# Universally Composable Protocols with Relaxed Set-up Assumptions


Boaz Barak*
Institute for Advanced Study
Princeton, NJ, USA
boaz@ias.edu

Ran Canetti
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA
canetti@watson.ibm.com

Jesper Buus Nielsen
ETH Zürich
Zürich, Switzerland
jnielsen@inf.ethz.ch

Rafael Pass†
MIT
Cambridge, MA, USA
pass@csail.mit.edu



## Abstract

*A desirable goal for cryptographic protocols is to guarantee security when the protocol is composed with other protocol instances.* Universally Composable (UC) *protocols provide this guarantee in a strong sense: A protocol remains secure even when composed concurrently with an unbounded number of instances of arbitrary protocols. However, UC protocols for carrying out general tasks are known to exist only if a majority of the participants are honest, or in the* common reference string (CRS) model *where all parties are assumed to have access to a common string that is drawn from some pre-defined distribution. Furthermore, carrying out many interesting tasks without honest majority is* impossible *in the plain model, even if ideally authenticated communication is provided.*

*We propose alternative set-up assumptions and show that they suffice for reproducing the general feasibility results for UC protocols in the CRS model. These alternative assumptions have the flavor of a "public-key infrastructure": parties have registered public keys, no single registration authority needs to be fully trusted, and no single piece of information has to be globally trusted and available. In addition, unlike known protocols in the CRS model, the proposed protocols guarantee some basic level of security even if the set-up assumption is violated.*


## 1. Introduction

Designing protocols that guarantee security in open, multi-protocol, multi-party execution environments is a challenging task. In such environments a protocol instance is executed concurrently with an unknown number of instances of the protocol, as well as arbitrary other protocols. Indeed, it has been demonstrated time and again that adversarially-coordinated interactions between different protocol instances can compromise the security of protocols that were demonstrated to be secure when run in isolation (see, e.g., [GK90, DDN00, DNS98, KSW97]). A natural way for guaranteeing security of protocols in such complex execution environments is to require that protocols satisfy a notion of security that provides a general *secure composability* guarantee. That is, it should be guaranteed that a secure protocol maintains its security even when composed with (i.e., runs alongside) arbitrary other protocols. Such a general notion of security is provided by the universally composable (UC) security framework [C01], which provides a very general composability property: A secure protocol is guaranteed to maintain its security (in the sense of emulating an ideally trusted and secure service) even when run concurrently with multiple copies of itself, plus arbitrary network activity.

However, such strong secure composability properties come at a price: Carrying out many interesting cryptographic tasks in the UC framework has been shown to be *impossible* in the plain model, unless a majority of the participants are completely honest [C01, CF01, CKL03]. The impossibility holds even if ideally authenticated communication is guaranteed. Furthermore, it has been shown [L03, L04] that this impossibility is not a result of technical characteristics of any particular framework; rather, they are inherent to the strong composability requirements.

In light of these results, researchers have turned to constructing protocols under some setup assumptions. Specifically, the *common reference string* (CRS) model was used.

* Work done while visiting the IBM T.J. Watson Research Center.
† Work done while at the Royal Institute of Technology, Sweden

In this model, originally proposed in [BFM88], all parties have access to a common string $r$ that was ideally drawn from some publicly known distribution. It is further assumed that no secret information correlated with this string is known. (It can be thought of as if this string is published by some "trusted dealer" that plays no further part in the protocol execution). Several strong feasibility results are known in the CRS model. In particular, it was shown that practically any functionality can be realized by a UC protocol in the CRS model, with any number of faults, assuming authenticated communication [CF01, CLOS02]. Furthermore, non-interactive UC Zero-Knowledge protocols were constructed in the CRS model model under similar assumptions (and assuming secure data erasure is possible) [DDO$^+$01, CLOS02].

**Drawbacks of the CRS model.** Security in the CRS model depends in a crucial way on the fact that the common string is chosen properly. That is, the model provides no security guarantees in the case that the common string is chosen from a different distribution than the specified one, or when some "secret trapdoor information" correlated with the reference string is leaked. In fact, in most existing protocols in the CRS model the security proof actually provides an efficient strategy for *completely breaking* the security of the protocol when the reference string can be chosen in a malicious way. In particular, if the CRS is chosen by a single entity then this entity could, if it wished, read sensitive data, forge proofs, and in general completely undermine the security of the protocol, all in a way that is undetectable by the honest parties. This means that the naive way of realizing the CRS model by letting a single entity choose the reference string forces all the participants to put absolute trust in this entity.

**Our approach.** We reproduce the above feasibility results, proven in the CRS model, in a number of alternative models (or, "set-up assumptions"). The main advantage of these alternative models is that they are realizable in ways that reduce the trust that the participants need to put in other entities. All these models have a common theme: They avoid the need that all participants in a protocol execution put complete trust in a single entity, or a single reference string. Instead, we adopt a trust model that somewhat resembles the trust model of a "public-key infrastructure." That is, each party registers a "public key" with a "registration authority" that it trusts. In order to engage in a joint computation, the participants obtain the public keys of each other from the respective authorities. The trust that each participant has to put in other authorities, except for the authority it is registered with, is quite minimal; thus no single entity needs to be completely trusted by all participants. Our protocols do not require parties to keep any secret data that is associated with their public keys. Furthermore, they are naturally aligned with the standard trust model of a public-key infrastructure, that is anyhow needed for obtaining authenticated communication. They also provides better "plausible deniability" properties.

**Our results in more detail.** We present our set-up assumptions as follows. We first formulate a "key registration (KR) service" that captures the "minimal common denominator" of several set-up scenarios. That is, we demonstrate how this service can be realized in each one of these scenarios. Then, we show how to reproduce the above results given this service. All our results hold in face of any number of adaptively corrupted parties. (Technically, the KR service is captured as an "ideal functionality" within the UC framework. See details within.)

The key registration service is parameterized by a function $f : \{0,1\}^* \to \{0,1\}^*$ (representing a method for computing the public key from a seed, that represents the secret key), and provides roughly the following "ideal services". Any party can register with the KR service and obtain a public key in return. In a "normal" registration process, the registering party obtains a public key $v = f(r)$ for some uniformly chosen $r$ that is known only to the service. (Such keys are called $\mathsf{safe}$.) In addition, a corrupted party may provide the service with any arbitrary $r$ and have its public key set to $f(r)$. (Such keys are called $\mathsf{well\text{-}formed}$.) Furthermore, uniqueness of keys is not guaranteed: the public key of any party may be set as equal to the public key of any other party, subject to the sole restriction that honest parties are assigned safe keys. Parties may also have multiple keys. Whenever a party asks the service for the public key of another party, the service returns one of the keys registered for that party.

This formulation of the KR service provides relatively weak security guaranteed to users. It captures either a single entity that provides keys to all parties (such as in the case of a CRS), or a collection of separate entities, where no single entity is trusted by all parties. The difference between the powers of honest and corrupted registering parties, and the ability to copy keys, models the fact that a party may put more trust in the entity it registers with than in the entities chosen by other parties. Specifically, a party can trust only its own key to be safe. Other keys may be only well-formed.

We show how to realize any ideal functionality given a KR service with an appropriate choice of the derivation function $f$. Our constructions are natural adaptations of the [CF01, CLOS02, DN02] constructions and rely on the same cryptographic assumptions. We also show how to construct *non-interactive* UC Zero-Knowledge protocols given a KR service with an appropriate $f$. Here our construction is quite different than existing ones, as no natural extension of the existing constructions seems to work.

We show that the KR service can be easily realized in a number of natural settings (or, trust models). A first such model is the CRS model itself (thus demonstrating that the KR service model is indeed a relaxation of the CRS model). Other trust models include a number of “PKI-like” settings where the parties have access to registration authorities that either choose public keys for the registering parties, or alternatively expect the registering party to exhibit the secret seed that corresponds to the registered public key. These models constitute a sequence where the trust in the registration authorities becomes weaker and weaker. See details in Section 3.

Our trust assumptions should be contrasted with the more standard trust assumptions on registration authorities for the purpose of providing authenticated communication. (These assumptions are formalized within the UC framework in [C04], where it is also shown that without trust assumptions no authenticated communication is possible.) Essentially, for the purpose of providing authenticated communication, the registration authority should simply register the registrant’s identity together with a public value that is provided by the registrant, and supply the registered public key of any party upon request. This is a strictly weaker trust assumption than that the KR service. However, for the purpose of guaranteeing authentication it is essential that the keys of each two honest parties are different from each other, and furthermore that each party maintains a secret key associated with its public key. In contrast, in our case the keys of honest parties can be identical and no secret information related to those keys needs to be known.

Finally, we consider the case where parties have no trust whatsoever in the registration services used by other parties. (This setting can be thought of as the setting where each party runs its own registration service which registers only itself.) Here we’re back in the plain model of computation, thus UC computation is in general impossible. Yet, our protocols still provide some security guarantees. Specifically, we demonstrate that they remain secure with respect to the standard notion of stand-alone security (as in, say, [C00]).

**Our Techniques.** We use two different types of techniques: one for realizing general functionalities (i.e., re-establishing the [CLOS02] results), and another for constructing non-interactive UC Zero-Knowledge protocols. We sketch them in order. Recall that the [CLOS02] general construction for realizing any ideal functionality proceeds as follows. First, a *UC Commitment* protocol, i.e. a protocol that realizes the ideal commitment functionality, is constructed. Next, any functionality is realized given ideal commitment. It thus suffices for our purpose to demonstrate how to realize the ideal commitment functionality in our model. Recall that the ideal commitment functionality comes in two flavors: $\mathcal{F}_{com}$, which handles a single commitment, and $\mathcal{F}_{mcom}$, which handles multiple commitments and is realizable using a protocol that handle multiple commitments using a single copy of the CRS. We concentrate on the more challenging task of realizing $\mathcal{F}_{mcom}$; this allows us to run multiple commitments (and, consequently, multipel copies of our protocols for realizing any functionality) using a single public key per party.

Our starting point is an observation that existing protocols for realizing $\mathcal{F}_{mcom}$ in the CRS model actually use the CRS for two separate purposes, called *extractability* and *equivocation.* (A sketch of these properties appears within.) Furthermore, the CRS consists of two separate parts, where each part is used to guarantee only one of the properties. Finally, extractability is a concern only when the committer is corrupted, and equivocation is a concern only when the receiver is corrupted. It thus seems natural to “split” the CRS between the public keys of the committer and the receiver, where each party holds the corresponding part. That is, the public key of each party will consist of a part guaranteeing extractability for the commitments where it plays the committer, plus a part guaranteeing equivocation for the commitments where it plays the receiver. We show that this approach works, modulo some technical complications. In another “twist,” taken from [DN02], we show how to realize $\mathcal{F}_{mcom}$ in a setting where only the committer has a registered public key.

Next we sketch our approach for constructing non-interactive UC Zero-Knowledge protocols. Here the exiting protocols are not so naturally amenable to separating the CRS. Specifically, all known constructions use the so-called “hidden bit model” (see, e.g., [GOL01]), where the same string is used to guarantee the concerns of both parties, i.e. extractability and simulatability. We thus propose a new construction, that allows for such separation: Essentially, the parties will run the ZAP protocol of [DN00], where the verifier’s challenge is included in its public key. In addition, the prover’s public key contains information that allows for extractability of the witness. The verifier’s public key will contain additional information that guarantees simulatability.

**Obtaining plausible deniability.** An intriguing property of traditional zero-knowledge protocols in the plain model is that the interaction is “deniable” for the prover, in the sense that the verifier cannot later “convince” a third party, who did not witness the interaction, that the interaction took place. Essentially, this is so since the verifier can, using the simulator guaranteed by the zero-knowledge property, generate a valid-looking transcript of an interaction even when the verifier never actually interacted with the prover. The crucial property here is that there exists a simulator that needs only information that is locally known to the verifier. We call this property self simulatability, and generalize it in a natural way to two-party protocols for any task.[1]

1 We use a different term since “deniability” means different things in

Interestingly, zero-knowledge and other secure two-party protocols in the CRS model are not necessarily self-simulatable, since that model allows the simulator to use "trapdoor information" on the CRS that is not available to the parties in a real execution. Furthermore, none of the known zero-knowledge and general two party protocols in the UC framework is self-simulatable.

Our constructions, both of UC non-interactive zero-knowledge and of protocols for realizing any two-party functionality, are self-simulatable for some of the instantiations of the KR service. (Specifically, we need an instantiation where each party explicitly provides its secret seed to the registration service.) This may be regarded as an additional advantage of our protocols over existing ones. We note that the way we achieve self-simulatability is reminiscent of the technique of Impagliazzo and Jakobsson [JSI96].

**Related Work.** Prabhakaran and Sahai [PS04] have recently proposed a way to relax the UC framework so as to allow general secure computation in the plain model while maintaining the ability to prove the universal composition theorem. This approach is complementary to ours: While we investigate how far can one relax the set-up assumptions within the UC framework, they investigate how far can one relax the framework itself, while still guaranteeing some sort of security and composability in certain cases. In particular, while they make no set-up assumptions, the security guarantees provided by their notion are strictly weaker than the ones provided here.

Herzog, Liskov and Micali [HLM03] use an enhanced public-key model that has some similarities to our model, in order to obtain plaintext-aware encryption.

**Organization.** Section 2 contains some brief background on the UC framework. Section 3 presents the key registration functionality $\mathcal{F}_{kr}$ and discusses how it can be realized. Section 4 shows how to realize any well-formed functionality in the $\mathcal{F}_{kr}$-hybrid model. Section 5 sketches our construction of UC non-interactive ZK protocols, and Section 6 sketches our argument that the protocols we construct remain stand-alone secure even if all the trust assumptions fail. Throughout, the presentation in the body of the paper is quite informal. More rigorous treatment is provided in [BCNP04] .

## 2. Preliminaries

We provide a brief overview of the universally composable security framework of [C01]. The framework allows for defining the security properties of cryptographic tasks so that the security of protocols is maintained under a general composition operation with an unbounded number of instances of arbitrary protocols running concurrently in the system. This composition operation is called universal composition. Similarly, definitions of security in this framework are called universally composable (UC).

As in other general definitions (e.g., [MR91, B91, C00, PSW00]), we use the definitional approach of [GMW87], where a protocol is said to securely realize a given task if running the protocol amounts to "emulating" an ideal process where the parties and the adversary hand their inputs to a trusted party that locally evaluates the appropriate outputs and hands them back to the parties. The algorithm run by the trusted party (which is aimed at capturing the requirements of the task at hand) is called an ideal functionality. This algorithm may simply evaluate a function of the inputs of the parties, or alternatively be an ongoing *reactive* process where inputs and outputs occur over time and local state is maintained.

The model of computation includes the parties running the protocol, an adversary $\mathcal{A}$ that controls the communication channels and potentially corrupts parties, and an *environment* $\mathcal{Z}$ that generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. (This interaction models the inevitable "information flow" between a protocol execution and the rest of the system, including other protocols running concurrently.) A protocol "emulates" the ideal process with a given ideal functionality $\mathcal{F}$ if for any "real-life" adversary $\mathcal{A}$ that interacts with the protocol and the environment there exists an "ideal-process adversary" $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that interact with $\mathcal{F}$ in the ideal process. In a sense, here $\mathcal{Z}$ serves as an "interactive distinguisher" between a run of the protocol and the ideal process with access to $\mathcal{F}$.

In addition to serving as a security criterion for protocols, the concept of a "trusted party" is used also to capture semi-idealized computation, and in particular set-up assumptions. Specifically, the $\mathcal{F}$-hybrid model is defined as the model where the parties have, in addition to the usual communication mechanisms, also access to multiple copies of a trusted party running ideal functionality $\mathcal{F}$. The copies of $\mathcal{F}$ are identified via *session IDs (SIDs).* That is, each call to a copy of $\mathcal{F}$ and each response from this copy should hold the SID of that copy.

The following *universal composition theorem* is proven in [C01]. Consider a protocol $\pi$ that operates in the $\mathcal{F}$-hybrid model, and let $\rho$ be a protocol that securely realizes $\mathcal{F}$ as sketched above. Let the composed protocol $\pi^\rho$ be identical to $\pi$ with the exception that the interaction with *each copy* of $\mathcal{F}$ is replaced with an interaction with a *separate*

different contexts. For instance, in the context of encryption and voting, deniability means the ability of the sender or a data (or voter) to equivocate its local data and randomness even when the true communication transcript is known (see e.g. [1, CDNO97]).

*instance* of $\rho$. Then, $\pi$ and $\pi^\rho$ have essentially the same input/output behavior. In particular, if $\pi$ securely realizes some ideal functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model then $\pi^\rho$ securely realizes $\mathcal{I}$ as well, without access to $\mathcal{F}$.

**The CRS model.** In the UC framework, the CRS model is formalized as the $\mathcal{F}_{crs}$-hybrid model, where $\mathcal{F}_{crs}$ is the common reference string ideal functionality, presented in Figure 1. A call to $\mathcal{F}_{crs}$ contains the SID of that intance, $sid$. Here all calls are answered by the same reference string that is chosen by the functionality according to a publicly known distribution.

---

**Functionality $\mathcal{F}_{crs}^D$**

$\mathcal{F}_{crs}^D$ is parameterized by distribution $D$. It proceeds as follows, running with a set of parties and an adversary $\mathcal{S}$:

1. Choose a value $r \stackrel{\mathrm{R}}{\leftarrow} D$.
2. When receiving ($\texttt{CRS}$,$sid$) from some party send ($\texttt{CRS}$,$sid, r$) to that party.

**Figure 1. The Common Reference String functionality**

---

## 3. The key registration functionality, $\mathcal{F}_{kr}$

This section presents and motivates our relaxed key registration functionality. We also present several alternative ways to realize it. The idea is to provide a relatively general and minimal set-up assumption that can be realized by a number of quite different and alternative "set-up mechanisms", and at the same time suffices for realizing general functionalities. We first present the functionality. Next, we describe a number of ways to realize it.

The key registration functionality, $\mathcal{F}_{kr}$, is presented in Figure 2. It is parameterized by a (deterministic) function $f : \{0,1\}^* \to \{0,1\}^*$, that represents a method for computing a public key given a secret (and supposedly random) key. The functionality allows parties to register their identities together with an associated "public key". The "public key" to be associated with a party upon registration is determined as follows. The functionality keeps a (public) set $R$ of "good public keys". Upon receiving a registration request from party $P_i$ (which is either corrupted or uncorrupted), the functionality first notifies the adversary that a request was made and gives the adversary the option to set the registered key to some key $p$ that is already in $R$. If the adversary declines to set the registered key, then the functionality determines the key on its own, by choosing a random secret $r$ from a given domain (say, $\{0,1\}^k$ for a security parameter $k$) and letting $p = f(r)$. Once the registered key $p$ is chosen, the functionality records $(P_i, p)$ and returns $p$ to $P_i$ and to the adversary. Finally, if $p$ was chosen by the functionality itself then $p$ is added to $R$. If the registering party is corrupted, then it can also specify, if it chooses, an arbitrary "secret key" $r$ and then register with the value $f(r)$. A retrieval request (made by any party) for the public key of $P_i$ is answered with either an error message $\perp$ or one of the registered public keys of $P_i$, where the adversary chooses which registered public key, if any, is returned. (That is, the adversary can prevent a party from retrieving any of the registered keys of another party.)

---

**Functionality $\mathcal{F}_{kr}^f$**

$\mathcal{F}_{kr}^f$ proceeds as follows, given function $f$ and security parameter $k$, and running with a set of parties and an adversary $\mathcal{S}$. At the first activation a set $R$ of strings is initialized to be empty.

**Registration:** When receiving a message $(\texttt{register}, sid)$ from a party $P_i$ (which is either corrupted or uncorrupted), send $(\texttt{register}, sid, P_i)$ to $\mathcal{S}$ and receive a value $p'$ from $\mathcal{S}$. Then, if $p' \in R$ then let $p \leftarrow p'$. Else, choose $r \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}^k$, let $p \leftarrow f(r)$, and add $p$ to $R$. Finally, record $(P_i, p)$ and return $(sid, p)$ to $P_i$ and to $\mathcal{S}$.

**Registration by a corrupted party:** When receiving a message $(\texttt{register}, sid, r)$ from a corrupted party $P_i$, record $(P_i, f(r))$. In this case, $f(r)$ is *not* added to $R$.

**Retrieval:** When receiving a message $(\texttt{retrieve}, sid, P_i)$ from party $P_j$, send $(\texttt{retrieve}, sid, P_i, P_j)$ to $\mathcal{S}$, and obtain a value $p$ from $\mathcal{S}$. If $(P_i, p)$ is recorded then return $(sid, P_i, p)$ to $P_j$. Else, return $(sid, P_i, \perp)$ to $P_j$.

**Figure 2. The Key Registration functionality for function $f$, $\mathcal{F}_{kr}^f$**

---

Notice that the uncorrupted parties do not obtain any secret keys associated with their public keys, whereas the corrupted parties may know the secret keys of their public keys. Furthermore, $\mathcal{F}_{kr}$ gives the adversary a fair amount of freedom in choosing the registered keys. It can set the keys associated with corrupted parties to be any arbitrary value (as long as the value is "well-formed", in the sense that functionality sees the corresponding private key). The adversary can also cause the keys of both corrupted and uncorrupted parties to be identical to the keys of other (either corrupted or uncorrupted) parties. Still, $\mathcal{F}_{kr}$ guarantees two basic properties: **(a)** the public keys of good parties are "safe" (in the sense that their secret keys were chosen at random and kept secret from the adversary), and **(b)** the public keys of the corrupted parties are "well-formed", in the sense that the functionality has seen the corresponding secret keys. We demonstrate that $\mathcal{F}_{kr}$ can be realized using a variety of mechanisms:

**Realizing $\mathcal{F}_{kr}$ in the $\mathcal{F}_{crs}$-hybrid model.** We first demonstrate that $\mathcal{F}_{kr}^{f}$ can be easily realized in the $\mathcal{F}_{crs}^{D}$-hybrid model, where $D = D_k$ is the distribution of $f(r)$ where $r$ is uniform in $\{0,1\}^k$. The protocol is straightforward: On input either $(\texttt{register}, sid)$ or $(\texttt{retrieve}, sid, P_j)$, party $P_i$ sends $(\texttt{CRS}, sid)$ to $\mathcal{F}_{crs}$ and returns the obtained value. The following proposition is proven in [BCNP04]:

**Proposition 1** *The above protocol securely realizes $\mathcal{F}_{kr}^{f}$ in the $\mathcal{F}_{crs}^{D}$-hybrid model.*

**Realizing $\mathcal{F}_{kr}$ given a registration service with knowledge.** Next, consider an alternative setting where the registration service lets the registering parties choose their public keys on their own, but insists on seeing the corresponding secret keys. This model reduces the trust in the service: no longer is it trusted to make truly random choices. It is only trusted to verify "well-formedness" of the keys. In particular, the keys of corrupted parties are no longer guaranteed to be randomly chosen. (This setting is also very close to the standard "public-key infrastructure" setting.) We let $\mathcal{F}_{krk}$ (for key registration with knowledge) denote the ideal functionality that captures this registration service. We show how to realize $\mathcal{F}_{kr}$ in this setting. On input $(\texttt{register}, sid)$, party $P_i$ chooses a random $r_i \stackrel{R}{\leftarrow} \{0,1\}^k$, provides $r_i$ to the registration authority, computes $p_i = f(r_i)$, *erases* $r_i$, and returns $p_i$.

**Proposition 2** *The above protocol securely realizes $\mathcal{F}_{kr}^{f}$ in the $\mathcal{F}_{krk}^{f}$-hybrid model.*

**Realizing $\mathcal{F}_{kr}$ with semi-trusted registration service.** The above two settings assumed that the registration service is completely trusted to perform its task according to the specification. Alternatively, we can consider a setting where there are multiple registration services, where no single service is trusted by all the parties. Instead, each party is willing to "fully trust" only *one* of the services (the one it registered with), and in addition it is willing to "partially trust" all other ones. More specifically, the service that the party registered with is trusted to keep its secret key secret. The only trust put in other services is that they agree to record only "well-formed public keys" that were computed based on *some* secret key. The keys can be copies of each other, and the secret keys can be made public. Again, $\mathcal{F}_{kr}$ can be realized even in such a setting via essentially the same protocol.

**Realizing $\mathcal{F}_{kr}$ using traditional proofs of knowledge.** Finally, we consider the case where the registration service is similar to that of $\mathcal{F}_{krk}$, except that the parties do not provide the seed $r$ explicitly to the service. Instead, each registering party provides $f(r)$, and in addition engages with the service in a traditional (non-UC) zero-knowledge proof of knowledge of $r$. This protocol does *not* realize $\mathcal{F}_{kr}$ in the plain UC framework. Still, if we assume that there is no network activity during the execution of each zero-knowledge proof (formally, if we assume that the environment does not interact with the adversary for the duration of the zero-knowledge proof with $\mathcal{F}_{krk}$), then this protocol *does* realize $\mathcal{F}_{kr}$. Such an assumption may be reasonable in some cases.

## 4. Realizing any well-formed functionality

We show how to realize any well-formed functionality in the $\mathcal{F}_{kr}$-hybrid model. By the following result from [CLOS02] it suffices to realize $\mathcal{F}_{mcom}$ (presented in Figure 3) in the $\mathcal{F}_{kr}$-hybrid model.

**Theorem 3** *If there exists augmented non-committing encryption, then any well-formed functionality can be realized in the $\mathcal{F}_{mcom}$-hybrid model.*[2]

---

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

$\mathcal{F}_{mcom}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Upon receiving a value $(\texttt{Commit}, sid, cid, P_i, P_j, x)$ from $P_i$, where $x \in \{0,1\}$, record the tuple $(cid, P_i, P_j, x)$ and send the message $(\texttt{Receipt}, sid, cid, P_i, P_j)$ to $\mathcal{S}$. On a subsequent message $(\texttt{Receipt}, sid, cid, P_i, P_j)$ from $\mathcal{S}$, send $(\texttt{Receipt}, sid, cid, P_i, P_j)$ to $P_j$ and ignore subsequent $(\texttt{Commit}, sid, cid, P_i, P_j, ...)$ values.
2. Upon receiving a value $(\texttt{Open}, sid, cid, P_i, P_j)$ from $P_i$: If the tuple $(cid, P_i, P_j, x)$ is recorded then send the message $(\texttt{Open}, sid, cid, P_i, P_j, x)$ to $P_j$ and $\mathcal{S}$. Otherwise, do nothing.

**Figure 3. The ideal functionality for Multiple COMmitments**

---

**Realizing $\mathcal{F}_{mcom}$.** All known UC commitment schemes [CF01, CLOS02, DN02] are for the CRS model. A general technique of separating the trust in the CRS allows to adopt them to the $\mathcal{F}_{kr}$-hybrid model. We demonstrate this technique for the scheme in [CLOS02]. The full version contains a detailed treatment of all mentioned schemes.

**The CLOS scheme.** The protocol from [CLOS02], which is a variant of the scheme in [CF01], uses a CRS $v = (c, e)$,

2 An augmented non-committing encryption scheme is a realization of the ideal functionality for secure message transmission meeting an extra technical requirement. Such a protocol exists e.g. under the DDH assumption.

where $c$ is a commitment key for a statistically hiding trapdoor commitment scheme and $e = (e^{cca}, e^{prc})$ is an encryption key for an encryption scheme of the form $E_e(m) = E^{prc}_{e^{prc}}(E^{cca}_{e^{cca}}(m))$, where $E^{prc}$ has pseudo-random ciphertexts (PRC)[3] and $E^{cca}$ is CCA secure; Below we call such an encryption scheme a CLOS encryption scheme. A commitment to a bit $x$ is of the form $(C, E_0, E_1)$, where $C = commit_c(x; r)$, $E_x = E_e(r; s)$ and $E_{1-x} \stackrel{R}{\leftarrow} \{0,1\}^{|E_x|}$; Here $r$ and $s$ are the random bits used by $commit_c$ respectively $E_e$. To open the commitment one sends $(x, r, s)$. The scheme is binding as an opening to $x$ includes an opening of $C$ to $x$. The scheme is hiding by the hiding of $commit_c$ and the CCA security of $E_e$. In addition to being binding and hiding, a realization of $\mathcal{F}_{mcom}$ should have the following two properties: *Simulation equivocality:* The simulator should be able to produce *equivocal commitments* for which it can open to both 0 and 1. *Simulation extractability:* The simulator should be able to extract the contents of any openable commitment, even after supplying an adversary with arbitrary many equivocal commitments and their openings.

Simulation equivocality is achieved using the trapdoor of $commit_c$, which allows to compute $(C, r_0, r_1)$ such that $C = commit_c(0; r_0) = commit_c(1; r_1)$ and compute $(C, E_0 = E_e(r_0; s_0), E_1 = E_e(r_1; s_1))$; This looks like a real commitment by the PRC assumption. To open the commitment to $x$ the simulator sends $(x, r_x, s_x)$. Simulation extractability is achieved using the decryption key $d$ of $E_e$. When the simulator receives a commitment $(C, E_0, E_1)$ from the adversary it computes $r_0 = D_d(E_0)$ and $r_1 = D_d(E_1)$. If $(C, E_0, E_1)$ is openable, then either $C = commit_c(0; r_0)$ or $C = commit_c(1; r_1)$, which allows the simulator to determine the bit $x$. Since the adversary does not know the trapdoor of $commit_e$, both possibilities happen simultaneously only with negligible probability.

Two observations allow to adopt the above scheme to the $\mathcal{F}_{kr}$-hybrid model. First of all, each property of the primitives used to build the UC commitment scheme is in the interest of *either* the sender *or* the receiver, never both. Second, some of the properties hold as long as the keys are *well-formed*, i.e. $e$ is *an* encryption key and $c$ is *a* commitment key. We go over the properties and observe which parties are interested in which and when they hold: *Computational binding of $commit_c$:* Prevents double openings; In the interest of the receiver; Holds when $c$ is a random key. *Equivocality of $commit_c$:* Used to construct equivocal commitments; In the interest of the sender; Holds when $c$ is well-formed and the simulator can access the trapdoor. *Statistical hiding of $commit_c$:* Needed for the overall commitment to be hiding; In the interest of the sender; Holds when $c$ is well-formed. *Computational hiding of $E_e$:* Needed for the overall commitment to be computational hiding; In the interest of the sender; Holds when $e$ is random. *Decryption of $E_e$:* Ensures extractability; In the interest of the receiver; Holds when $e$ is well-formed and the simulator can access the decryption key. *Pseudo-random ciphertext of $E_e$:* Needed for equivocal commitments to look like real commitments; In the interest of the sender; Holds when $e$ is random. We learn that the trust is asymmetric: The sender is interested in $e$ being random and $c$ being well-formed, and the receiver is interested in $e$ being well-formed and $c$ being random. This can be guaranteed by the sender generating and registering $e$ and the receiver generating and registering $c$.[4] The resulting scheme is presented in Figure 4.

**Proposition 4** *If there exist CCA secure encryption, PRC secure encryption and statistically hiding trapdoor commitments, then there exists a realization of $\mathcal{F}_{mcom}$ in the $\mathcal{F}_{kr}$-hybrid model.*

**Keying only the committer.** In the above commitment scheme both parties must register public keys. Using a tool from [DN02], called a mixed commitment scheme, we can get a scheme where only the sender needs to have registered a public key. This is inconsequential to the main goal of realizing any well-formed functionality in the $\mathcal{F}_{kr}$-hybrid model, but has practical consequences as it e.g. allows to realize any two party functionality if just one of the two parties has registered a public key.[5]

A *mixed commitment scheme* is a commitment scheme $(genE, genX, commit)$ with a usual commitment function $commit$ and *two* key generators $genE$ and $genX$. A key generated as $K = genE(r)$ is called an *E-key* and $r$ is called the *E-trapdoor*. A key generated as $K = genX(r)$ is called an *X-key* and $r$ is called the *X-trapdoor*. When $K$ is an E-key, $commit_K$ is computationally hiding and *E*quivocal (given the *E-trapdoor*). When $K$ is an X-key, $commit_K$ is perfect binding and e*X*tractable (given the *X-trapdoor*). The term *mixed* refers to the *key indistinguishability* requirement that random keys sampled using $genE$ and $genX$ are computationally indistinguishable. The main motivation for considering mixed commitment schemes is that the equivocation property of a commitment scheme is predominantly a

3 A PRC encryption scheme $E$ has the properties that all ciphertexts under an encryption key $e$ have the same length $l_e$ and that an encryption $C \leftarrow E_e(m)$ of a chosen message $m$ is computationally indistinguishable from a uniformly random string $C' \stackrel{R}{\leftarrow} \{0,1\}^{l_e}$. Such an encryption scheme exists e.g. under the DDH assumption or the RSA assumption.

4 The simulator for the $\mathcal{F}_{kr}$-hybrid model also simulates $\mathcal{F}_{kr}$ and therefore, as required, learns the trapdoor (respectively decryption key) of all registered commitment (respectively encryption) keys.

5 Having a UC commitment scheme allowing $P_1$ to commit to $P_2$ allows $P_1$ and $P_2$ to realize $\mathcal{F}^D_{crs}$ for the distribution where $D$ is a uniformly random string (by running a Blum coin-flip protocol). Having realized $\mathcal{F}^D_{crs}$ the parties can then run the protocol from [CLOS02], which is for the $\mathcal{F}^D_{crs}$-hybrid model. Details can be found in [C04, CR03].

**Protocol CLOS-KR**

CLOS-KR proceeds as follows, running with parties $P_1, ..., P_n$ in the $\mathcal{F}_{kr}^f$-hybrid model, where $f(r) = (e, c)$ with $e$ being an encryption key for a CLOS encryption scheme and $c$ being a key for a statistically hiding trapdoor commitment scheme.

1. Upon its initial activation with session id $sid$ the party $P_i$ inputs $(\texttt{register}, sid)$ to $\mathcal{F}_{kr}^f$, waits for a value $(sid, p_i = (e_i, c_i))$ from $\mathcal{F}_{kr}$ and stores $p_i$.
2. Upon receiving an input $(\texttt{Commit}, sid, cid, P_i, P_j, x)$, where $x \in \{0,1\}$, the party $P_i$ inputs $(\texttt{retrieve}, sid, P_j)$ to $\mathcal{F}_{kr}^f$ and waits for a value $(sid, P_j, p_j)$ from $\mathcal{F}_{kr}$. If $p_j = \perp$ then $P_j$ terminates the protocol with commitment id $cid$. Otherwise, it let's $p_j = (e_j, c_j)$, computes $C = commit_{c_j}(x; r)$, $E_x = E_{e_i}(r; s)$ and $E_{1-x} \xleftarrow{\text{R}} \{0,1\}^{|E_x|}$, stores $(sid, cid, x, r, s)$ and sends $(sid, cid, C, E_0, E_1)$ to $P_j$.
3. Upon receiving a value $(sid, cid, C, E_0, E_1)$ from $P_i$, where $cid$ was not used before, the party $P_j$ inputs $(\texttt{retrieve}, sid, P_i)$ to $\mathcal{F}_{kr}^f$ and waits for a value $(sid, P_i, p_i)$ from $\mathcal{F}_{kr}$. If $p_i = \perp$ then $P_j$ terminates the protocol with commitment id $cid$. Otherwise, it let's $p_i = (e_i, c_i)$, stores $(sid, cid, P_i, e_i, C, E_0, E_1)$ and outputs $(\texttt{Receipt}, sid, cid, P_i, P_j)$.
4. Upon receiving an input $(\texttt{Open}, sid, cid, P_i, P_j)$, where a value $(sid, cid, x, r, s)$ is stored, the party $P_i$ sends $(\texttt{Open}, sid, cid, x, r, s)$ to $P_j$.
5. Upon receiving for the first time a value $(sid, cid, x, r, s)$ from $P_i$, where a value $(cid, sid, P_i, e_i, C, E_0, E_1)$ is stored and $C = commit_{c_j}(x; r)$ and $E_x = E_{e_i}(r; s)$, the party $P_j$ outputs $(\texttt{Open}, sid, cid, P_i, P_j, x)$.

**Figure 4. The CLOS commitment scheme for the $\mathcal{F}_{kr}^f$-hybrid model**

property that is solely needed in simulation. A mixed commitment scheme allows to use a perfect binding commitment scheme in the real-world and then indistinguishably replace it by an equivocal commitment scheme for the sake of simulation and/or analysis. We demonstrate this technique.

Recall that the receiver picks the commitment key in the CLOS scheme to guarantee binding and that the commitment scheme was statistically hiding to guarantee that it is hiding even when the receiver picks the key. If instead we let the *sender* register also the commitment key $c$, but now as an X-key for a mixed commitment scheme, then $commit_c$ is perfect binding and the receiver is again guaranteed that $commit_c$ is binding (in fact, now the obtained UC commitment scheme is perfect binding). To get back simulation equivocality, the simulator simply picks the registered key $c$ to be a random E-key instead of a random X-key. This is possible as the simulator simulates $\mathcal{F}_{kr}$, and by the key indistinguishability this change of key space will go unnoticed by the adversary. After the change of key space $commit_c$ is again a trapdoor commitment scheme, and the overall scheme is identical to the first CLOS scheme for the $\mathcal{F}_{kr}$-hybrid model. The analysis then follows that of the CLOS scheme. In [BCNP04] we show how to realize mixed commitment schemes based on PRC encryption, giving us the following result.

**Proposition 5** *If there exist CCA secure encryption and PRC secure encryption, then there exists a sender-keyed realization of $\mathcal{F}_{mcom}$ in the $\mathcal{F}_{kr}$-hybrid model.*

## 5. Non-interactive UC zero-knowledge

In this section we show that, as in the CRS model, we can in fact obtain a *non-interactive* UC zero-knowledge argument system in the $\mathcal{F}_{kr}$-hybrid model.[6] More precisely, we show to to realize the multi-session extension of the ideal zero-knowledge functionality $\hat{\mathcal{F}}_{zk}$ (presented in Figure 5) in the $\mathcal{F}_{kr}$-hybrid model.

**Functionality $\hat{\mathcal{F}}_{zk}^R$**

$\hat{\mathcal{F}}_{zk}^R$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$, given a binary relation $R$.

1. Upon a value $(\texttt{ZK-prover}, sid, ssid, P_i, P_j, x, w)$ from a party $P_i$: If $(x, w) \in R$, then send $(\texttt{ZK-proof}, sid, ssid, P_i, P_j, x)$ to $P_j$ and $\mathcal{S}$. Otherwise, ignore.

**Figure 5. The multi-session Zero-Knowledge functionality for relation $R$, $\hat{\mathcal{F}}_{zk}^R$**

The main components of our construction are a CCA secure encryption scheme (with errorless decryption), and the ZAP system of Dwork and Naor [DN00], which is a two-round public-coins witness indistinguishable (WI) proof system for $\mathbf{NP}$. Protocol UC-NIZK(presented in Figure 6) is our non-interactive UC zero-knowledge protocol. Roughly speaking, the protocol's outline is as follows: the public key for the protocol consists of a triple $(e, v, z)$, where $e$ is a public key for the encryption scheme, $v$ is the first message of the ZAP system, and $z = g(y)$, where $g$ is a one-way function and $y$ is a uniformly chosen string. To prove that a statement $x$ is in

6 As in the CRS model, we rely on erasures to obtain a protocol that is secure against adaptive adversaries. Alternatively, without erasures our protocol (as well as known non-interactive zero-knowledge protocols in CRS model) are only secure against static adversaries.

**Protocol UC-NIZK**

$UC-NIZK$ proceeds as follows, running with parties $P_1,...,P_n$ in the $\mathcal{F}_{kr}^{f}$ hybrid model.

**Key Generation:** The function $f$ is given by $f(1^k, r) = (e, v, z)$, where $f$ splits $r$ into $r_0, v, y$, and $r_0$ is used to generate a encryption/decryption key pair $(e, d) = gen(1^k, r_0)$ for a CCA secure encryption scheme $(gen, E, D)$ with errorless decryption, $v$ is the first message for a ZAP system, and $y$ is used to compute $z = g(y)$, where $g$ is a one-way function.

**Initialization:** Upon its initial activation with session identifier $sid$, party $P_i$ sends the message (register, $sid$) to $\mathcal{F}_{kr}^{f}$ and waits to receive back the message $(sid, p_i = (e_i, v_i, v_z))$ from $\mathcal{F}_{kr}^{f}$. $P_i$ then stores $p_i$. Parties retrieve keys of other parties as in Figure 4; Below we describe how the protocol proceeds between $P_i$ and $P_j$ when both parties succeed in retrieving the other party's key $(e_j, v_j, z_j)$ respectively $(e_i, v_i, z_i)$.

**Prover:** On input (ZK-prover, $sid, ssid, P_i, P_j, x, w$) party $P_i$ proceeds as follows:

1. compute $c_0 = E_{e_i}(P_i, P_j, sid, ssid, w)$
2. compute $c_1 = E_{e_i}(P_i, P_j, sid, ssid, 0^k)$
3. compute a ZAP proof $\pi$ for the statement that *either* $c_0$ is an encryption, using $e_i$ as encryption key, of the string $(P_i, P_j, sid, ssid, w)$, where $w$ is a witness for $x$ (i.e., $(x, w) \in R$) *or* $c_1$ is an encryption, using $e_i$ as encryption key, of the string $(P_i, P_j, sid, ssid, y)$ where $z_j = g(y)$. The ZAP proof is computed with respect to $v_j$
4. *erase* all randomness used to generate $(c_0, c_1, \pi)$.
5. send (PROOF, $sid, ssid, x, c_0, c_1, \pi$) to $P_j$.

**Verifier:** Upon receiving (PROOF, $sid, ssid, x, c_0, c_1, \pi$) from party $P_i$, party $P_j$ proceeds as follows:

1. $P_j$ verifies that $\pi$ is an accepting ZAP with respect to the message $v_j$ (i.e., the second part of the verifier's public key).
2. If the proof is accepting, $P_j$ outputs (ZK-proof, $sid, ssid, P_i, P_j, x$).

**Figure 6. The UC Non-Interactive Zero-Knowledge Protocol**

some $\mathbf{NP}$-language $L$, the prover first computes two encryptions $c_0, c_1$ using the encryption key specified by the first part of its own public key: $c_0$ is an encryption of a witness that $x \in L$, $c_1$ is an encryption of the string $0^k$. The prover then uses the ZAP system, with respect to the verifier message specified by the second part of verifier's public key, to prove that either $c_0$ is an encryption of a witness for $x$ or $c_1$ is an encryption of $g^{-1}(z)$ where $z$ is the third part of the verifier's public key (i.e., the prover shows that either it knows a witness to the statement $x$, or that it knows (the third part of) the verifier's secret key).[7]

We defer the proof of security of the protocol to the full version of this paper. However, we will sketch the intuition behind why it is secure here:

**simulation:** To simulate a proof, the simulator only needs to know a string $y$ such that $g(y) = z$ (where $g$ is the one-way function, and $z$ is the third part of the verifier's public key). Thus, it is not hard to simulate given the randomness used to construct (this part of) the verifier's public key. Note that only the verifier's secret key is used in this process. Also, we do not require that the verifier's key is random but only that it is well formed.

**extraction:** To extract a witness, the simulator simply decrypts $c_0$ using the prover's secret key. Once again, we only use the fact that the prover's key is well formed.

Since both CCA secure encryption schemes, with errorless decryption, and ZAP systems are known to exist under the assumption of enhanced trapdoor permutations [DDN00, DN00], we thus obtain the following theorem.

**Theorem 6** *Assume the existence of enhanced trapdoor permutations. Then, there exists a non-interactive protocol, with erasures, that realizes $\hat{\mathcal{F}}_{zk}^{R}$ in the $\mathcal{F}_{kr}$-hybrid model. Furthermore, there exists a non-interactive protocol (without erasures) that realizes $\hat{\mathcal{F}}_{zk}^{R}$ in the $\mathcal{F}_{kr}$-hybrid model, with respect to static adversaries.*

## 6. Handling failures gracefully

The CRS model is a very clean and useful model. However, as mentioned above, when the CRS model fails it fails spectacularly. By this we mean that if the common reference string is generated by a corrupted authority, then this authority is able to *completely break* the security of the system, compromising completely both the secrecy and the integrity of the system in a way that is completely undetectable. In contrast, we show that in our model it is possible to construct protocols that enjoy a more "graceful" security analysis. These protocols will be UC-secure in the case that our assumptions about the other parties' public keys are satisfied (essentially, in the case that other parties' public keys are well-formed). However, even if these assumptions are *not* satisfied, we will still be able to analyze these protocols and show that they are *stand-alone* secure.[8] We note

7 In order to obtain security also against adaptive adversaries we furthermore require that the prover *erases* all random coins used to construct the proof.

8 This is under the assumption that we can trust our own public key. We believe this is a reasonable assumption as any party can choose its own public key and may even erase the corresponding private key later to avoid leakage (as honest parties in our protocols do *not* need to use their private key). Also, an honest party $P$ can refuse to partici-

that without some kind of setup assumptions, it is not possible to obtain UC security, and thus in some sense this is the best that we can hope for.[9]

The crucial observation for constructing such protocols is that the commitment scheme in Proposition 4 is actually stand-alone secure. By this we mean that the binding property only requires the receiver's public key to be generated at random (and does *not* require the sender's key to be even well-formed), and the hiding property only requires the sender's public key to be generated at random (and does *not* require the receiver's key to be even well-formed).[10] Thus this commitment scheme already enjoys the kind of "graceful" analysis we were looking for. We call such a commitment scheme (that is analyzed to be stand-alone secure whenever the honest party's key is random, and in addition is UC secure if the adversary's key is well-formed) a *graceful commitment scheme*.

We use a graceful commitment scheme to construct a "graceful" protocol for *any* functionality. The key step is constructing a "graceful" zero-knowledge argument system for $\mathbf{NP}$ (since then we can implement the "commit-and-prove" functionality which can be used to implement any functionality by the results of [CLOS02]). For this, we use the results of Canetti and Fischlin [CF01] on how to use a UC secure commitment to construct a UC secure zero knowledge protocol.[11] Thus, if we take a constant-round zero-knowledge protocol, such as the one of Feige and Shamir [FS89], and use there our graceful commitment scheme as the underlying commitment scheme, we get a protocol that is **(a)** stand-alone zero-knowledge if the honest party's keys random by the standard analysis of the zero-knowledge protocol and the fact that the graceful commitment is stand-alone secure under this assumption and **(b)** UC-secure zero-knowledge if in addition the adversary's key is guaranteed to be well-formed. Thus, we get a graceful zero-knowledge protocol. Using such a zero-knowledge protocol, we can follow the approach of [CLOS02] to construct a graceful protocol for any functionality. We defer the details to the full version of this work.

---

pate in a protocol if a corrupt CA claims than $P$'s public key is different from the correct value.

9 For more discussion what can and cannot be done without setup assumption, see the recent work of Lindell [L04] and the references therein, and the very recent work of Prabhakaran and Sahai [PS04].

10 This claim requires that the receiver can verify that the key $e$ of the committer defines a binding encryption function $E_e$; Such schemes can be constructed under standard assumptions. Furthermore, the sender should be able to verify that the key $c$ of the committer defines a hiding commitment scheme $C = commit_c(x)$; If this is not already the case, then the sender can be protected by encrypting the commitment and send $E_e(C)$.

11 [CF01] proved that if you plug a UC-secure commitment into Blum's parallel Hamiltonicity protocol then it will be UC zero knowledge. However, the same analysis hold for many other honest-verifier zero-knowledge protocols. In particular, we use their result to show that this holds for the zero-knowledge argument system of [FS89].

# References


[BCNP04] B. Barak, R. Canetti, J.B. Nielsen and R. Pass. Universally composable protocols with relaxedset-up assumptions. Eprint archive, http://eprint.iacr.org/2004.

[B91] D. Beaver. Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. *Journal of Cryptology*, 4:75–122, 1991.

[1] J. Benaloh and D. Tunistra. Receipt-Free Secret-Ballot Elections. *26th STOC,* 1994, pp. 544-552.

[BFM88] M. Blum, P. Feldman, and S. Micali. Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract). In *Proceedings of 20th STOC*, pp. 103–112, 1988.

[C00] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[C01] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proc. 42st FOCS*, pp. 136–145. IEEE, 2001. http://eprint.iacr.org/2000/067.

[C04] R. Canetti. Universally Composable Signature, Certification, and Authentication. *17th IEEE Computer Security Foundations Workshop,* 2004. Long version in http://eprint.iacr.org/2003/239.

[CDNO97] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In *Crypto '97*, pp. 90–104, 1997.

[CF01] R. Canetti and M. Fischlin. Universally composable commitments. In *Crypto 2001*, pp. 19–40, 2001.

[CKL03] R. Canetti, E. Kushilevitz and Y. Lindell. On the Limitations of Universally Composable Two-Party Computation Without Set-Up Assumptions. In Eurocrypt '03, pp. 68–86, 2003.

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the 24th STOC*, pp. 494–503, 2002.

[CR03] R. Canneti and T. Rabin. Universal composition with joint state. In *Crypto 2003*, pp. 265–281, 2003.

[DDN00] D. Dolev, C. Dwork and M. Naor. Non-Malleable Cryptography. *SIAM Jour. on Computing*, Vol. 30(2), pp. 391–437, 2000.

[DDO+01] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust Non-interactive Zero Knowledge. In *CRYPTO ' 2001*, pp. 566–598, 2001.

[DN02] I. Damgård and J. B. Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In *Crypto 2002*, pp. 581–596, 2002.

[DIO98] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Non-interactive and non-malleable commitment. In *Proceedings of the 30th STOC*, pp. 141–150, 1998.

[DN00] C. Dwork and M. Naor. Zaps and Their Applications. In *Proc. 41st FOCS*, pp. 283–293. IEEE, 2000.

[DNS98] C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. In *Proceedings of the 20th STOC*, pp. 409–418, 1998.

[FLS99] Feige, Lapidot, and Shamir. Multiple Noninteractive Zero Knowledge Proofs Under General Assumptions. *SIAM J. Comput.*, 29, 1999.

[FS89] U. Feige and A. Shamir. Zero-knowledge proofs of knowledge in two rounds. In *Crypto '89*, pp. 526–544, 1989.

[GK90] O. Goldreich and H. Krawczyk. On the composition of zero knowledge proof systems. In *Proceedings of ICALP 90*, Berlin, 1990. Springer-Verlag. Lecture Notes in Computer Science Volume 443.

[GL00] O. Goldreich and Y. Lindell. Session-Key Generation using Human Passwords Only. Report 2000/057, Cryptology ePrint Archive, Nov. 2000. Extended abstract in CRYPTO 2001.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the 19th STOC*, pp. 218–229, 1987.

[GOL01] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[HLM03] J. Herzog, M. Liskov and S. Micali. Plaintext-Awareness Through via Key Registration. CRYPTO'03. LNCS 2729, 548-564, 2003.

[JSI96] M. Jakobsson, K. Sako and R. Impagliazzo. Designated verifier proofs and their applications. Eurocrypt'96, LNCS 1070, pp. 143-154, 1996.

[KSW97] J. Kelsey, B. Schneier and D. Wagner. Protocol interactions and the chosen protocol attack. Security Protocols Workshop, 1997.

[L03] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. In 44th FOCS, pp. 394-403, 2003.

[L04] Y. Lindell. Lower Bounds for Concurrent Self Composition. In the 1st Theory of Cryptography Conference (TCC '04), pp. 203-222, 2004.

[MR91] S. Micali and P. Rogaway. Secure Computation. In *Crypto '91*, pp. 392–404, 1991. Manuscript available.

[PS04] M. Prabhakaran and A. Sahai. New Notions of Security: Achieving Universal Composability without Trusted Setup To appear in STOC' 04.

[PSW00] B. Pfitzmann, M. Schunter, and M. Waidner. Secure Reactive Systems. IBM Research Report RZ 3206 (#93252), IBM Research, Zurich, May 2000.