Treading the Impossible: A Tour of Set-Up Assumptions for Obtaining Universally Composable Security*

Ran Canetti

IBM T.J. Watson Research Center

Abstract. A desirable goal for cryptographic protocols is to guarantee security when the protocol is composed with other protocol instances. Universally Composable (UC) security provides this guarantee in a strong sense: A UC-secure protocol maintains its security properties even when composed concurrently with an unbounded number of instances of arbitrary protocols. However, many interesting cryptographic tasks are provably impossible to realize with UC security, unless some trusted set-up is assumed. Impossibility holds even if ideally authenticated communication channels are provided.

This brief survey examines and compares a number of set-up assumptions (models) that were recently demonstrated to suffice for constructing UC-secure protocols that realize practically any cryptographic task. We start with the common reference string (CRS) and key registration (KR) models. We then proceed to the "sunspot" models, which allow for some adversarial control over the set-up, a number of models which better captures set-up that is globally available in the system, and a timing assumption. Finally, we briefly touch upon set-up models for obtaining authenticated communication.

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., [GK89,DDN00,KSW97,DNS98,KLR06,Can06]). 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.,

* This survey complements a talk given by the author at this conference. Work supported by NSF grants CT-0430450 and CFF-0635297, and US-Israel Binational Science Foundation Grant 2006317.

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 UC 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.

Which cryptographic tasks are realizable by protocols that guarantee UC security? Are existing protocols, which are known to be secure in a stand-alone setting, UC secure? When the majority of the parties are honest (i.e., they are guaranteed to follow the protocol), the general feasibility results for stand-alone secure computations can be extended to the case of UC security. In fact, some known protocols for general secure function evaluation turn out to be UC secure. For instance, the [BGW88] protocol (both with and without the simplification of [GRR98]), together with encrypting each message using non-committing encryption [CFGN96], is universally composable as long as less than a third of the parties are corrupted, and authenticated and synchronous communication is available. Using [RB89], any corrupted minority is tolerable. Asynchronous communication can be handled using the techniques of [BCG93,BKR94]. Note that here some of the participants may be "helpers" (e.g., dedicated servers) that have no local inputs or outputs; they only participate in order to let other parties obtain their outputs in a secure way.

However, things are different when honest majority of the parties is not guaranteed, and in particular in the case where only two parties participate in the protocol and either one of the parties may be corrupted: It turns out that many interesting tasks are impossible to realize in the "bare" model of computation. Impossibility holds even if ideally authenticated communication is guaranteed. (In keeping with common terminology, we use the terms plain protocols and protocols in the plain model to denote protocols that assume ideally authenticated communication but no other set-up.) For instance, basic cryptographic tasks such as Bit Commitment, Coin-Tossing, Zero-Knowledge, or Oblivious Transfer cannot be realized by plain protocols, when naturally translated to the UC framework. Impossibility also extends to many other tasks [CF01,C01,CKL03,DDMRS06], including multi-party extensions of these primitives, whenever the honest parties are not in majority.

One potential approach for circumventing these impossibility results is to come up with relaxed notions of security that would still guarantee meaningful composable security, and at the same time would be realizable by plain protocols. It turns out, however, that such an approach will necessarily result in notions of security that either do not provide general composability guarantees, or alternatively are too weak to guarantee even stand-alone security as in, say, [C00] (see e.g. [L03,L04,Can06]). Still, some meaningful such relaxations exist, see e.g. [PS04,BS05,MMY06].

Another approach is to stick with UC security, but consider protocols that rely on some trusted set-up assumption on the system. Here the meaningfulness of the security guarantee hinges on the "reasonability" of the set-up assumption, or in other words on the ability to realize the assumed set-up in actual systems.

This survey studies some set-up assumptions (or, models) that were recently proposed and shown to suffice for realizing essentially any cryptographic task in a universally composable way. The various set-up models are compared to each other, and the relative strengths and weaknesses are discussed.

The survey is organized as follows. Section 2 provides a brief review of the UC security framework. Section 3 reviews the basic impossibility result for obtaining UC commitment in the plain model. Section 4 reviews the common reference string (CRS) set-up. Section 5 reviews the key registration (KR) set-up. Section 6 reviews the adversarially controlled CRS (Sunspot) set-up. Section 7 reviews the augmented CRS (ACRS) set-up. Section 8 reviews the first set-up assumption, which relates to the delays on message delivery. Section 9 briefly discusses st-up assumptions for the purpose of obtaining authenticated communication. Section 10 concludes and discussed some open problems.

2 UC security: A brief review

This section briefly reviews the UC framework. As in many other frameworks (e.g., [GL90,MR91,B91,C00,PW00,PW01]), the security of protocols with respect to a given task is defined via the "trusted party paradigm" [GMW87], where the protocol execution is compared with an ideal process where the outputs are computed by a trusted party that sees all the inputs. That is, a protocol is said to securely carry out a given task if running the protocol with a realistic adversary amounts to "emulating" the ideal process with the appropriate trusted party. We call the algorithm run by the trusted party an ideal functionality.

The UC framework substantiates this approach as follows. First, the process of executing a protocol in the presence of an adversary and in a given computational environment is substantiated. Next, the "ideal process" for carrying out the task is substantiated. Finally, one defines what it means for an execution of the protocol to "mimic" the ideal process. We sketch these three steps.

The model of protocol execution. The model for executing an multiparty protocol π consists of a system of computing elements (modeled as interactive Turing machines, or ITMs) $(\mathcal{Z}, \mathcal{A}, M_1, M_2, ...)$ where $\mathcal{Z}$ and $\mathcal{A}$ are adversarial entities called the environment and adversary, respectively, and the machines $M_1, M_2, ...$ represent parties that run an "extended instance" of π. (An *instance* of protocol π is a set of ITMs that run π and in addition have a common identifier, called the *session ID*. The number of parties in an instance may vary from instance to instance, as well as during the lifetime of an instance.) Intuitively, the environment represents all the other protocols running in the system, including the protocols that provide inputs to, and obtain outputs from, the protocol instance under consideration. The adversary represents adversarial activities that are directly aimed at the protocol execution under consideration, including attacks on protocol messages and corruption of protocol participants.

An execution of the system consists of a sequence of activations of the individual elements, where the environment is activated first, and in each activation the active element determines the next element to be active, by sending information to it. This information may be labeled as either input, output, or protocol message. We impose the following restrictions on the way in which the above system runs. The environment $\mathcal{Z}$ is allowed to provide only *inputs* to other machines. A party of π may send messages to $\mathcal{A}$, or give inputs to the environment. The adversary $\mathcal{A}$ may give output to $\mathcal{Z}$ or send messages to other parties.

Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(z)$ denote the random variable (over the local random choices of all the involved machines) describing the output of environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and parties running protocol π on input z as described above. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(z)\}_{z\in\{0,1\}^*}$. We restrict attention to the case where the environment outputs only a single bit; namely, the ensemble $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ is an ensemble of distributions over $\{0,1\}$.

Subroutines. For the purpose of formulating the ideal process and the notion of protocol composition it will be convenient to allow designating an ITM as a *subroutine* of another ITM. If an ITM M is a subroutine of M' then M' can give input to M and M can give output to M'. Note that M and M may have different session ID and run different codes. The above model of protocol execution is then extended in the natural way to protocols where the parties have subroutines, with the important restriction that the environment only provides inputs to and receives outputs from the parties of a single instance of π. In particular, it does not directly communicate with any subroutine of a party of that single instance.

Ideal functionalities and ideal protocols. Security of protocols is defined via comparing the protocol execution to an *ideal process* for carrying out the task at hand. For convenience of presentation, we formulate the ideal process for a task as a special protocol within the above model of protocol execution. (This avoids formulating an ideal process from scratch.) A key ingredient in this special protocol, called the ideal protocol, is an ideal functionality that captures the desired functionality, or the specification, of the task by way of a set of instructions for a "trusted party".

That is, let $\mathcal{F}$ be an ideal functionality (i.e., an algorithm for the trusted party). Then an instance of the ideal protocol $\text{IDEAL}_{\mathcal{F}}$ consists of dummy parties, plus a party $\mathcal{F}$ that's a subroutine of all the main parties. Upon receiving an input v, each dummy party forwards v as input to the subroutine $\mathcal{F}$. Any subroutine output coming from $\mathcal{F}$ is forwarded by the dummy party as subroutine output for the environment. We note that $\mathcal{F}$ can model reactive computation, in the sense that it can maintain local state and its outputs may depend on all the inputs received and all random choices so far. In addition, $\mathcal{F}$ may receive messages directly from the adversary $\mathcal{A}$, and may contain instructions to send messages to $\mathcal{A}$. This "back-door channel" of direct communication between $\mathcal{F}$ and $\mathcal{A}$ provides a way to *relax* the security guarantees provided $\mathcal{F}$. Specifically, by letting $\mathcal{F}$ take into account information received from $\mathcal{A}$, it is possible to capture the "allowed influence" of the adversary on the outputs of the parties, in terms of both contents and timing. By letting $\mathcal{F}$ provide information directly to $\mathcal{A}$ it is possible to capture the "allowed leakage" of information on the inputs and outputs of the parties.

Protocol emulation. It remains to define what it means for a protocol to "mimic" or "emulate" the ideal process for some task. As a step towards this goal, we first formulate a more general notion of emulation, which applies to any two protocols. Informally, protocol π emulates protocol ϕ if, from the point of view of any environment, protocol π is "just as good" as ϕ, in the sense that no environment can tell whether it is interacting with π and some (known) adversary, or with ϕ and some other adversary. More precisely:

Definition (protocol emulation): *Protocol π* UC-emulates *protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that, for any environment $\mathcal{Z}$ the ensembles $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ are indistinguishable. That is, on any input, the probability that $\mathcal{Z}$ outputs 1 after interacting with $\mathcal{A}$ and parties running π differs by at most a negligible amount from the probability that $\mathcal{Z}$ outputs 1 after interacting with $\mathcal{S}$ and ϕ.*

Once the general notion of protocol emulation is defined, the notion of realizing an ideal functionality is immediate:

Definition (realizing functionalities): *Protocol π* UC-realizes *an ideal functionality $\mathcal{F}$ if π emulates $\text{IDEAL}_{\mathcal{F}}$, the ideal protocol for $\mathcal{F}$.*

2.1 The composition theorem

As in the case of protocol emulation, we present the composition operation and theorem in the more general context of composing two arbitrary protocols. The case of composing ideal protocols follows as a special case.

The universal composition operation. The universal composition operation is a natural generalization of the "subroutine substitution" operation for sequential algorithms to the case of distributed protocols. That is, let ρ be a protocol that contains instructions to call protocol protocol ϕ as a subroutine, and let π be a protocol that UC-emulates ϕ. The composed protocol, denoted $\rho^{\pi/\phi}$, is the protocol that is identical to ρ, except that each instruction to call protocol ϕ is replaced with an instruction to call protocol π *with the same parameters an inputs.* Similarly, any output from a party running π is treated as an input form a party running ϕ. In particular, if some party running ρ calls multiple instances of ϕ, differentiated via their session IDs, then the corresponding instance of $\pi^{\rho/\phi}$ will use multiple instances of ρ.

The composition theorem. In its general form, the composition theorem says that if protocol π UC-emulates protocol ϕ then, for any protocol ρ, the composed protocol $\rho^{\pi/\phi}$ emulates ρ. This can be interpreted as asserting that replacing calls to ϕ with calls to π does not affect the behavior of ρ in any distinguishable way.

There is one caveat: For this result to hold we need that protocols π and ρ are "nice" in that only the main parties of the protocol have I/O with the outside world. More precisely, say that a protocol π is *subroutine respecting* if only the main parties of any instance of π receive input from external parties and send output to external parties. In particular, subroutines of the main parties, and subroutines thereof, do not directly get inputs from or send outputs to an external party. Then:

Theorem (universal composition): *Let ρ, ϕ, π be subroutine respecting protocols such that ρ uses ϕ as subroutine and π UC-emulates ϕ. Then protocol $\rho^{\pi/\phi}$ UC-emulates ρ. In particular, if ρ UC-realizes an ideal functionality $\mathcal{G}$ then so does $\rho^{\pi/\phi}$.*

A first, immediate corollary of the general theorem states that if protocol π UC-realizes an ideal functionality $\mathcal{F}$, and π uses as subroutine protocol $\textsc{ideal}_{\mathcal{F}}$, the ideal protocol for $\mathcal{F}$, then the composed protocol $\rho^{\pi/\phi^{\mathcal{F}}}$ UC-emulates ρ.[1] Another corollary states that if π UC-realizes an ideal functionality $\mathcal{G}$, then so does $\rho^{\pi/\phi}$.

Remark: On the universality of universal composition. Many different ways of "composing together" protocols into larger systems are considered in the literature. Examples include sequential, parallel, and concurrent composition, of varying number of protocol instances, where the composed instances are run either by the same set of parties or by different sets of parties, use either the same program or different programs, and have either the same input or different inputs. A more detailed discussion appears in [Can06].

We observe that all of these composition methods can be captured as special cases of universal composition. That is, any such method for composing together protocol instances can be captured by an appropriate "calling protocol" ρ that uses the appropriate number of protocol instances as subroutines, provides them with appropriately chosen inputs, and arranges for the appropriate synchronization in message delivery among the various subroutine instances. Consequently, it is guaranteed that a protocol that UC-realizes an ideal functionality $\mathcal{F}$ continues to UC-realize $\mathcal{F}$ even when composed with other protocols using any of the composition operations considered in the literature.

2.2 Generalized UC security

In the UC framework the UC theorem holds only for protocols which are subroutine respecting. This simplifies the model and the analysis of protocols within it, but it does not allow to guarantee security in interesting cases where the same computational entity is used as a subroutine within multiple protocol instances.

To get around this limitation, the generalized UC (GUC) framework [CDPW07] modifies the model of protocol execution by allowing the environment to create and interact with other entities, in addition to the adversary and the parties of a single instance of the analyzed protocol, π. These additional entities may in

[1] We say that an instance of protocol ρ uses an instance of protocol ϕ as a subroutine if each party in the instance of ϕ is a subroutine of some party of the instances of ρ.

turn provide inputs to and get outputs from participants in π. Say that protocol π GUC-emulates protocol ϕ if π UC-emulates ϕ with the modified protocol execution model. Now it can be seen that, within the GUC framework, the UC theorem holds even with respect to protocols that are not subroutine respecting:

Theorem (generalized universal composition): *Let ρ, ϕ, π be protocols such that ρ uses ϕ as subroutine and π GUC-emulates ϕ. Then protocol $\rho^{\pi/\phi}$ UC-emulates ρ. In particular, if ρ GUC-realizes an ideal functionality $\mathcal{G}$ then so does $\rho^{\pi/\phi}$.*

Two results surveyed here use this generalized model, for different purposes. One is the modeling of the augmented CRS model in [CDPW07], with the purpose of modeling set-up that's available to more than one protocol instance. The other is the modeling of adversarially controlled reference strings in [CPS07].

3 Prologue: Impossibility of UC commitment

We recall some basic results regarding realizability of functionalities in the UC framework. These results motivate and shape the search for better set-up assumptions.

In a nutshell, the natural formulations of Commitment, Zero-Knowledge, Coin Tossing, or Oblivious Transfer as ideal functionalities within the UC framework turn out to be "complete" for UC realizability. That is, UC-realizing any one of these functionalities is necessary and sufficient for obtaining general realizability results for practically any ideal functionality.

In other words, there exist ideal functionalities, $\mathcal{F}_{\text{COM}}$, $\mathcal{F}_{\text{ZK}}$, $\mathcal{F}_{\text{COIN}}, \mathcal{F}_{\text{OT}}$, that naturally capture the security requirements from the corresponding primitives, and such that it is possible to UC-realize any one of these ideal functionalities by protocols that make use of any one of these ideal functionalities as a subroutine (see [C01] for more details). Furthermore, there exist constructions for UC-realizing any "well-formed" ideal functionality via protocols that use, say, $\mathcal{F}_{\text{ZK}}$ as subroutine (see, e.g. [CLOS02]).

Furthermore, it is impossible to UC-realize any one of these functionalities via two-party plain protocols.

Here we briefly recall the impossibility result for UC-realizing the ideal commitment functionality, $\mathcal{F}_{\text{COM}}$. Impossibility for the other primitives follow similar lines. First, however, let us recall the formulation of $\mathcal{F}_{\text{COM}}$.

The ideal commitment functionality. The ideal commitment functionality, $\mathcal{F}_{\text{COM}}$, formalizes the "sealed envelope" intuition in a straightforward way. That is, when receiving from the committer C an input requesting to commit to value x to a receiver R, $\mathcal{F}_{\text{COM}}$ records (x, R) and notifies R and the adversary that C has committed to some value. (Notifying the adversary means that the fact that a commitment took place need not be hidden.) The opening phase is initiated by the committer inputting a request to open the recorded value. In response, $\mathcal{F}_{\text{COM}}$ outputs x to R and the adversary. (Giving x to the adversary means that the opened value can be publicly available.)

In order to correctly handle adaptive corruption of the committer during the course of the execution, $\mathcal{F}_{\text{COM}}$ responds to a request by the adversary to corrupt C by first outputting a corruption output to C, and then revealing the recorded value x to the adversary. In addition, if the `Receipt` value was not yet delivered to R, then $\mathcal{F}_{\text{COM}}$ allows the adversary to modify the committed value. This last stipulation captures the fact that the committed value is fixed only at the end of the commit phase, thus if the committer is corrupted during that phase then the adversary might still be able to modify the committed value. (Corruption of the receiver does not require any move.)

$\mathcal{F}_{\text{COM}}$ is described in Figure 1. For brevity, we use the following terminology: The instruction "send a delayed output x to party P" should be interpreted as "send (x, P) to the adversary; when receiving `ok` from the adversary, output x to P."

Functionality $\mathcal{F}_{\text{COM}}$

1. Upon receiving an input $(\texttt{Commit}, x)$ from party C, record (C, R, x) and generate a delayed output $(\texttt{Receipt})$ to R. Ignore any subsequent $(\texttt{Commit}...)$ inputs.
2. Upon receiving an input $(\texttt{Open})$ from C, do: If there is a recorded value x then generate a delayed output $(\texttt{Open}, x)$ to R. Otherwise, do nothing.
3. Upon receiving a message $(\texttt{Corrupt}, C)$ from the adversary, output a `Corrupted` value to C, and send x to the adversary. Furthermore, if the adversary now provides a value x', and the $(\texttt{Receipt})$ output was not yet written on R's tape, then change the recorded value to x'.

Fig. 1. The Ideal Commitment functionality, $\mathcal{F}_{\text{COM}}$

Impossibility of realizing $\mathcal{F}_{\text{COM}}$ in the plain model. Roughly speaking, the requirements from a protocol that UC-realizes $\mathcal{F}_{\text{COM}}$ boil down to the following two requirements from the ideal-process adversary (simulator) $\mathcal{S}$. *(a)*. When the committer is corrupted (i.e., controlled by the adversary), $\mathcal{S}$ must be able to "extract" the committed value from the commitment. That is, $\mathcal{S}$ has to come up with a value x such that the committer will almost never be able to successfully decommit to any $x' \neq x$. This is so since in the ideal process $\mathcal{S}$ has to explicitly provide $\mathcal{F}_{\text{COM}}$ with a committed value. *(b)*. When the receiver is uncorrupted, $\mathcal{S}$ has to be able to generate a "simulated commitment" c that looks like a real commitment and yet can be "opened" to any value, to be determined at the time of opening. This is so since $\mathcal{S}$ has to provide adversary $\mathcal{A}$ and environment $\mathcal{Z}$ with the simulated commitment c before the value committed to is known. All this needs to be done *without rewinding the environment $\mathcal{Z}$*.

Intuitively, these requirements look impossible to meet: A simulator that has the above abilities can be used by a dishonest receiver to "extract" the committed value from an honest committer. This intuition can indeed be formalized to show that in the plain model it is impossible to UC-realize $\mathcal{F}_{\text{COM}}$ by a two-party protocol. Essentially, the proof proceeds as follows. Let π be a protocol that UC-realizes $\mathcal{F}_{\text{COM}}$. Consider an execution of π by an adversarially controlled committer C and an honest receiver R, and assume that the adversary merely sends messages that are generated by the environment, and delivers to the environment any message received from R. The environment, $\mathcal{Z}_C$, secretly picks a random bit b at the beginning and generates the messages for C by running the protocol of the honest committer for b and R's answers. Once $\mathcal{Z}_C$ received a "receipt" output from R, it starts running the honest opening protocol in the name of C. Finally, $\mathcal{Z}_C$ outputs 1 iff the b' that R outputs equals the secret bit b. We know that the in an execution of π with honest receiver and committer, in the opening stage the receiver always outputs the bit committed to by the committer. However, since π UC-realizes $\mathcal{F}_{\text{COM}}$, there should exist an ideal-model adversary $\mathcal{S}$ that interacts with $\mathcal{F}_{\text{COM}}$ and generates a view for $\mathcal{Z}_C$ that is indistinguishable from a real execution with π. In particular, it must also be the case that $b = b'$ almost always even in the ideal process. For this to hold, it must be that $\mathcal{S}$ must have given to $\mathcal{F}_{\text{COM}}$ the correct bit b at the commitment stage. Now, given $\mathcal{S}$, we can construct another environment, $\mathcal{Z}_R$, and a corrupted receiver $\mathcal{A}_R$, such that $\mathcal{Z}_R$ successfully distinguishes between an execution of π and an interaction with $\mathcal{F}_{\text{COM}}$ and *any* adversary $\mathcal{S}_R$. $\mathcal{Z}_R$ and $\mathcal{A}_R$ proceed as follows: $\mathcal{Z}_R$ chooses a random bit b and hands b as input to the honest committer C. It then waits to receive a bit b' from $\mathcal{A}_R$ (which controls the receiver). $\mathcal{Z}_R$ outputs 1 iff $b = b'$. $\mathcal{A}_R$ proceeds as follows: Recall that $\mathcal{S}$ can "extract" the committed bit b via simple interaction with the committers messages, without rewinding or any additional information. Therefore, $\mathcal{A}_R$ can simply run $\mathcal{S}$ and guess b almost always. In contrast, when $\mathcal{Z}_R$ interacts with $\mathcal{F}_{\text{COM}}$, the adversary's view is independent of b, and thus $b = b'$ with probability exactly one half.

4 The Common Reference String model

The common reference string model, first proposed in [BFM88] and used extensively since, assumes that the parties have access to a common string that is guaranteed to come from a pre-specified distribution. Furthermore, it is guaranteed that the string was chosen in an "opaque" way, namely that no information related to the process of choosing this string is available to any party. A very natural distribution for the common string, advocated in [BFM88], is the uniform distribution over the strings of some length. Still, it is often useful to consider reference strings that are taken from other distributions.

In the Zero-Knowledge context of [BFM88], the fact that the reference string comes from an external source that is unrelated to the actual computation is captured by allowing the simulator to choose the reference string as it wishes — as long as the adversary cannot distinguish this "simulated string" from a "real string" taken from the prescribed distribution. Indeed, it is this extra freedom given to the simulator which makes this model powerful.

Within the present framework, the CRS model can be captured in a natural way by modeling the reference string as coming from an appropriate ideal functionality. More specifically, we formulate functionality $\mathcal{F}_{\text{CRS}}$, presented in Figure 2 below. The functionality is parameterized by a distribution D and a set $\mathcal{P}$ of recipients of the reference string. Upon invocation, it first draws a value r from distribution D. Next, on input from a party $P \in \mathcal{P}$, $\mathcal{F}_{\text{CRS}}^{D}$ returns r to P.

Letting the adversary know r models the fact that r is public, and cannot be assumed secret. Prohibiting parties not in $\mathcal{P}$ from obtaining r directly from $\mathcal{F}_{\text{CRS}}$ models the fact that r is treated as local to a specific protocol instance, and is intended to be used only within this protocol instance. (This point is elaborated on in Section 7.) Other protocol instances should use different "draws" from distribution D. This restriction on the use of the reference string limits the applicability of the CRS model: To realize $\mathcal{F}_{\text{CRS}}$ in reality, the participants of each protocol execution need to somehow "get together" and obtain a reference string that they all trust to be taken from the specified distribution. The next sections discuss set-up assumptions that are aimed at mitigating this limitations in a number of different ways.

Functionality $\mathcal{F}_{\text{CRS}}^{D,\mathcal{P}}$

1. When receiving input ($\texttt{CRS}$,sid) from party P, first verify that $P \in \mathcal{P}$; else ignore the input. Next, if there is no value r recorded then choose and record a value $r \stackrel{\text{R}}{\leftarrow} D$. Finally, send a public delayed output $(\texttt{CRS}, sid, r)$ to P.

Fig. 2. The Common Reference String functionality

From $\mathcal{F}_{\text{CRS}}$ to $\mathcal{F}_{\text{COM}}$. Several protocols that UC-realize $\mathcal{F}_{\text{COM}}$ given access to $\mathcal{F}_{\text{CRS}}$ are known. Here we briefly sketch the protocol of [CF01]. What "saves" the simulator in the CRS model from the above impossibility result is the following observation, which parallels the original CRS model of [BFM88]: When interacting with a commitment protocol that used $\mathcal{F}_{\text{CRS}}$, the environment learns about the value of the reference string only from the adversary. This means that, the ideal process for $\mathcal{F}_{\text{COM}}$, the simulator can choose the reference string on its own. Consequently, the simulator can know some "trapdoor information" associated with the reference string, and even change its distribution slightly.

The [CF01] commitment protocol uses this observation as follows. The reference string will consist of a public key e of an encryption scheme and a claw-free pair of permutations f_0, f_1 with trapdoor. (That is, given only the description f_0, f_1 it is infeasible to find x_0, x_1 such that $f_0(x_0) = f_1(x_1)$, but given a trapdoor t one can efficiently invert, say, f_0.) Now, to commit to bit b, the committer chooses a random value r and sends the commitment message $(f_b(r), E_e(r_0, id), E_e(r_1, id))$ where $r_b = r$, $r_{1-b} = 0$, and id is an identifier for the session. (Typically, id would include the identities of the committer and receiver, plus additional commitment-specific information.) To open to bit b, the committer sends r and the randomness used for encrypting r; this is the first or second encryption, depending on b.[2] Now, in a standard execution of the protocol the commitment is committing (due to the claw-freeness of f_0, f_1), and hiding (due to the security of the encryption scheme). However, in a simulated execution the simulator can know both t and the decryption key associated with e. It can thus easily generate commitment strings that can be opened both ways, and at the same time it can easily extract the hidden value committed in an honestly generated commitment. When the encryption scheme is secure against chosen ciphertext attacks, it can be shown that the simulator can successfully extract the hidden value even when the commitment string is chosen adversarially. This ideas are at the basis of the proof of security of the protocol.

We note that the above protocol can generate multiple commitments using a single reference string. In other words, it actually realizes a "multi-session version" of $\mathcal{F}_{\text{COM}}$, where a single instance allows multiple parties can commit and open multiple commitments. (This multi-session version is called $\mathcal{F}_{\text{MCOM}}$ in the literature.) This somewhat alleviates the need to agree on a different reference string for each protocol instance, since a single instance of the above protocol suffices for generating commitments for an entire system. However, the solution is far from satisfying: First, strictly speaking, all protocol instances that use the same commitment protocol now have some joint state and can no longer be analyzed separately and be composed later. Second, no security guarantee is given with respect to other protocols that use the same reference string in other ways than via that global instance of the commitment protocol. The first issue is handled by the Universal Composition with Joint State (JUC) theorem of [CR03]. The second issue is more subtle and is addressed in Section 7.

5 The Key Set-Up model

The CRS set-up assumption has the advantage that it only requires knowledge of a single short string. In particular, it does not require parties to identify themselves or to go through a registration process before participating in a protocol. Thus, in settings where it is reasonable to assume existence of trusted reference string, this assumption is very attractive. However, when the reference string is being generated by a computational entity that may be corrupted or subverted, the CRS modeling is somewhat unsatisfactory, in that it puts complete trust in a single entity. In fact, this entity, if subverted, can completely undermine the security of the protocol by choosing the reference string from a different distribution, or alternatively by leaking to some parties some secret information related to the string. Furthermore, it can do so without being detected.

The key set-up functionality, $\mathcal{F}_{\text{KS}}$, formulated in [BCNP04] and presented in Figure 3, is written in a way that can be realized by real-world mechanisms

[2] The actual protocol is slightly different, to account for adaptive corruptions.

that do not require all participants to put full trust in a single string. At the same time, it can be realized even in the CRS model itself. We first describe the functionality and its use, and then discuss how it can be realized.

$\mathcal{F}_{\text{KS}}$ is parameterized by a set $\mathcal{P}$ of parties and 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". However, $\mathcal{F}_{\text{KS}}$ provides only relatively weak guarantees regarding this public key, giving the adversary considerable freedom in determining this key. (This freedom is what makes $\mathcal{F}_{\text{KS}}$ so relaxed.) Specifically, the "public key" to be associated with a party upon registration is determined as follows. The functionality keeps a set R of "good public keys". Upon receiving a registration request from party $P \in \mathcal{P}$, 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, p) and returns p to P 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 the adversary can also specify, if it chooses, an arbitrary "secret key" r. In this case, P is registered with the value $f(r)$ (but r is not added to R).

A retrieval request, made by a party in $\mathcal{P}$, for the public key of party P is answered with either an error message $\perp$ or one of the registered public keys of P, 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.)

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, $\mathcal{F}_{\text{KS}}$ 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 functionality received 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, $\mathcal{F}_{\text{KS}}$ 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 received the corresponding private keys.

In [BCNP04] it is shown how to UC-realize $\mathcal{F}_{\text{MCOM}}$ given access to $\mathcal{F}_{\text{KS}}$. A non-interactive protocol for realizing $\mathcal{F}_{\text{ZK}}$ given access to KS is also shown. The protocol for realizing $\mathcal{F}_{\text{MCOM}}$ is essentially identical to the [CF01] protocol described above; the only difference is that the claw-free pair f_0, f_1 is now the public key of the receiver, whereas the encryption key e is now the public key of the committer. Intuitively, this works since the committer is only concerned that the secret

Functionality $\mathcal{F}_{\mathrm{KS}}^{\mathcal{P},f}$

$\mathcal{F}_{\mathrm{KS}}^{f}$ proceeds as follows, given set $\mathcal{P}$ of identities, function f and security parameter k. At the first activation a set R of strings is initialized to be empty.

Registration: When receiving input $(\texttt{Register}, sid)$ from a party P, verify that that $P \in \mathcal{P}$; else ignore the input. Next, send $(\texttt{Register}, sid, P)$ to the adversary, and receive a value p' from the adversary. 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, p) and return (sid, p) to P and to the adversary.

Registration by a corrupted party: When receiving input $(\texttt{Register}, sid, r)$ from a corrupted party $P \in \mathcal{P}$, record $(P, f(r))$. In this case, $f(r)$ is *not* added to R.

Retrieval: When receiving a message $(\texttt{Retrieve}, sid, P)$ from party $P' \in \mathcal{P}$, send $(\texttt{Retrieve}, sid, P, P')$ to the adversary and obtain a value p in return. If (P, p) is recorded then return (sid, P, p) to P'. Else, return $(sid, P, \perp)$ to P'.

Fig. 3. The Key Registration functionality

decryption key associated with e remains unknown, whereas the receiver is only concerned that the trapdoor t of f_0, f_1 remains unknown. We note, however, that this protocol remains secure only for non-adaptive party corruption.

Realizing $\mathcal{F}_{\mathrm{KS}}$. $\mathcal{F}_{\mathrm{KS}}$ can be realized in a number of different ways. First, we observe that $\mathcal{F}_{\mathrm{KS}}^{\mathcal{P},f}$ can be realized in the $\mathcal{F}_{\mathrm{CRS}}^{\mathcal{P},D}$-hybrid model, where $D = D_k$ is the distribution of $f(r)$ for r that is uniform in $\{0,1\}^k$. The protocol is straightforward: On input either $(\texttt{Register}, sid)$ or $(\texttt{Retrieve}, sid, P)$, party P sends $(\texttt{CRS}, sid)$ to $\mathcal{F}_{\mathrm{CRS}}$ and returns the obtained value.

Realizing $\mathcal{F}_{\mathrm{KS}}$ given a distributed registration service. Consider a setting where the parties have access to registration servers where parties can register and obtain public keys that were chosen at random according to a given distribution (i.e., the public key is $f(r)$ for an $r \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}^k$). Alternatively, parties can choose their keys themselves and provide them to the server. Note that here each party needs to put full trust (to keep its key secret) only in the server it registers with. The trust put in other servers is much lower - it only needs to be trusted that the public keys obtained from these servers are "well formed".

Realizing $\mathcal{F}_{\mathrm{KS}}$ using traditional proofs of knowledge. Finally, it is possible to realize $\mathcal{F}_{\mathrm{KRK}}^{f}$ (and thus also $\mathcal{F}_{\mathrm{KS}}^{f}$) via traditional (non-UC) proofs of knowledge of the private key, under the assumption that the proofs of knowledge occur when there is no related network activity. (Intuitively, in this case it is ok to "rewind" the environment, as necessary to prove security of the traditional proof of knowledge.

6 Adversarially controlled set-up

The common reference string model provides the guarantee that the reference string is drawn from a pre-specified distribution. This is a very convenient abstraction for the purpose of designing protocols. Indeed, all existing protocols use this guarantee in a crucial sense: Security analyses quickly fall apart as soon as distribution of the reference string is changed even slightly.

This property is quite limiting. In particular, it seems to rule out "physical implementations" where the reference string is taken to be the result of joint measurement of some physical phenomenon such as astronomic measurements, fluctuations of the stock market, or even network delays across the Internet. Indeed, while it is reasonable to believe that such phenomena are largely unpredictable and uncontrollable, namely they have "high entropy", it is a stretch of the imagination to believe that they are taken from a distribution that is known to and useful for the protocol designer.

Can composable security be obtained if we only have an *imperfect* reference strings, or alternatively a reference string that are adversarially controlled to some extent? More specifically, are there protocols that UC-realize, say, $\mathcal{F}_{\text{COM}}$ in such a setting?

A first indication that this might not be an easy task is the result of Dodis et. al. [DOPS04] that demonstrates the impossibility of NIZK in a relaxed variant of the CRS model in which the distribution of the reference string can be arbitrary subject to having some minimal min-entropy. However, this result does not rule rule out composable protocols; more importantly, it does not consider the case where the reference string is guaranteed to be taken from an efficiently samplable distribution. Indeed, for such distributions deterministic extractors are known to exist (under computational assumptions) [TV00]. Thus, one might expect it to be possible to "compile" any protocol in the CRS model (or at least protocols that can do with a uniformly distributed reference string) into a protocol that uses a reference string that is taken from any efficiently samplable distribution that has sufficient min-entropy: First have the parties use a deterministic extractor to transform the reference string into a string that is almost uniformly distributed. Next, run the original protocol. Since the extracted string is almost uniform, one might expect the original analysis to work in the same way.

However, deterministic extractability turns out to be insufficient for this purpose. In fact, it turns out that if one relaxes $\mathcal{F}_{\text{CRS}}$ so as to allow the distribution to be adversarially determined, then UC-realizing $\mathcal{F}_{\text{COM}}$ becomes impossible [CPS07]. Impossibility holds even if the chosen distribution is guaranteed to have full min-entropy minus a polynomially vanishing fraction, even if the distribution is guaranteed to be sampled via an *algorithmic* process, namely via a sampling process that has a relatively succinct description, and even when this process is guaranteed to be computationally efficient.

As a recourse, one may restrict attention to the case where the algorithm for sampling the reference string is known to the adversaries involved. (Still, it is of course unknown to the protocol.) Here it turns out to be possible to UC-realize

$\mathcal{F}_{\text{COM}}$, as long as the reference string is taken from a distribution that is guaranteed to have a polynomial time sampling algorithm, a short description, and super-logarithmic min-entropy. Furthermore, all three conditions are simultaneously necessary, in the sense that impossibility holds as soon as any one of the conditions is relaxed [CPS07].

Discussion. It may appear over-optimistic to assume that the physical (or man-made) phenomena used to generate the reference string are governed by distributions where the sampling algorithm is computable in polynomial time. Indeed, why should Nature be governed by succinct and efficient algorithms? However, beyond the technical fact that these restrictions are necessary, one can view our analysis as a proof that any successful attack against the proposed protocols demonstrates that either the underlying hardness assumptions are violated, *or else that the process for choosing the reference string is not efficiently computable, or has long description.* This might be an interesting revelation in itself. Alternatively, the positive result may be interpreted as addressing situations where the process of choosing the reference string is influenced by an actual attacker. Here the guarantee that the distribution has some min-entropy represents the fact that the attacker's influence on the sampling process is limited.

The [CPS07] **results in more detail.** Three relaxations of $\mathcal{F}_{\text{CRS}}^{\mathcal{P},D}$ are formulated. The first relaxation, called $\mathcal{F}_{\text{BBSUN}}$, proceeds as follows. (Here SUN stands for "sunspots", which is the term used in the first works that propose the CRS model when referring to astronomic observations [BFM88,F88] and BB stands for "black-box"). Instead of treating the distribution D as a fixed, public parameter, let the environment determine the distribution by providing a description of a sampling algorithm for D. Then, $\mathcal{F}_{\text{BBSUN}}$ chooses a sufficiently long random string ρ and computes the reference string $r = D(\rho)$. In addition, $\mathcal{F}_{\text{BBSUN}}$ lets the adversary (and simulator) obtain additional independent samples from the distribution "on the side". These samples are not seen by the environment or the parties running the protocol.

Three parameters of $\mathcal{F}_{\text{BBSUN}}$ turn out to be salient. First is the min-entropy, or "amount of randomness" of the reference string (measured over the random choices of both the environment and the sunspot functionality). Next is the runtime, or computational complexity of the sampling algorithm D. Last is the description-size of D (namely, the number of bits in its representation as a string); this quantity essentially measures the amount of randomness in the reference string that comes from the random choices of the environment. All quantities are measured as a function of the length n of the reference string; that is, we treat n as the security parameter.

Theorem: There exist no two-party protocols that UC-realize $\mathcal{F}_{\text{COM}}$ when given access to of $\mathcal{F}_{\text{BBSUN}}$. This holds even if the distribution of the reference string is guaranteed to have min-entropy greater than $n - n^{\epsilon}$, and even if both the description size and the computational complexity of the provided sampling algorithm are guaranteed to be at most n^{ϵ}, for any $\epsilon > 0$.

Next a more restricted setting is considered, where the adversary has access to the "code", or description of the sampling algorithm D. This is modeled by having the set-up functionality explicitly send the description of D to the adversary. (Note that this relaxation is meaningful only for sampling algorithms that can be described in $poly(n)$ bits, else the adversary cannot read the description.) Call this functionality $\mathcal{F}_{\text{GBSUN}}$ (for “gray box”). The third variant, called $\mathcal{F}_{\text{SUN}}$, gives to the adversary also the local random choices used to generate the reference string. It turns out that this variant provides an incomparable setup guarantee to that of $\mathcal{F}_{\text{GBSUN}}$. (This is so since the setup functionality is invoked directly by the environment. Consequently, the functionality exists both in the real-life and in the ideal models.)

Theorem: There exist no two-party protocols that UC-realize $\mathcal{F}_{\text{COM}}$ when given access to either $\mathcal{F}_{\text{GBSUN}}$ or $\mathcal{F}_{\text{SUN}}$. This holds even if either one of the following holds

1. The computational complexity of the sampling algorithm can be super-polynomial in n, as long as the distribution of the reference string is guaranteed to have min-entropy $n - \text{poly} \log n$, and the description size of the provided sampling algorithm is guaranteed to be at most $\text{poly} \log n$ (assuming one-way functions with sub-exponential hardness).
2. The description size of the sampling algorithm is at least $\mu(n) - \log n$, as long as the distribution of the reference string is guaranteed to have min-entropy $\mu(n) = n$ and the computational complexity is guaranteed to be at most $O(n)$.
3. The distribution of the reference string has min-entropy at most $\log n$, as long as the description length is $O(1)$ and the computational complexity is $O(n)$.

On the other hand, we have:

Theorem: Assume there exist collision-resistant hash functions, dense cryptosystems and one-way functions with sub-exponential hardness. Then there exists a two-party protocol that UC-realizes $\mathcal{F}_{\text{MCOM}}$, when given access to $O(1)$ instances of either $\mathcal{F}_{\text{GBSUN}}$ or $\mathcal{F}_{\text{SUN}}$, as long as it is guaranteed that the min-entropy of the reference string is at least $\mu(n) = \text{poly} \log n$ the computational complexity of the provided sampling algorithm is at most $poly(n)$ and its description size is at most $\mu(n) - \text{poly} \log n$.

Furthermore, the protocol from Theorem 3 withstands even adaptive party corruptions, with no data erasure, whereas Theorems 1 and 2 apply even to protocols that only withstand static corruptions.

In other words, under computational assumptions, Theorem 2 and 3 provide an essentially tight characterization of the feasibility of UC protocols, in terms of the min-entropy, computational complexity and description length of the reference string. Informally,

> *UC-security of non-trivial tasks is possible if and only if the reference string has min-entropy at least* $\mu(n) = \text{poly} \log n$, *and is generated by a computationally-efficient sampling algorithm with description length at most* $\mu(n) - \text{poly} \log n$.

Techniques for the impossibility results. The impossibility results combine the [CF01] proof of impossibility of UC-realizing $\mathcal{F}_{\text{COM}}$ in the plain model with techniques from [GK89]. Recall that the model does not let the environment see the reference string directly, which in principle allows the simulator to present the environment with any string of its choosing and claim that this is the reference string chosen in the execution. To mitigate this freedom, the environment chooses a special distribution D that makes sure that the string presented by the simulator as the actual reference string can only be one of the strings that the simulator received as "extra samples" from the set-up functionality. Since the simulator can only ask for a polynomial number of such samples, it can be seen that a dishonest verifier can still use the simulator to extract the committed bit from an honest committer, much as in the proof of [CF01], and with only polynomial degradation in success probability. All impossibility results use this idea, with different techniques or choosing the distribution D so as to obtain the desired effect.

Protocol techniques. To explain the main idea behind the protocol, it is useful to first sketch a simpler protocol that is only secure with respect to static corruptions. Also, the protocol aims to realize the zero-knowledge functionality, $\mathcal{F}_{\text{ZK}}$, rather than $\mathcal{F}_{\text{MCOM}}$. The idea is to use a variation on Barak's protocol [B01]: Let L be an NP language and assume that a prover P wishes to prove to a verifier V that $x \in L$, having access to a reference string r that is taken from an unknown distribution with min-entropy at least $\mu = n^{\epsilon}$. Then, P and V will engage in a witness-indistinguishable proof that *"either $x \in L$ or the reference string r has a description of size $\mu/2$"*. (As in Barak's protocol, the description size is measured in terms of the Kolmogorov complexity, namely existence of a Turing machine M with description size $\mu/2$ that outputs r on empty input. Also, in order to guarantee that the protocol is simulatable in polynomial-time M should be polynomial time.) Soundness holds because in a real execution of the protocol, r is taken from a distribution with min-entropy at least μ, so the second part of the "or" statement is false with high probability. To demonstrate zero-knowledge, the simulator generates a simulated reference string $\tilde{r}$ by running the sampling algorithm D for the distribution on a *pseudo*random random-input. That is, the simulator chooses a random string $\tilde{\rho}$ of length, say, $\mu/2 - |D|$ (where $|D|$ denotes the description size of D) and computes $\tilde{r} = D(G(\tilde{\rho}))$, where G is some length-tripling pseudo-random generator. Now, $\tilde{r}$ indeed has description of size $\mu/2$ (namely, $\tilde{\rho}$ plus $|D|$ plus the constant-size description of G); furthermore, the simulator knows this description. Also, since both D and the environment are polynomial time, the simulated string $\tilde{r}$ is indistinguishable from the real string r.

The above protocol allows for straight-line simulation. It is not yet straight-line extractable, but it can be modified to be so using the techniques of [BL04]. Still, it is only secure against *static* corruptions of parties. In order to come up with a protocol that withstands *adaptive* corruptions a somewhat different technique is used, which combines the above idea with techniques from [CDPW07]. First, they move to realizing $\mathcal{F}_{\text{MCOM}}$. They then proceed in several steps: The first step is to construct a commitment scheme that is *equivocal* and adaptively secure. This is done using Feige and Shamir's technique [FS89] for constructing equivocal commitments from Zero-Knowledge protocols such as the one described above. Next, the constructed equivocal commitment scheme is used in a special type of a coin-tossing protocol, and use the obtained coin tosses as a reference string for a standard UC commitment protocol such as [CF01].

The protocol allows *two* parties to perform *multiple* commitment and decommitment operations between them, using only *two* reference strings —one for the commitments by each party. This means that in a multi-party setting it is possible to realize any ideal functionality using one reference string for each (ordered) pair of parties, regardless of the number of commitments and decommitment performed. Furthermore, each reference string needs to be trusted only by the two parties who use it.

7 Globally available set-up

All the set-up models considered so far model the set-up information as information that's available only to the participants of a single protocol instance. This means that, in order to implement such a model, one has to generate a fresh reference string (or fresh public keys) for each instance of a protocol that uses it. Furthermore, this has to be done in a way that makes the reference string available only to the protocol participants. While such implementations are possible (say, via joint measurements of physical phenomena at the onset of an execution), this is a severe limitation. In particular, this modeling stands in contrast with the prevalent intuitive perception of the reference string (or public key infrastructure) as a "global" construct that is chosen in advance and made available to all throughout the lifetime of the system.

Furthermore, this limitation turns out to be not only "academic". For instance, all existing protocols designed in the CRS model turn out to be *insecure* in a setting where the reference string can be used by multiple, arbitrary protocols. In fact, as shown in [CDPW07], this limitation is inherent: No set-up assumption that only gives out public set-up information can suffice for realizing, say, $\mathcal{F}_{\text{COM}}$, if the same set-up information can be used by all protocols in the system.

To exemplify this point, consider the "non-transferability" (or, "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 set-up, or alternatively set-up that's local to each protocol instance, this intuition works, in the sense that UC-realizing such a trusted party automatically implies non-transferability. However, when a global set-up is used, this is no longer the case: There are protocols that emulate such a trusted party but do *not* guarantee non-transferability.

For instance, consider the case of Zero-Knowledge protocols, namely protocol that emulate the trusted party for the "Zero-Knowledge functionality": Zero-Knowledge protocols in the plain model are inherently deniable, but all existing Zero-Knowledge protocols in the CRS model are completely *un*deniable whenever the reference string is public knowledge (see [P03]).

Non-transferability is not the only concern that remains un-captured in the present formulation of security in the CRS model. For instance, the protocol in [CF01] 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. Other demonstrations of this point are given in [YYZ07A].

The global CRS model. Taking a second look at the way we modeled set-up so far, the main reason for the inability to capture such global set-up is the fact that so far the set-up was modeled as an ideal functionality that interacts only with the parties of a given protocol execution. In particular, the set-up does not explicitly take part in the ideal process. A natural way to capture global set-up is thus to model the set-up as an ideal functionality 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 set-up functionality exists not only as part of the protocol execution, but also in the *ideal process,* where the protocol is replaced by the trusted party.

More precisely, modify the CRS functionality, $\mathcal{F}_{\text{CRS}}$, as follows: Instead of giving the reference string only to the adversary and the parties running the actual protocol instance, the new "global CRS" functionality, $\mathcal{F}_{\text{GCRS}}$, will give the reference string to all parties and in particular to the environment. (Technically, in order to model $\mathcal{F}_{\text{GCRS}}$ one has to use the generalized UC security notion, as sketched in Section 2.2. Indeed, it is for this reason that the generalized model was first formulated.)

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 set-up, are reminiscent of the proofs of security without set-up, in the sense that the only freedom enjoyed by the simulator is to control the local random choices of the uncorrupted parties. Indeed, as mentioned above, in [CDPW07] the argument of [CF01] is extended to show that no two-party protocol can UC-realize $\mathcal{F}_{\text{COM}}$. The proof extends to rule out any set-up functionality that makes all of its inputs and outputs available to the environment.

New set-up assumptions and constructions. It turns out, however, that it is possible to come up with global set-up functionalities that lend to reasonable implementation and are still sufficient for UC-realizing $\mathcal{F}_{\text{COM}}$. We briefly sketch three such functionalities.

The first functionality is reminiscent of the key set-up functionality from Section 5, $\mathcal{F}_{\text{KS}}$, with the exception that 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, public keys are directly accessible by the environment, even in the ideal process. It turns out that one of the [BCNP04] protocols for UC-realizing $\mathcal{F}_{\text{COM}}$ given $\mathcal{F}_{\text{KS}}$ continues to work even when $\mathcal{F}_{\text{KS}}$ is replaced by the global variant, $\mathcal{F}_{\text{GKS}}$, as long as party corruptions are *non-adaptive.* However, when party corruptions can be adaptive, and the adversary can observe the past internal data of corrupted parties, this protocol becomes insecure. To address this concern, a more sophisticated protocol is constructed in [CDPW07].

A second functionality, called $\mathcal{F}_{\text{ACRS}}$ for "augmented CRS (ACRS)", is reminiscent of the CRS set-up, but is somewhat augmented so as to circumvent the impossibility result for plain CRS. That is, as in the case of $\mathcal{F}_{\text{GCRS}}$, all parties and the environment have access to a short reference string that is taken from a pre-determined distribution. In addition, the ACRS set-up 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 in the formal model *only corrupted parties* may obtain their secret keys. This effect of this modeling is that protocol may not include instructions that require knowledge of the secret keys, and yet corrupted parties are assumed to have access to their secret keys. A protocol for UC-realizing $\mathcal{F}_{\text{COM}}$ (in fact, $\mathcal{F}_{\text{MCOM}}$) given $\mathcal{F}_{\text{ACRS}}$ is constructed in [CDPW07]. The main additional technique on top of the protocol using $\mathcal{F}_{\text{GKS}}$ is a new *identity-based trapdoor commitment (IBTC)* protocol. (IBTC protocols in the Random Oracle model appear in [ZSS 03,AM04].)

"Real world implementations" of $\mathcal{F}_{\text{GKS}}$ and $\mathcal{F}_{\text{ACRS}}$ 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 set-up (CRS), this seems to be a minimal "interactiveness" requirement from the trusted entity.

Another global set-up assumption, formulated in Hofheinz et. al [HMU06], provides each party p with a public "verification key" V_p (chosen by the functionality). Next, the functionality provides p with unforgeable signatures on messages of p's choice, where the signatures can be publicly verified using V_p. It is stressed that the signing keys are not made available to the parties, even to corrupted ones. A protocol for realizing $\mathcal{F}_{\text{COM}}$ given access to this functionality, for non-adaptive corruptions, is given in [HMU06]. This functionality is much more interactive than $\mathcal{F}_{\text{ACRS}}$ or $\mathcal{F}_{\text{GKS}}$. Still, as suggested in [HMU06], in reality it can be implemented by a tamper-proof signing device such as a smart-card.

8 A timing assumption

Last but not least, we consider an alternative approach for making assumptions on the the system in order to guarantee composable security. Specifically, rather than assuming that parties have access to some trusted information, some minimal assumptions are made regarding the synchrony of the system as some point in its execution. More precisely, it is assumed that all messages sent are eventually delivered unmodified within some time bound, and in addition there is a bound on the amount of relative "drift" between local clocks of parties in the system. In [LPT04] it is shown how to UC-realize $\mathcal{F}_{\text{CRS}}$ and $\mathcal{F}_{\text{COM}}$ in such a setting.

The fact that a timing assumption suffices for UC-realizing, say, $\mathcal{F}_{\text{CRS}}$, is not surprising in of itself: Assume for instance that the network is completely synchronous, and furthermore no party (not even corrupted ones) receives messages sent in round i before the last chance to send out its messages for round i. Then a simple, unconditionally secure two-party protocol for UC-realizing $\mathcal{F}_{\text{CRS}}$ would be to simply have each of the two parties send a random string of the appropriate length at a certain round, and then let the reference string be the bitwise xor of the two strings. In [LPT04] it is shown, via a sophisticated protocol and under standard hardness assumptions, how to obtain a similar effect while making (much) weaker synchronization assumptions on the system.

It is interesting to note that the timing assumptions have to hold only during the execution of the protocol for UC-realizing $\mathcal{F}_{\text{CRS}}$. Once the reference string is fixed, no timing assumptions are needed. Also, since there is no trusted piece of information to be passed around, this approach bypasses the "transferability" issues of the other set-up assumptions and provides complete deniability.

9 Realizing authenticated communication

The treatment of Sections 3 through 8 concentrates on the case of ideally authenticated networks, where messages are not modified en route and arrive with an authentic sender identity. More precisely, the parties are assumed to have access to multiple copies of an ideal functionality, $\mathcal{F}_{\text{AUTH}}$, that, roughly, takes input (sid, B, m) from party A, and provides output (sid, A, m) to B, where sid is a session identifier.

It is interesting to note that the above ideal authentication guarantee implicitly carries with it a non-transferability guarantee: The above ideally authenticated communication setting does not provide the recipient of a message sent by party A with any means to convince a third party that a message was indeed sent by A. $\mathcal{F}_{\text{AUTH}}$ provides a similar guarantee. This means that communication via $\mathcal{F}_{\text{AUTH}}$ is in effect "non-transferable", or in other words "deniable".

As observed in [C04], it is impossible to UC-realize $\mathcal{F}_{\text{AUTH}}$ in the "bare" model with no set-up assumptions. Still, $\mathcal{F}_{\text{AUTH}}$ can be UC-realized, via standard authentication mechanisms, when given access to an ideal functionality that allows parties to register public values associated with their identities [C04]. It is stressed that this functionality, $\mathcal{F}_{\text{REG}}$, does not verify knowledge of any secret information associated with the registered value; it merely provides a registration (or, "bulletin-board") service.

However, akin to the formulation of the traditional CRS model, the formulation of $\mathcal{F}_{\text{REG}}$ in [C04] is that of a "local" set-up that is available only to the parties that run the specific protocol instance. Implementing $\mathcal{F}_{\text{REG}}$ is thus susceptible to the same limitations that apply to implementing $\mathcal{F}_{\text{CRS}}$ (see Section 7): Essentially, a new instance of the registration service is needed for each new protocol instance. In particular, similarly to the case of $\mathcal{F}_{\text{CRS}}$, when the [C04] protocol for UC-realizing $\mathcal{F}_{\text{AUTH}}$ uses a "global" registration service that's available to arbitrary protocols, authentication becomes "transferable". (In fact, a publicly verifiable signature by the sender on the transmitted information becomes available.)

Modeling authenticated communication in the presence of global set-up is an interesting challenge. One direction is to model the security guarantees provided by standard authentication mechanisms (such as the simple signature-based mechanism studied in [C04]) in the presence of global set-up. These guarantees are naturally described by means of an ideal authentication functionality that allows for transferability *even in ideal process.* Another direction is to study protocols that UC-realize the original, non-transferable version of $\mathcal{F}_{\text{AUTH}}$ even when given only globally available set-up. This is an interesting venue for current and future research.

10 Conclusion and open problems

We have exemplified the need for trusted set-up models in order to obtain composable security, and have studied a variety of set-up models. These models have very different characteristics, both from the point of view of the guarantees provided to protocols designed in these models, and from the point of view of the requirements from practical implementations of the models.

While some progress has been made in the past few years towards understanding how to formulate models that allow bypassing the strong impossibility results regarding composable security, how to develop protocols in these models, and how to implement such models in practice, much remains to be understood. Some specific challenges and questions include:

1. Finding protocols that use current set-up models more efficiently. Finding easier and more secure ways to implement existing set-up models in practice. Finding new set-up models that allow for more efficient protocols and/or easier implementations.
2. Finding a characterization of the set-up models that allow for UC-realizing, say, $\mathcal{F}_{\text{COM}}$ (or any other ideal functionality that allows for UC-realizing general ideal functionalities). We've seen that set-up functionalities can have very different flavors and characteristics. Are there some salient properties that are common to all and are necessary and/or sufficient for UC-realizing $\mathcal{F}_{\text{COM}}$?

3. More specifically, are there *global* set-up models that allow for adversarial control over the set-up information akin to $\mathcal{F}_{\text{SUN}}$, and still allow for UC-realizing $\mathcal{F}_{\text{COM}}$? Are there set-up models that allow for adversarial control over the set-up information, and at the same time allow for UC-realizing authenticated communication?
4. Are there general relationships between set-up models that allow for UC-realizing authenticated communication and set-up models that allow for UC-realizing $\mathcal{F}_{\text{COM}}$?
5. More generally, how can we better model the information shared between protocol instances in arbitrary systems? Is global set-up information the only information that can be shared, or are there other ways to share state and information? How to capture these? An indication that in some situations protocols indeliberately (but inevitably) share more information than just the set-up is given in [YYZ07B].

References

[AM04] 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/`.

[B01] B. Barak. How to go Beyond the Black-Box Simulation Barrier. In *42nd FOCS*, pp. 106–115, 2001.

[BCNP04] B. Barak, R. Canetti, J. B. Nielsen, R. Pass. Universally Composable Protocols with Relaxed Set-Up Assumptions. 45th FOCS, pp. 186–195. 2004.

[BL04] B. Barak and Y. Lindell. Strict polynomial-time in simulation and extraction. *SIAM J. Comput*, 33(4):738–818, 2004.

[BS05] B. Barak and A. Sahai, How To Play Almost Any Mental Game Over the Net - Concurrent Composition via Super-Polynomial Simulation. 46th FOCS, 2005.

[B91] D. Beaver. Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. J. Cryptology, (1991) 4: 75-122.

[BCG93] M. Ben-Or, R. Canetti and O. Goldreich. Asynchronous Secure Computation. 25th Symposium on Theory of Computing (STOC), 1993, pp. 52-61. Longer version appears in TR #755, CS dept., Technion, 1992.

[BGW88] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. 20th Symposium on Theory of Computing (STOC), ACM, 1988, pp. 1-10.

[BKR94] M. Ben-Or, B. Kelmer and T. Rabin. Asynchronous Secure Computations with Optimal Resilience. *13th PODC*, 1994, pp. 183-192.

[BFM88] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *STOC 88*, pages 103–112, 1988.

[C00] R. Canetti. Security and composition of multi-party cryptographic protocols. J. Cryptology, Vol. 13, No. 1, winter 2000.

[C01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Extended abstract in *42nd FOCS*, 2001. A revised version (2005) is available at IACR Eprint Archive, eprint.iacr.org/2000/067/ and at the ECCC archive, http://eccc.uni-trier.de/eccc-reports/2001/TR01-016/.

[C04] R. Canetti. Universally Composable Signature, Certification, and Authentication. *17th Computer Security Foundations Workshop (CSFW)*, 2004. Long version at eprint.iacr.org/2003/239.

[Can06] Ran Canetti. Security and composition of cryptographic protocols: A tutorial. SIGACT News, Vol. 37, Nos. 3 & 4, 2006. Available also at the Cryptology ePrint Archive, Report 2006/465.

[CDPW07] R. Canetti, Y. Dodis, R. Pass and S. Walfish. Universally Composable Security with Pre-Existing Setup. *4th theory of Cryptology Conference (TCC)*, 2007.

[CFGN96] R. Canetti, U. Feige, O. Goldreich and M. Naor. Adaptively Secure Computation. *28th Symposium on Theory of Computing (STOC)*, ACM, 1996. Fuller version in MIT-LCS-TR 682, 1996.

[CF01] R. Canetti and M. Fischlin. Universally Composable Commitments. Crypto '01, 2001.

[CKL03] R. Canetti, E. Kushilevitz, Y. Lindell. On the Limitations of Universally Composable Two-Party Computation without Set-up Assumptions. EUROCRYPT 2003, pp. 68–86, 2003. Extended version at the eprint archive, eprint.iacr.org/2004/116.

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, A. Sahai. Universally composable two-party and multi-party secure computation. 34th STOC, pp. 494–503, 2002.

[CPS07] R. Canetti, R. Pass, a. shelat. Cryptography from sunspots: How to use an imperfect reference string. *39th Symposium on Theory of Computing (STOC),* ACM, 2007.

[CR03] R. Canetti and T. Rabin. Universal Composition with Joint State. Crypto'03, 2003.

[DDMRS06] A. Datta, A. Derek, J. C. Mitchell, A. Ramanathan and A. Scedrov. Games and the Impossibility of Realizable Ideal Functionality. 3rd theory of Cryptology Conference (TCC), 2006.

[DOPS04] Y. Dodis, S. Ong, M. Prabhakaran, and A. Sahai. On the (im)possibility of cryptography with imperfect randomness. In *FOCS 04*, pages 196–205, 2004.

[DDN00] D. Dolev. C. Dwork and M. Naor. Non-malleable cryptography. SIAM. J. Computing, Vol. 30, No. 2, 2000, pp. 391-437. Preliminary version in 23rd Symposium on Theory of Computing (STOC), 1991.

[DNS98] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In 30th STOC, pages 409–418, 1998.

[FS89] U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In *CRYPTO*, pages 526–544, 1989.

[F88] F. Forges. Can sunspots replace a mediator? *J. of Math. Ec.*, 17(4):347–368, 1988.

[GRR98] R. Gennaro, M. Rabin and T Rabin. Simplified VSS and Fast-track Multiparty Computations with Applications to Threshold Cryptography, *17th PODC*, 1998, pp. 101-112.

[GK89] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. SIAM. J. Computing, Vol. 25, No. 1, 1996.

[GMW87] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game. 19th Symposium on Theory of Computing (STOC), 1987, pp. 218-229.

[GL90] S. Goldwasser, and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. CRYPTO '90, LNCS 537, 1990.

[HMU06] D. Hofheinz, J. Muller-Quade, and D. Unruh. Universally Composable Zero-Knowledge Arguments and Commitments from Signature Cards. Tatra Mountains Mathematical Publications, 2005.

[KSW97] J. Kelsey, B. Schneier, D. Wagner. Protocol Interactions and the Chosen Protocol Attack. Security Protocols Workshop 1997: 91-104

[KLR06] E. Kushilevitz, Y. Lindell and T. Rabin. Information-Theoretically Secure Protocols and Security Under Composition. 38th STOC, pages 109-118, 2006.

[L03] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. 43rd FOCS, pp. 394–403. 2003.

[L04] Y. Lindell. Lower Bounds for Concurrent Self Composition. 1st Theory of Cryptology Conference (TCC), pp. 203–222. 2004.

[LPT04] Y. Lindell, M. Prabhakaran, Y. Tauman. Concurrent General Composition of Secure Protocols in the Timing Model. Manuscript, 2004.

[MMY06] T. Malkin, R. Moriarty and N. Yakovenko. Generalized Environmental Security from Number Theoretic Assumptions. *3rd Theory of Cryptology Conference (TCC),* 2006, pp. 343-359.

[MR91] S. Micali and P. Rogaway. Secure Computation. unpublished manuscript, 1992. Preliminary version in CRYPTO '91, LNCS 576, 1991.

[P03] R. Pass. On Deniabililty in the Common Reference String and Random Oracle Model. In *Proc. of Crypto*, LNCS 2729, pp. 216–337, 2003.

[PW00] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. 7th ACM Conf. on Computer and Communication Security (CCS), 2000, pp. 245-254.

[PW01] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. IEEE Symposium on Security and Privacy, May 2001. Preliminary version in http://eprint.iacr.org/2000/066 and IBM Research Report RZ 3304 (#93350), IBM Research, Zurich, December 2000.

[PS04] M. Prabhakaran, A. Sahai. New notions of security: achieving universal composability without trusted setup. 36th STOC, pp. 242–251. 2004.

[RB89] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multi-party Protocols with Honest Majority. 21st Symposium on Theory of Computing (STOC), 1989, pp. 73-85.

[TV00] L. Trevisan and S. Vadhan. Extracting randomness from samplable distributions. In *FOCS 00*, pages 32–42, 2000.

[YYZ07A] A. Yao, F. F. Yao and Y. Zhao. A Note on Universal Composable Zero Knowledge in Common Reference String Model. *TAMC'07,* pages 462-473, 2007.

[YYZ07B] A. Yao, F. F. Yao and Y. Zhao. A Note on the Feasibility of Generalized Universal Composability. *TAMC'07,* pages 474-485, 2007.

[ZSS 03] F. Zhang, R. Safavi-Naini and W. Susilo. ID-Based Chameleon Hashes from Bilinear Pairings. Available at `http://eprint.iacr.org/2003/208/`.