\mathcal{S}$ must open to $b = 1$ it can use the decommitment $d^{(1)}$, and by the pseudo-random property of the encryption scheme, it can "pretend" that $\hat{\psi}$ was chosen at random.

**Simulating the commit and reveal phases for corrupt senders (Strategy 3) -** If the sender is corrupted while the recipient is honest during the commit phase, the simulator will need to learn the committed bit $b$ from the sender in order to correctly provide the corresponding commit phase input to $\mathcal{F}_{com}$. Once again, we note that the simulator need not generate any actual protocol flows until the actual time of corruption, so the following strategy may be employed irrespective of the particular step in the protocol of the commit phase at which $P_c$ is corrupted. Upon the initial corruption of $P_c$, $\mathcal{S}$ will issue a `retrieve` request to $\bar{\mathcal{G}}_{acrs}^{\mathsf{Setup},\mathsf{Extract}}$ on $P_c$'s behalf in order to retrieve $SK_{P_c}$. The simulation strategy will replace the flows sent by $P_r$ in Steps 2 and 4 with specially "rigged" flows. $\mathcal{S}$ first generates and stores a valid key-pair for the dense encryption scheme $(\overline{\rho}, \overline{\rho^{-1}}) \leftarrow G(1^\lambda)$. For Step 2, $\mathcal{S}$ computes and stores $(\hat{\kappa_1}, \alpha) \leftarrow \mathsf{ECom}(P_c, SK_{P_c})$, sending $\hat{\kappa_1}$ in place of the honestly computed $\kappa_1$ (on behalf of the honest party $P_r$). In Step 4, $\mathcal{S}$ computes and stores $\hat{\rho_1} \leftarrow \overline{\rho} \cdot \rho_2^{-1}$ and $\hat{d_1} \leftarrow \mathsf{Eqv}(P_c, SK_{P_c}, \hat{\rho_1})$. $\mathcal{S}$ replaces the legitimate decommitment $d_1$ that would normally be sent in Step 4 with $\hat{d_1}$, and the result is that $\kappa_1 = \mathsf{Com}_{P_c}(\hat{d_1}, \hat{\rho_1})$. This procedure ensures that $\mathcal{S}$ knows the decryption key for the ciphertext $\psi$ sent by $P_c$ in Step 5, and can thus decrypt $\psi$ if $b = 0$ and it was legitimately formed. If $\mathcal{S}$ is able to decrypt $\psi$ to obtain a valid $d$ such that $\kappa = \mathsf{Com}_{P_r}(d, 0)$, then $\mathcal{S}$ proceeds by invoking $\mathcal{F}_{com}$ as if $P_c$ had committed to 0, otherwise it invokes it with a commitment to 1 (since $P_c$ cannot possibly produce a valid opening of the commitment to 0 in this case).

Notice that in this scenario, the simulator will not need to send any message during the reveal phase, since the sender is corrupted and is the only party sending messages during a reveal operation. The simulation will only fail (and halt) if a valid reveal phase message is sent from $P_c$ (who is corrupt, and controlled by $\mathcal{A}$) which opens to a different bit $b$ than the simulator chose when invoking $\mathcal{F}_{com}$ at the end of the commit phase.

**Message delivery -** $\mathcal{S}$ will merely deliver messages from $\mathcal{F}_{com}$ whenever $\mathcal{A}$ delivers the corresponding protocol messages in the simulated $\mathsf{UAIBC}$ interaction, in the obvious manner.

Given the above simulation strategies, we must prove that the environment $\mathcal{Z}$'s interactions with $\mathcal{S}$ will be indistinguishable from $\mathcal{Z}$'s interactions with $\mathcal{A}$ in the real-world, even if $\mathcal{A}$ is allowed to corrupt the trusted party $\mathsf{T}^{\bar{\mathcal{G}}_{acrs}}$ at the end of the experiment. To show this, we consider the following hybrid experiments.

$I_0$ **- Real-world interaction.** Protocol $\mathsf{UAIBC}$ running in the "real-world" (in this case, the $\bar{\mathcal{G}}_{acrs}$-Relativized setting), interacting with the real-world adversary $\mathcal{A}$.

$I_1$ **- Employing the commitment part of Strategy 2.** The interaction proceeds exactly as in the real-world, except that wherever Step 5 of the protocol is executed with a corrupt receiver $P_r$, the honest sender $P_c$ computes $(\hat{\kappa}, \alpha) \leftarrow \mathsf{ECom}(P_r, SK_{P_r})$ (for now, we will simply imagine that that $P_c$ is somehow provided with $SK_{P_r}$). $P_c$ then computes $d \leftarrow \mathsf{Eqv}(P_r, SK_{P_r}, \hat{\kappa}, \alpha, b)$. The remainder of the protocol is then conducted as usual. In the event that $P_c$ is later corrupted, its internal state is constructed to be consistent with a random selection process for the same value of $d$.

**Lemma 5.4.** *$I_1$ is computationally indistinguishable from $I_0$.*

*Proof.* The proof is by reduction to the equivocability property of $IBTC$s, since the only difference from $I_0$ is the use of an equivocable commitment. We observe that if there is an adversary $\mathcal{A}$ such that $\mathcal{Z}$ can distinguish interactions in $I_0$ from $I_1$, there is a (simple) reduction which breaks the equivocability property of the $IBTC$. This indistinguishability holds even for adversaries in the strong fully simulatable setting, since the reduction is given $MSK$ (containing the random coins used by $\mathsf{Setup}$) as part of its input. Thus, it is easy for the reduction to properly simulate even the corruption of the trusted party. We omit the trivial details of the reduction. □

$I_2$ **- Employing only Strategy 2.** This interaction is the same as $I_1$, only $P_c$ will perform Step 5 completely as described in Strategy 1. That is, in addition to the change made in $I_1$, the only difference between $I_2$ and $I_1$ is the computation of the value of $\psi$. In the event that the honest party commits a bit $b = 1$, the value of $\psi$ now contains $E_\rho(r, d^{(0)})$, whereas it would have been random in $I_1$.

**Lemma 5.5.** *$I_2$ is computationally indistinguishable from $I_1$.*

*Proof.* The proof is by reduction to the pseudo-random property of ciphertexts from $E$. Unfortunately, it is not obvious how to build a reduction attacking the security of a particular public key $\rho^*$, as we must do in order to achieve the reduction. In fact, we must use a "rewinding" technique in order to accomplish this. While it is critical that UC secure protocols be straight-line *simulatable* (*i.e.* they may not use rewinding techniques for simulation purposes, see [32] for details), we stress that here we are using the rewinding only in the proof of indistinguishability for the simulator. The simulator itself does not perform any rewinding.

Therefore, our reduction makes use of Lemma 5.3 (the Coin-tossing Lemma) to abstract away the complicated analysis of the rewinding procedure. Observe that the first three flows of

$\mathsf{UAIBC}$ are a simple Coin-tossing protocol, satisfying all the conditions of Lemma 5.3, provided that the GUC experiment we are running does not require access to $P_c$'s trapdoor (which is relevant for the binding property of the commitment used in the coin-tossing phase) until after the coin-tossing has completed. Since $P_c$ is an honest party in this scenario, we can be sure that the neither $\mathcal{A}$ nor $\mathcal{Z}$ (nor, of course, the simulator itself) will expect access to $P_c$'s trapdoor, and the conditions of the Coin-tossing Lemma are thus satisfied. Accordingly, we may substitute the coin-flipped public key $\rho$ with any randomly generated public key (as if the coin-flipping phase of the protocol had never happened, in either the real or ideal/simulated views). In particular, we may substitute a challenge key $\rho^*$ that our reduction to the security of $E$ is attempting to break. If the resulting versions of $I_2$ and $I_1$ are indistinguishable (when using such a random challenge key $\rho^*$), then we know that the unmodified versions of $I_2$ and $I_1$ (where $\rho$ is the outcome of the coin-flipping protocol) are also indistinguishable.

In fact, it is easy to see that $I_2$ and $I_1$ must indeed be indistinguishable if the public key $\rho^*$ (challenged to the reduction) is used instead of the coin-flipping protocol output – after all, the reduction to the security of $E$ is now a direct one: the only change between $I_2$ and $I_1$ is the substitution of a ciphertext encrypted under $\rho^*$ for a random string. Indistinguishability of this substitution is precisely the guarantee provided by Dense PRC encryption. Given that $I_2$ and $I_1$ are indistinguishable when the parties are fed with a random key $\rho^*$, we may apply the Coin-tossing Lemma to obtain indistinguishability of the full-fledged versions of $I_2$ and $I_1$ (which use the coin-tossing protocol).

Once again, we observe that providing the adversary with $MSK$ at the end of the game does not help the adversary to distinguish ciphertexts, since it is too late to use it to equivocate any of the commitments, and thus the adversary cannot "rig" any public keys. □

$I_3$ **- Adding Strategy 3.** This interaction is the same as $I_2$ only now $P_r$ also employs Strategy 2. We first note that it is not a problem that the simulator is revealing commitments to $\hat{\kappa_1}$ instead of $\kappa_1$ since the distributions are indistinguishable by the "dense" public key property of the encryption scheme. The only remaining difference is the use of an equivocable commitment, and thus we prove indistinguishability of $I_3$ and $I_2$ using the same technique as Lemma 5.4.

$I_4$ **- Adding Strategy 1.** This interaction is the same as $I_3$, only we now employ Strategy 1 as well. This is, in fact, just a conceptual change to $I_3$, since Strategy 1 merely delays the actual computation of protocol flows until they are observable by $\mathcal{A}$. Indistinguishability follows immediately in the secure channels model.

$I_5$ **- Simulated interaction.** This is the same as $I_5$ unless the simulation fails and halts, which can only occur if a reveal phase successfully opens a commit phase to for the bit $b' \neq b$, where $b$ is the bit the simulation invoked $\mathcal{F}_{com}$ with after the commit phase. This can occur in one of two ways. 1) The simulator was unable to decrypt $\psi$ to yield a valid $d$ in Step 5, yet the reveal phase shows that $\psi$ contains a correct encryption of a valid $d$. This would violate correctness property of the encryption scheme, so this case never occurs. 2) The simulator decrypted $\psi$, yielding a valid $d$ such that $\kappa = \mathsf{Com}_{P_r}(d, 0)$, but the reveal phase showed a decommitment $d'$ such that $\kappa = \mathsf{Com}_{P_r}(d', 1)$. This breaks the binding property of the $IBTC$, and a reduction to this property can be built in the obvious way (we observe that the binding property is broken at "runtime", so the reduction will not need to simulate the corruption of the trusted party in the strong fully simulatable setting, which is critical since it will not know $MSK$). Thus $I_5$ must be indistinguishable from $I_4$.

□

## 6 Acknowledgments

The authors would like to thank Philip MacKenzie, Johan Håstad, and Silvio Micali for insightful observations, as well as various anonymous referees for helpful comments and suggestions. The authors would also like to thank Victor Shoup for discovering a subtle error in our original rewinding proof, and proposing the Coin-tossing Lemma to overcome it.

## References

[1] M. Abe, and S. Fehr. Perfect NIZK with Adaptive Soundness. To appear in *Proc. of TCC*, 2007.

[2] G. Ateniese and B. de Medeiros. Identity-based Chameleon Hash and Applications. *Proc. of Financial Cryptography*, 2004. Available at `http://eprint.iacr.org/2003/167/`.

[3] D. Beaver. Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. in *J. Cryptology*, vol 4., pp. 75–122, 1991.

[4] M. Blum. How to prove a theorem so no one else can claim it. In *Proc. of the International Congress of Mathematicians*, 1986.

[5] B. Barak, R. Canetti, Y. Lindell, R. Pass and T. Rabin. Secure Computation Without Authentication. In *CRYPTO 2005*, Springer-Verlag (LNCS 3621), pages 361-377, 2005.

[6] B. Barak, R. Canetti, J. Nielsen and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *Proc. of FOCS*, 2004.

[7] D. Boneh, and M. Franklin. Identity Based Encryption from the Weil Pairing. In *Proc. of Crypto*, 2001.

[8] B. Barak and Y. Lindell. Strict Polynomial-time Simulation and Extraction. In *SIAM J. Comput.*, 33(4), pp. 783-818, 2004.

[9] M. Bellare and A. Palacio. GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In *Proc. of Crypto*, Springer-Verlag (LNCS 2442), 2002.

[10] B. Barak and A. Sahai, How To Play Almost Any Mental Game Over the Net - Concurrent Composition via Super-Polynomial Simulation. In *Proc. of FOCS*, 2005.

[11] R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology,* Vol. 13, No. 1, winter 2000.

[12] R. Canetti. Universally Composable Security: A New paradigm for Cryptographic Protocols. In *Proc. of FOCS*, pages 136–145, 2001.

[13] R. Canetti. Universally Composable Security: A New paradigm for Cryptographic Protocols. Cryptology ePrint Archive, Report 2000/067, **revised edition from Dec. 2005**. Available at: `http://eprint.iacr.org/2000/067`

[14] R. Canetti. Universally Composable Signature, Certification, and Authentication. In *Proc. of CSFW*, p. 219, 2004.

[15] Ronald Cramer, Ivan Damgard, Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Proc. of CRYPTO*, pp. 174–187, 1994.

[16] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Conference on Security in Communication Networks (SCN)*, 2002.

[17] R. Canetti and M. Fischlin. Universally Composable Commitments. In *Proc. of Crypto*, pages 19–40, 2001.

[18] R. Canetti, E. Kushilevitz and Y. Lindell. On the Limitations of Universally Composable Two-Party Computation Without Set-Up Assumptions. In *Proc. of Eurocrypt*, Springer-Verlag (LNCS 2656), pp. 68–86, 2003.

[19] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Secure Computation. In *Proc. of STOC*, pp. 494–503, 2002.

[20] R. Canetti and T. Rabin. Universal Composition with Joint State. In *Proc. of Crypto 2003*, Springer-Verlag, pp. 265-281, 2003.

[21] I. Damgard and J. Nielsen. Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor. In *Proc. of Crypto*, Springer-Verlag, pp. 581–596, 2002.

[22] Y. Dodis and S. Micali. Parallel Reducibility for Information-Theoretically Secure Computation. In *Proc. of Crypto*, Springer-Verlag (LNCS 1880), pp. 74–92, 2000.

[23] U. Feige. Alternative Models for Zero Knowledge Interactive Proofs. Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel, 1990.

[24] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Based on a Single Random String. In *Proc. of FOCS*, 1990.

[25] U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In G. Brassard, editor, *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 526–545. Springer-Verlag, 1990, 20–24 Aug. 1989.

[26] S. Goldwasser, and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. *CRYPTO '90, LNCS 537,* 1990.

[27] S. Goldwasser, S. Micali, C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computation*, 18(1):186–208, 1989.

[28] O. Goldreich, S. Micali, and A. Wigderson. How to Solve any Protocol Problem. In *Proc.of STOC*, 1987.

[29] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.

[30] D. Hofheinz, J. Muller-Quade, and D. Unruh. Universally Composable Zero-Knowledge Arguments and Commitments from Signature Cards. In *Proc. of the 5th Central European Conference on Cryptology MoraviaCrypt 2005*, June 2005.

[31] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated Verifier Proofs and their Applications. In *Proc. of Eurocrypt*, Springer-Verlag, 1996.

[32] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. In *Proc. of FOCS*, Springer-Verlag, pp. 394–403, 2003.

[33] S. Micali and P. Rogaway. Secure Computation. unpublished manuscript, 1992. Preliminary version in *CRYPTO '91, LNCS 576,* 1991.

[34] P. MacKenzie and K. Yang. On Simulation-Sound Trapdoor Commitments. In *Proc. of Eurocrypt*, Springer-Verlag (LNCS 3027), pp. 382–400, 2004.

[35] R. Pass. On Deniabililty in the Common Reference String and Random Oracle Model. In *Proc. of Crypto*, LNCS 2729, pp. 216–337, 2003.

[36] R. Pass. Bounded-Concurrent Secure Multi-Party Computation with a Dishonest Majority. In *Proc. of STOC*, pp. 232–241, 2004.

[37] M. Prabhakaran and A. Sahai. New Notions of Security: Achieving Universal Composability without Trusted Setup. In *Proc. of STOC*, 2004.

[38] B. Pfitzmann and M. Waidner. Composition and Integrity Preservation of Secure Reactive Systems. In *Proc. of ACM CCS*, pages 245–254, 2000.

[39] F. Zhang, R. Safavi-Naini and W. Susilo. ID-Based Chameleon Hashes from Bilinear Pairings. Available at `http://eprint.iacr.org/2003/208/`.