Universally Composable Commitments
(Extended Abstract)

Ran Canetti[1] and Marc Fischlin[2*]

[1] IBM T.J. Watson Research Center
canetti@watson.ibm.com
[2] Goethe-University of Frankfurt
marc@mi.informatik.uni-frankfurt.de

Abstract. We propose a new security measure for commitment protocols, called Universally Composable (UC) Commitment. The measure guarantees that commitment protocols behave like an "ideal commitment service," even when concurrently composed with an arbitrary set of protocols. This is a strong guarantee: it implies that security is maintained even when an unbounded number of copies of the scheme are running concurrently, it implies non-malleability (not only with respect to other copies of the same protocol but even with respect to other protocols), it provides resilience to selective decommitment, and more.
Unfortunately, two-party UC commitment protocols do not exist in the plain model. However, we construct two-party UC commitment protocols, based on general complexity assumptions, in the *common reference string model* where all parties have access to a common string taken from a predetermined distribution. The protocols are non-interactive, in the sense that both the commitment and the opening phases consist of a single message from the committer to the receiver.

Keywords: Commitment schemes, concurrent composition, non-malleability, security analysis of protocols.

1 Introduction

Commitment is one of the most basic and useful cryptographic primitives. It is an essential building block in Zero-Knowledge protocols (e.g., [GMW91,BCC88, D89]), in general function evaluation protocols (e.g., [GMW87,GHY88,G98]), in contract-signing and electronic commerce, and more. Indeed, commitment protocols have been studied extensively in the past two decades (e.g., [N91,DDN00, NOVY92,B99,DIO98,FF00,DKOS01]).

The basic idea behind the notion of commitment is attractively simple: A *committer* provides a *receiver* with the digital equivalent of a "sealed envelope" containing a value x. From this point on, the committer cannot change the value inside the envelope, and, as long as the committer does not assist the receiver

[*] part of this work done while visiting IBM T.J. Watson Research Center.

J. Kilian (Ed.): CRYPTO 2001, LNCS 2139, pp. 19–40, 2001.

in opening the envelope, the receiver learns nothing about x. When both parties cooperate, the value x is retrieved in full.

Formalizing this intuitive idea is, however, non-trivial. Traditionally, two quite distinct basic flavors of commitment are formalized: *unconditionally binding* and *unconditionally secret* commitment protocols (see, e.g., [G95]). These basic definitions are indeed sufficient for some applications (see there). But they treat commitment as a "stand alone" task and do not in general guarantee security when a commitment protocol is used as a building-block within other protocols, or when multiple copies of a commitment protocol are carried out together. A good first example for the limitations of the basic definitions is the *selective decommitment* problem [DNRS99], that demonstrates our inability to prove some very minimal composition properties of the basic definitions.

Indeed, the basic definitions turned out to be inadequate in some scenarios, and stronger variants that allow to securely "compose" commitment protocols —both with the calling protocol and with other invocations of the commitment protocol— were proposed and successfully used in some specific contexts. One such family of variants make sure that knowledge of certain trapdoor information allows "opening" commitments in more than a single way. These include *chameleon commitments* [BCC88], *trapdoor commitments* [FS90] and *equivocable commitments* [B99]. Another strong variant is *non-malleable commitments* [DDN00], where it is guaranteed that a dishonest party that receives an unopened commitment to some value x will be unable to commit to a value that depends on x in any way, except for generating another commitment to x. (A more relaxed variant of the [DDN00] notion of non-malleability is *non-malleability with respect to opening* [DIO98,FF00,DKOS01].)

These stronger measures of security for commitment protocols are indeed very useful. However they only solve specific problems and stop short of guaranteeing that commitment protocols maintain the expected behavior in general cryptographic contexts, or in other words when composed with arbitrary protocols. To exemplify this point, notice for instance that, although [DDN00] remark on more general notions of non-malleability, the standard notion of non-malleability considers only other copies of the same protocol. There is no guarantee that a malicious receiver is unable to "maul" a given commitment by using a totally different commitment protocol. And it is indeed easy to come up with two commitment protocols $\mathcal{C}$ and $\mathcal{C}'$ such that both are non-malleable with respect to themselves, but an adversary that plays a receiver in $\mathcal{C}$ can generate a $\mathcal{C}'$-commitment to a related value.

This work proposes a measure of security for commitment protocols that guarantees the "envelope-like" intuitive properties of commitment even when the commitment protocol is *concurrently composed* with an arbitrary set of protocols. In particular, protocols that satisfy this measure (called universally composable (UC) commitment protocols) remain secure even when an unbounded number of copies of the protocol are executed concurrently in an adversarially controlled way; they are resilient to selective decommitment attacks; they are non-malleable both with respect to other copies of the same protocol and with respect to arbitrary commitment protocols. In general, a UC commitment protocol successfully emulates an "ideal commitment service" for any application protocol (be it a Zero-Knowledge protocol, a general function evaluation protocol, an e-commerce application, or any combination of the above).

This measure of security for commitment protocols is very strong indeed. It is perhaps not surprising then that UC commitment protocols which involve only the committer and the receiver do not exist in the standard "plain model" of computation where no set-up assumptions are provided. (We formally prove this fact.) However, in the *common reference string* (CRS) model things look better. (The CRS model is a generalization of the *common random string* model. Here all parties have access to a common string that was chosen according to some predefined distribution. Other equivalent terms include the *reference string* model [D00] and the *public parameter* model [FF00].) In this model we construct UC commitment protocols based on standard complexity assumptions. A first construction, based on any family of trapdoor permutations, requires the length of the reference string to be linear in the number of invocations of the protocol throughout the lifetime of the system. A second protocol, based on any claw-free pair of trapdoor permutations, uses a short reference string for an unbounded number of invocations. The protocols are non-interactive, in the sense that both the commitment and the decommitment phases consist of a single message from the committer to the receiver. We also note that UC commitment protocols can be constructed in the plain model, if the committer and receiver are assisted by third parties (or, "servers") that participate in the protocol without having local inputs and outputs, under the assumption that a majority of the servers remain uncorrupted.

1.1 On the New Measure

Providing meaningful security guarantees under composition with arbitrary protocols requires using an appropriate framework for representing and arguing about such protocols. Our treatment is based in a recently proposed such general framework [C00a]. This framework builds on known definitions for function evaluation and general tasks [GL90,MR91,B91,PW94,C00,DM00,PW01], and allows defining the security properties of practically any cryptographic task. Most importantly, in this framework security of protocols is maintained under general *concurrent* composition with an unbounded number of copies of arbitrary protocols. We briefly summarize the relevant properties of this framework. See more details in Section 2.1 and in [C00a].

As in prior general definitions, the security requirements of a given task (i.e., the functionality expected from a protocol that carries out the task) are captured via a set of instructions for a "trusted party" that obtains the inputs of the participants and provides them with the desired outputs. However, as opposed to the standard case of secure function evaluation, here the trusted party (which is also called the ideal functionality) runs an arbitrary algorithm and in particular may interact with the parties in several iterations, while maintaining state in between. Informally, a protocol securely carries out a given task if running the protocol amounts to "emulating" an ideal process where the parties hand their inputs to the appropriate ideal functionality and obtain their outputs from it, without any other interaction.

In order to allow proving the concurrent composition theorem, the notion of emulation in [c00a] is considerably stronger than previous ones. Traditionally, the model of computation includes the parties running the protocol and an adversary, $\mathcal{A}$, and "emulating an ideal process" means that for any adversary $\mathcal{A}$ there should exist an "ideal process adversary" (or, simulator) $\mathcal{S}$ that results in similar distribution on the outputs for the parties. Here an additional adversarial entity, called the environment $\mathcal{Z}$, is introduced. The environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. A protocol is said to securely realize a given ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$ 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 $\mathcal{F}$. See [c00a] for more motivating discussion on the role of the environment.) Note that the definition requires the "ideal-process adversary" (or, simulator) $\mathcal{S}$ to interact with $\mathcal{Z}$ throughout the computation. Furthermore, $\mathcal{Z}$ cannot be "rewound".

The following *universal composition* theorem is proven in [c00a]. Consider a protocol π that operates in a *hybrid* model of computation where parties can communicate as usual, and in addition have ideal access to (an unbounded number of copies of) some ideal functionality $\mathcal{F}$. Let ρ be a protocol that securely realizes $\mathcal{F}$ as sketched above, and let π^ρ be the "composed protocol". That is, π^ρ is identical to π with the exception that each interaction with some copy of $\mathcal{F}$ is replaced with a call to (or an invocation of) an appropriate instance of ρ. Similarly, ρ-outputs are treated as values provided by the appropriate copy of $\mathcal{F}$. Then, π and π^ρ have essentially the same input/output behavior. In particular, if π securely realizes some ideal functionality $\mathcal{G}$ given ideal access to $\mathcal{F}$ then π^ρ securely realizes $\mathcal{G}$ from scratch.

To apply this general framework to the case of commitment protocols, we formulate an ideal functionality $\mathcal{F}_{com}$ that captures the expected behavior of commitment. Universally Composable (uc) commitment protocols are defined to be those that securely realize $\mathcal{F}_{com}$. Our formulation of $\mathcal{F}_{com}$ is a straightforward transcription of the "envelope paradigm": $\mathcal{F}_{com}$ first waits to receive a request from some party C to commit to value x for party R. (C and R are identities of two parties in a multiparty network). When receiving such a request, $\mathcal{F}_{com}$ records the value x and notifies R that C has committed to some value for him. When C later sends a request to open the commitment, $\mathcal{F}_{com}$ sends the recorded value x to R, and halts. (Some other variants of $\mathcal{F}_{com}$ are discussed within.) The general composition theorem now implies that running (multiple copies of) a uc commitment protocol π is essentially equivalent to interacting with the same number of copies of $\mathcal{F}_{com}$, regardless of what the calling protocol does. In particular, the calling protocol may run other commitment protocols and may use the committed values in any way. As mentioned above, this implies a strong version of non-malleability, security under concurrent composition, resilience to selective decommitment, and more.

The definition of security and composition theorem carry naturally to the CRS model as well. However, this model hides a caveat: The composition operation requires that each copy of the UC commitment protocol will have its own copy of the CRS. Thus, a protocol that securely realizes $\mathcal{F}_{com}$ as described above is highly wasteful of the reference string. In order to capture protocols where multiple commitments may use the same reference string we formulate a natural extension of $\mathcal{F}_{com}$ that handles multiple commitment requests. Let $\mathcal{F}_{mcom}$ denote this extension.

We remark that UC commitment protocols need not, by definition, be neither unconditionally secret nor unconditionally binding. Indeed, one of the constructions presented here has neither property.

1.2 On the Constructions

At a closer look, the requirements from a UC commitment protocol 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 $\mathcal{F}_{com}$ with a committed value. *(b).* When the committer is uncorrupted, $\mathcal{S}$ has to be able to generate a kosher-looking "simulated commitment" c that can be "opened" to any value (which will become known only later). 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 $\mathcal{Z}$.* (Note that non-malleability is not explicitly required in this description. It is, however, implied by the above requirements.)

From the above description it may look plausible that no simulator $\mathcal{S}$ exists that meets the above requirements in the plain model. Indeed, we formalize and prove this statement for the case of protocols that involve only a committer and a receiver. (In the case where the committer and the receiver are assisted by third parties, a majority of which is guaranteed to remain uncorrupted, standard techniques for multiparty computation are sufficient for constructing UC commitment protocols. See [C00a] for more details.)

In the CRS model the simulator is "saved" by the ability to choose the reference string and plant trapdoors in it. Here we present two UC commitment protocols. The first one (that securely realizes functionality $\mathcal{F}_{com}$) is based on the equivocable commitment protocols of [DIO98], while allowing the simulator to have trapdoor information that enables it to extract the values committed to by corrupted parties. However, the equivocability property holds only with respect to a single usage of the CRS. Thus this protocol fails to securely realize the multiple commitment functionality $\mathcal{F}_{mcom}$.

In the second protocol (that securely realizes $\mathcal{F}_{mcom}$), the reference string contains a description of a claw-free pair of trapdoor permutations and a public encryption key of an encryption scheme that is secure against adaptive chosen ciphertext attacks (CCA) (as in, say, [DDN00,RS91,BDPR98,CS98]). Commitments are generated via standard use of a claw-free pair, combined with encrypting potential decommitments. The idea to use CCA-secure encryption in this context is taken from [L00,DKOS01].

Both protocols implement commitment to a single bit. Commitment to arbitrary strings is achieved by composing together several instances of the basic protocol. Finding more efficient UC commitment protocols for string commitment is an interesting open problem.

Applicability of the notion. In addition to being an interesting goal in their own right, UC commitment protocols can potentially be very useful in constructing more complex protocols with strong security and composability properties. To demonstrate the applicability of the new notion, we show how UC commitment protocols can be used in a simple way to construct strong Zero-Knowledge protocols *without any additional cryptographic assumptions.*

Related work. Pfitzmann et. al. [PW94,PW01] present another definitional framework that allows capturing the security requirements of general reactive tasks, and prove a concurrent composition theorem with respect to their framework. Potentially, our work could be cast in their framework as well; however, the composition theorem provided there is considerably weaker than the one in [C00a].

Organization. Section 2 shortly reviews the general framework of [C00a] and presents the ideal commitment functionalities $\mathcal{F}_{com}$ and $\mathcal{F}_{mcom}$. Section 3 presents and proves security of the protocols that securely realize $\mathcal{F}_{com}$ and $\mathcal{F}_{mcom}$. Section 4 demonstrates that functionalities $\mathcal{F}_{com}$ and $\mathcal{F}_{mcom}$ cannot be realized in the plain model by a two-party protocol. Section 5 presents the application to constructing Zero-Knowledge protocols. For lack of space most proofs are omitted. They appear in [CF01].

2 Definitions

Section 2.1 shortly summarizes the relevant parts of the general framework of [C00a], including the definition of security and the composition theorem. Section 2.2 defines the ideal commitment functionalities, $\mathcal{F}_{com}$ and $\mathcal{F}_{mcom}$.

2.1 The General Framework

Protocol syntax. Following [GMRa89,G95], protocols are represented as a set of interactive Turing machines (ITMs). Specifically, the input and output tapes model inputs and outputs that are received from and given to other programs running on the same machine, and the communication tapes model messages sent to and received from the network. Adversarial entities are also modeled as ITMs; we concentrate on a non-uniform complexity model where the adversaries have an arbitrary additional input, or an "advice".

The adversarial model. [c00a] discusses several models of computation. We concentrate on one main model, aimed at representing current realistic communication networks (such as the Internet). Specifically, the network is *asynchronous* without guaranteed delivery of messages. The communication is public (i.e., all messages can be seen by the adversary) but ideally authenticated (i.e., messages cannot be modified by the adversary). In addition, parties have unique *identities.*[1] The adversary is adaptive in corrupting parties, and is active (or, *Byzantine*) in its control over corrupted parties. Finally, the adversary and environment are restricted to probabilistic polynomial time (or, "feasible") computation.

Protocol execution in the real-life model. We sketch the "mechanics" of executing a given protocol π (run by parties $P_1, ..., P_n$) with some adversary $\mathcal{A}$ and an environment machine $\mathcal{Z}$ with input z. All parties have a security parameter $k \in \mathbf{N}$ and are polynomial in k. The execution consists of a sequence of *activations,* where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$, or some P_i) is activated. The activated participant reads information from its input and incoming communication tapes, executes its code, and possibly writes information on its outgoing communication tapes and output tapes. In addition, the environment can write information on the input tapes of the parties, and read their output tapes. The adversary can read messages off the outgoing message tapes of the parties and *deliver* them by copying them to the incoming message tapes of the recipient parties. The adversary can also corrupt parties, with the usual consequences that it learns the internal information known to the corrupted party and that from now on it controls that party.

The environment is activated first; once activated, it may choose to activate either one of the parties (with some input value) or to activate the adversary. Whenever the adversary delivers a message to some party P, this party is activated next. Once P's activation is complete, the environment is activated. Throughout, the environment and the adversary may exchange information freely using their input and output tapes. The output of the protocol execution is the output of $\mathcal{Z}$. (Without loss of generality $\mathcal{Z}$ outputs a single bit.)

Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \boldsymbol{r})$ denote the output of environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and parties running protocol π on security parameter k, input z and random input $\boldsymbol{r} = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1 \dots r_n$ as described above (z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$, $r_{\mathcal{A}}$ for $\mathcal{A}$; r_i for party P_i). Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ denote the random variable

[1] Indeed, the communication in realistic networks is typically *unauthenticated,* in the sense that messages may be adversarially modified en-route. In addition, there is no guarantee that identities will be unique. Nonetheless, since authentication and the guarantee of unique identities can be added independently of the rest of the protocol, we allow ourselves to assume ideally authenticated channels and unique identities. See [c00a] for further discussion.

describing $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \boldsymbol{r})$ when $\boldsymbol{r}$ is uniformly chosen. Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

The ideal process. Security of protocols is defined via comparing the protocol execution in the real-life model to an *ideal process* for carrying out the task at hand. A key ingredient in the ideal process is the ideal functionality that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM that interacts with the environment and the adversary via a process described below. More specifically, the ideal process involves an ideal functionality $\mathcal{F}$, an ideal process adversary $\mathcal{S}$, an environment $\mathcal{Z}$ on input z and a set of dummy parties $\tilde{P}_1, ..., \tilde{P}_n$. The dummy parties are fixed and simple ITMS: Whenever a dummy party is activated with input x, it forwards x to $\mathcal{F}$, say by copying x to its outgoing communication tape; whenever it is activated with incoming message from $\mathcal{F}$ it copies this message to its output. $\mathcal{F}$ receives information from the (dummy) parties by reading it off their outgoing communication tapes. It hands information back to the parties by sending this information to them. The ideal-process adversary $\mathcal{S}$ proceeds as in the real-life model, except that it has no access to the contents of the messages sent between $\mathcal{F}$ and the parties. In particular, $\mathcal{S}$ is responsible for delivering messages, and it can corrupt dummy parties, learn the information they know, and control their future activities.

The order of events in the ideal process is as follows. As in the real-life model, the environment is activated first. As there, parties are activated when they receive new information (here this information comes either from the environment or from $\mathcal{F}$). In addition, whenever a dummy party P sends information to $\mathcal{F}$, then $\mathcal{F}$ is activated. Once $\mathcal{F}$ completes its activation, P is activated again. Also, $\mathcal{F}$ may exchange messages directly with the adversary. It is stressed that in the ideal process there is no communication among the parties. The only "communication" is in fact idealized transfer of information between the parties and the ideal functionality. The output of the ideal process is the (one bit) output of $\mathcal{Z}$.

Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z, \boldsymbol{r})$ denote the output of environment $\mathcal{Z}$ after interacting in the ideal process with adversary $\mathcal{S}$ and ideal functionality $\mathcal{F}$, on security parameter k, input z, and random input $\boldsymbol{r} = r_\mathcal{Z}, r_\mathcal{S}, r_\mathcal{F}$ as described above (z and $r_\mathcal{Z}$ for $\mathcal{Z}$, $r_\mathcal{S}$ for $\mathcal{S}$; $r_\mathcal{F}$ for $\mathcal{F}$). Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)$ denote the random variable describing $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z, \boldsymbol{r})$ when $\boldsymbol{r}$ is uniformly chosen. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Securely realizing an ideal functionality. We say that a protocol ρ securely realizes an ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running ρ in the real-life process, or it is interaction with $\mathcal{A}$ and $\mathcal{F}$ in the ideal process. This means that, from the point of view of the environment, running protocol ρ is 'just as good' as interacting with an ideal process for $\mathcal{F}$. (In a way, $\mathcal{Z}$ serves as an "interactive distinguisher" between the two processes. Here it is important that $\mathcal{Z}$ can provide the process in question with *adaptively chosen* inputs throughout the computation.)

Definition 1. *Let* $\mathcal{X} = \{X(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ *and* $\mathcal{Y} = \{Y(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ *be two distribution ensembles over* $\{0,1\}$. *We say that* $\mathcal{X}$ *and* $\mathcal{Y}$ *are* indistinguishable *(written* $\mathcal{X} \overset{c}{\approx} \mathcal{Y}$*) if for any* $c \in \mathbf{N}$ *there exists* $k_0 \in \mathbf{N}$ *such that* $|\Pr(X(k,a) = 1) - \Pr(Y(k,a) = 1)| < k^{-c}$ *for all* $k > k_0$ *and all* a.

Definition 2 ([c00a]). *Let* $n \in \mathbf{N}$. *Let* $\mathcal{F}$ *be an ideal functionality and let* π *be an* n*-party protocol. We say that* π securely realizes $\mathcal{F}$ *if for any adversary* $\mathcal{A}$ *there exists an ideal-process adversary* $\mathcal{S}$ *such that for any environment* $\mathcal{Z}$ *we have* $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$.

The common reference string (CRS) model. In this model it is assumed that all the participants have access to a common string that is drawn from some specified distribution. (This string is chosen ahead of time and is made available before any interaction starts.) It is stressed that the security of the protocol depends on the fact that the reference string is generated using a pre-specified randomized procedure, and no "trapdoor information" related to the string exists in the system. This in turn implies full trust in the entity that generates the reference string. More precisely, the CRS model is formalized as follows.

- The real-life model of computation is modified so that all participants have access to a common string that is chosen in advance according to some distribution (specified by the protocol run by the parties) and is written in a special location on the input tape of each party.
- The ideal process is modified as follows. In a preliminary step, the ideal-model adversary chooses a string in some arbitrary way and writes this string on the input tape of the environment machine. After this initial step the computation proceeds as before. It is stressed that the ideal functionality has no access to the reference string.

Justification of the CRS model. Allowing the ideal-process adversary (i.e., the simulator) to choose the reference string is justified by the fact that the behavior of the ideal functionality does not depend on the reference string. This means that the security guarantees provided by the ideal process hold regardless of how the reference string is chosen and whether trapdoor information regarding this string is known.

On the composition theorem: The hybrid model. In order to state the composition theorem, and in particular in order to formalize the notion of a real-life protocol with access to an ideal functionality, the hybrid model of computation with access to an ideal functionality $\mathcal{F}$ (or, in short, the $\mathcal{F}$-hybrid model) is formulated. This model is identical to the real-life model, with the following exceptions. In addition to sending messages to each other, the parties may send messages to and receive messages from an unbounded number of copies of $\mathcal{F}$. Each copy of $\mathcal{F}$ is identified via a unique session identifier (SID); all messages addressed to this copy and all message sent by this copy carry the corresponding SID. (The SIDs are chosen by the protocol run by the parties.)

The communication between the parties and each one of the copies of $\mathcal{F}$ mimics the ideal process. That is, once a party sends a message to some copy of $\mathcal{F}$, that copy is immediately activated and reads that message off the party's tape. Furthermore, although the adversary in the hybrid model is responsible for delivering the messages from the copies of $\mathcal{F}$ to the parties, it does not have access to the contents of these messages.

Replacing a call to $\mathcal{F}$ with a protocol invocation. Let π be a protocol in the $\mathcal{F}$-hybrid model, and let ρ be a protocol that securely realizes $\mathcal{F}$ (with respect to some class of adversaries). The composed protocol π^ρ is constructed by modifying the code of each ITM in π so that the first message sent to each copy of $\mathcal{F}$ is replaced with an invocation of a new copy of π with fresh random input, and with the contents of that message as input. Each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of ρ, with the contents of that message given to ρ as new input. Each output value generated by a copy of ρ is treated as a message received from the corresponding copy of $\mathcal{F}$.

Theorem statement. In its general form, the composition theorem basically says that if ρ securely realizes $\mathcal{F}$ then an execution of the composed protocol π^ρ "emulates" an execution of protocol π in the $\mathcal{F}$-hybrid model. That is, for any real-life adversary $\mathcal{A}$ there exists an adversary $\mathcal{H}$ in the $\mathcal{F}$-hybrid model such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and π^ρ in the real-life model or it is interacting with $\mathcal{H}$ and π in the $\mathcal{F}$-hybrid model..

A more specific corollary of the general theorem states that if π securely realizes some functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model, and ρ securely realizes $\mathcal{F}$ in the real-life model, then π^ρ securely realizes $\mathcal{G}$ in the real-life model. (Here one has to define what it means to securely realize functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model. This is done in the natural way.)

Theorem 1 ([c00a]). *Let $\mathcal{F}, \mathcal{G}$ be ideal functionalities. Let π be an n-party protocol that realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model and let ρ be an n-party protocol that securely realizes $\mathcal{F}$ Then protocol π^ρ securely realizes $\mathcal{G}$.*

Protocol composition in the CRS model. Some words of clarification are in order with respect to the composition theorem in the CRS model. Specifically, it is stressed that each copy of protocol ρ within the composed protocol π^ρ should have its own copy of the reference string, or equivalently uses a separate portion of a long string. (If this is not the case then the theorem no longer holds in general.) As seen below, the behavior of protocols where several copies of the protocol use the same instance of the reference string can be captured using ideal functionalities that represent multiple copies of the protocol within a single copy of the functionality.

2.2 The Commitment Functionalities

We propose ideal functionalities that represent the intuitive "envelope-like" properties of commitment, as sketched in the introduction. Two functionalities are presented: functionality $\mathcal{F}_{com}$ that handles a single commitment-decommitment process, and functionality $\mathcal{F}_{mcom}$ that handles multiple such processes.. (Indeed, in the plain model functionality $\mathcal{F}_{mcom}$ would be redundant, since one can use the composition theorem to obtain protocols that securely realize $\mathcal{F}_{mcom}$ from any protocol that securely realizes $\mathcal{F}_{com}$. However, in the CRS model realizing $\mathcal{F}_{mcom}$ is considerably more challenging than realizing $\mathcal{F}_{com}$.) Some further discussion on the functionalities and possible variants appears in [CF01].

Both functionalities are presented as *bit* commitments. Commitments to strings can be obtained in a natural way using the composition theorem. It is also possible, in principle, to generalize $\mathcal{F}_{com}$ and $\mathcal{F}_{mcom}$ to allow commitment to strings. Such extensions may be realized by string-commitment protocols that are more efficient than straightforward composition of bit commitment protocols. Finding such protocols is an interesting open problem.

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

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

1. Upon receiving a value $(\texttt{Commit}, sid, P_i, P_j, b)$ from P_i, where $b \in \{0,1\}$, record the value b and send the message $(\texttt{Receipt}, sid, P_i, P_j)$ to P_j and $\mathcal{S}$. Ignore any subsequent $\texttt{Commit}$ messages.
2. Upon receiving a value $(\texttt{Open}, sid, P_i, P_j)$ from P_i, proceed as follows: If some value b was previously recoded, then send the message $(\texttt{Open}, sid, P_i, P_j, b)$ to P_j and $\mathcal{S}$ and halt. Otherwise halt.

Fig. 1. The Ideal Commitment functionality for a single commitment

Functionality $\mathcal{F}_{com}$, described in Figure 1, proceeds as follows. The commitment phase is modeled by having $\mathcal{F}_{com}$ receive a value $(\texttt{Commit}, sid, P_i, P_j, b)$, from some party P_i (the committer). Here *sid* is a Session ID used to distinguish among various copies of $\mathcal{F}_{com}$, P_j is the identity of another party (the receiver), and $b \in \{0,1\}$ is the value committed to. In response, $\mathcal{F}_{com}$ lets the receiver P_j *and the adversary* $\mathcal{S}$ know that P_i has committed to some value, and that this value is associated with session ID *sid*. This is done by sending the message $(\texttt{Receipt}, sid, P_i, P_j)$ to P_j and $\mathcal{S}$. The opening phase is initiated by the committer sending a value $(\texttt{Open}, sid, P_i, P_j)$ to $\mathcal{F}_{com}$. In response, $\mathcal{F}_{com}$ hands the value $(\texttt{Open}, sid, P_i, P_j, b)$ to P_j and $\mathcal{S}$.

Functionality $\mathcal{F}_{mcom}$, presented in Figure 2, essentially mimics the operation of $\mathcal{F}_{com}$ for an unbounded number of times. In addition to the session ID *sid*, functionality $\mathcal{F}_{mcom}$ uses an additional identifier, a Commitment ID *cid*, that is used to distinguish among the different commitments that take place within a single run of $\mathcal{F}_{mcom}$. The record for a committed value now includes the Commitment ID, plus the identities of the committer and receiver. To avoid ambiguities, no two commitments with the same committer and verifier are allowed to have the same Commitment ID. It is stressed that the various `Commit` and `Open` requests may be interleaved in an arbitrary way. Also, note that $\mathcal{F}_{mcom}$ allows a committer to open a commitment several times (to the same receiver).

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 $(\texttt{Commit}, sid, cid, P_i, P_j, b)$ from P_i, where $b \in \{0,1\}$, record the tuple (cid, P_i, P_j, b) and send the message $(\texttt{Receipt}, sid, cid, P_i, P_j)$ to P_j and $\mathcal{S}$. 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, proceed as follows: If the tuple (cid, P_i, P_j, b) is recorded then send the message $(\texttt{Open}, sid, cid, P_i, P_j, b)$ to P_j and $\mathcal{S}$. Otherwise, do nothing.

Fig. 2. The Ideal Commitment functionality for multiple commitments

Definition 3. *A protocol is a* universally composable (UC) commitment *protocol if it securely realizes functionality $\mathcal{F}_{com}$. If the protocol securely realizes $\mathcal{F}_{mcom}$ then it is called a* reusable-CRS UC *commitment protocol.*

Remark: On duplicating commitments. Notice that functionalities $\mathcal{F}_{com}$ and $\mathcal{F}_{mcom}$ disallow "copying commitments". That is, assume that party A commits to some value x for party B, and that the commitment protocol in use allows B to commit to the same value x for some party C, before A decommitted to x. Once A decommits to x for B, B will decommit to x for C. Then this protocol does not securely realize $\mathcal{F}_{com}$ or $\mathcal{F}_{mcom}$. This requirement may seem hard to enforce at first, since B can always play "man in the middle" (i.e., forward A's messages to C and C's messages to A.) We enforce it using the unique identities of the parties. (Recall that unique identities are assumed to be provided via an underlying lower-level protocol that also guarantees authenticated communication.)

3 Universally Composable Commitment Schemes

We present two constructions of UC commitment protocols in the common reference string model. The protocol presented in Section 3.1 securely realizes functionality $\mathcal{F}_{com}$, i.e. each part of the public string can only be used for a single commitment. It is based on any trapdoor permutation. The protocol presented in Section 3.2 securely realizes $\mathcal{F}_{mcom}$, i.e. it reuses the public string for multiple commitments. This protocol requires potentially stronger assumptions (either existence of claw-free pairs of trapdoor permutations or alternatively the hardness of discrete log).

3.1 One-Time Common Reference String

The construction in this section works in the common random string model where each part of the commitment can be used only once for a commitment. It is based on the equivocable bit commitment scheme of Di Crescenzo et al. [DIO98], which in turn is a clever modification of Naor's commitment scheme [N91].

Let G be a pseudorandom generator stretching n-bit inputs to $4n$-bit outputs. For security parameter n the receiver in [N91] sends a random $4n$-bit string σ to the sender, who picks a random $r \in \{0,1\}^n$, computes $G(r)$ and returns $G(r)$ or $G(r) \oplus \sigma$ to commit to 0 and 1, respectively. To decommit, the sender transmits b and r. By the pseudorandomness of G the receiver cannot distinguish both cases, and with probability 2^{-2n} over the choice of σ it is impossible to find openings r_0 and r_1 such that $G(r_0) = G(r_1) \oplus \sigma$.

In [DIO98] an equivocable version of Naor's scheme has been proposed. Suppose that σ is not chosen by the receiver, but rather is part of the common random string. Then, if instead we set $\sigma = G(r_0) \oplus G(r_1)$ for random r_0, r_1, and let the sender give $G(r_0)$ to the receiver, it is later easy to open this commitment as 0 with r_0 as well as 1 with r_1 (because $G(r_0) \oplus \sigma = G(r_1)$). On the other hand, choosing σ in that way in indistinguishable from a truly random choice.

We describe a UC bit commitment protocol $\mathsf{UCC}_{\mathsf{OneTime}}$ (for universally composable commitment scheme in the one-time-usable common reference string model). The idea is to use the [DIO98] scheme with a special pseudorandom generator, namely, the Blum-Micali-Yao generator based on any trapdoor permutation [Y82,BM84]. Let KGen denote an efficient algorithm that on input 1^n generates a random public key pk and the trapdoor td. The key pk describes a trapdoor permutation f_{pk} over $\{0,1\}^n$. Let $B(\cdot)$ be a hard core predicate for f_{pk}. Define a pseudorandom generator expanding n bits to $4n$ bits with public description pk by

$$G_{pk}(r) = \Big(f_{pk}^{(3n)}(r),\, B\big(f_{pk}^{(3n-1)}(r)\big), \ldots, B\big(f_{pk}(r)\big), B(r) \Big)$$

where $f_{pk}^{(i)}(r)$ is the i-th fold application of f_{pk} to r. An important feature of this generator is that given the trapdoor td to pk it is easy to recognize images $y \in \{0,1\}^{4n}$ under G_{pk}.

The public random string in our scheme consists of a random $4n$-bit string σ, together with two public keys pk_0, pk_1 describing trapdoor pseudorandom generators G_{pk_0} and G_{pk_1}; both generators stretch n-bit inputs to $4n$-bit output. The public keys pk_0, pk_1 are generated by two independent executions of the key generation algorithm KGen on input 1^n. Denote the corresponding trapdoors by td_0 and td_1, respectively.

In order to commit to a bit $b \in \{0,1\}$, the sender picks a random string $r \in \{0,1\}^n$, computes $G_{pk_b}(r)$, and sets $y = G_{pk_b}(r)$ if $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$. The sender passes y to the receiver. In the decommitment step the sender gives (b, r) to the receiver, who verifies that $y{=}G_{pk_b}(r)$ for $b = 0$ or that $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$. See also Figure 3.

Commitment scheme $\mathsf{UCC}_{\mathsf{OneTime}}$

public string:

σ — random string in $\{0,1\}^{4n}$
pk_0, pk_1 — keys for generators $G_{pk_0}, G_{pk_1} : \{0,1\}^n \to \{0,1\}^{4n}$

commitment for $b \in \{0,1\}$ with SID sid:

compute $G_{pk_b}(r)$ for random $r \in \{0,1\}^n$
set $y = G_{pk_b}(r)$ for $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$
send $(\mathtt{Com}, sid, y)$ to receiver

decommitment for y:

send b, r to receiver
receiver checks $y \overset{?}{=} G_{pk_b}(r)$ for $b = 0$, or $y \overset{?}{=} G_{pk_b}(r) \oplus \sigma$ for $b = 1$

Fig. 3. Commitment Scheme in the One-Time-Usable Common Reference String Model

Clearly, the scheme is computationally hiding and statistically binding. An important observation is that our scheme inherits the equivocability property of [DIO98]. In a simulation we replace σ by $G_{pk_0}(r_0) \oplus G_{pk_1}(r_1)$ and therefore, if we impersonate the sender and transmit $y = G_{pk}(r_0)$ to a receiver, then we can later open this value with 0 by sending r_0 and with 1 via r_1.

Moreover, if we are given a string y^*, e.g., produced by the adversary, and we know the trapdoor td_0 to pk_0, then it is easy to check if y^* is an image under G_{pk_0} and therefore represents a 0-commitment. Unless y^* belongs to G_{pk_0} and, simultaneously, $y^* \oplus \sigma$ belongs to G_{pk_1}, the encapsulated bit is unique and we can extract the correct value with td_0. (We stress, however, that this property will not be directly used in the proof. This is so since there the CRS has a different distribution, so a more sophisticated argument is needed.)

To summarize, our commitment scheme supports equivocability and extraction. The proof of the following theorem appears in [CF01].

Theorem 2. *Protocol $\mathsf{UCC}_{\mathsf{OneTime}}$ securely realizes functionality $\mathcal{F}_{com}$ in the* CRS *model.*

3.2 Reusable Common Reference String

The drawback of the construction in the previous section is that a fresh part of the random string must be reserved for each committed bit. In this section, we overcome this disadvantage under a potentially stronger assumption, namely the existence of claw-free trapdoor permutation pairs. We concentrate on a solution that only works for erasing parties in general, i.e., security is based on the parties' ability to irrevocably erase certain data as soon as they are supposed to. At the end of this section we sketch a solution that does not require data erasures. This solution is based on the Decisional Diffie-Hellman assumption.

Basically, a claw-free trapdoor permutation pair is a pair of trapdoor permutations with a common range such that it is hard to find two elements that are preimages of the same element under the two permutations. More formally, a key generation $\mathsf{KGen}_{\mathsf{claw}}$ outputs a random public key pk_{claw} and a trapdoor td_{claw}. The public key defines permutations $f_{0,pk_{\mathsf{claw}}}, f_{1,pk_{\mathsf{claw}}} : \{0,1\}^n \to \{0,1\}^n$, whereas the secret key describes the inverse functions $f^{-1}_{0,pk_{\mathsf{claw}}}, f^{-1}_{1,pk_{\mathsf{claw}}}$. It should be infeasible to find a claw x_0, x_1 with $f_{0,pk_{\mathsf{claw}}}(x_0) = f_{1,pk_{\mathsf{claw}}}(x_1)$ given only pk_{claw}. For ease of notation we usually omit the keys and write $f_0, f_1, f_0^{-1}, f_1^{-1}$ instead. Claw-free trapdoor permutation pairs exist for example under the assumption that factoring is hard [GMRi88]. For a more formal definition see [G95].

We also utilize an encryption scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ secure against adaptive-chosen ciphertext attacks, i.e., in the notation of [BDPR98] the encryption system should be IND-CCA2. On input 1^n the key generation algorithm KGen returns a public key $pk_{\mathcal{E}}$ and a secret key $sk_{\mathcal{E}}$. An encryption of a message m is given by $c \leftarrow \mathsf{Enc}_{pk_{\mathcal{E}}}(m)$, and the decryption of a ciphertext c is $\mathsf{Dec}_{sk_{\mathcal{E}}}(c)$. It should always hold that $\mathsf{Dec}_{sk_{\mathcal{E}}}(c) = m$ for $c \leftarrow \mathsf{Enc}_{pk_{\mathcal{E}}}(m)$, i.e., the system supports errorless decryption. Again, we abbreviate $\mathsf{Enc}_{pk_{\mathcal{E}}}(\cdot)$ by $\mathsf{Enc}(\cdot)$ and $\mathsf{Dec}_{sk_{\mathcal{E}}}(\cdot)$ by $\mathsf{Dec}(\cdot)$. IND-CCA2 encryption schemes exist for example under the assumption that trapdoor permutations exist [DDN00]. A more efficient solution is based on the decisional Diffie-Hellman assumption [CS98]. Both schemes have errorless decryption.

The commitment scheme $\mathsf{UCC}_{\mathsf{ReUse}}$ (for universally composable commitment with reusable reference string) is displayed in Figure 4. The (reusable) public string contains random public keys pk_{claw} and $pk_{\mathcal{E}}$. For a commitment to a bit b the sender P_i applies the trapdoor permutation f_b to a random $x \in \{0,1\}^n$, computes $c_b \leftarrow \mathsf{Enc}_{pk_{\mathcal{E}}}(x, P_i)$ and $c_{1-b} \leftarrow \mathsf{Enc}_{pk_{\mathcal{E}}}(0^n, P_i)$, and sends the tuple (y, c_0, c_1) with $y = f_b(x)$ to the receiver. The sender is also instructed to erase the randomness for the encryption of $(0^n, P_i)$ before the commitment message is sent. This ciphertext is called a dummy ciphertext.

To open the commitment, the committer P_i sends b, x and the randomness used for encrypting (x, P_i). The receiver P_j verifies that $y = f_b(x)$, that the encryption randomness is consistent with c_b, and that *cid* was never used before in a commitment of P_i to P_j.

We remark that including the sender's identity in the encrypted strings plays an important role in the analysis. Essentially, this precaution prevents a corrupted committer from "copying" a commitment generated by an uncorrupted party.

The fact that the dummy ciphertext is never opened buys us equivocability. Say that the ideal-model simulator knows the trapdoor of the claw-free permutation pair. Then it can compute the pre-images x_0, x_1 of some y under both functions f_0, f_1 and send y as well as encryptions of (x_0, P_i) and (x_1, P_i). To open it as 0 hand $0, x_0$ and the randomness for ciphertext (x_0, P_i) to the receiver and claim to have erased the randomness for the other encryption. For a 1-decommitment send $1, x_1$, the randomness for the encryption of (x_1, P_i) and

Commitment scheme $\mathsf{UCC}_{\mathsf{ReUse}}$

```
public string:
    $pk_{\mathsf{claw}}$ — public key for claw-free trapdoor permutation pair $f_0, f_1$
    $pk_{\mathcal{E}}$ — public key for encryption algorithm $\mathsf{Enc}$

commitment by party $P_i$ to party $P_j$ to $b \in \{0,1\}$ with identifier $sid, cid$:
    compute $y = f_b(x)$ for random $x \in \{0,1\}^n$;
    compute $c_b \leftarrow \mathsf{Enc}(x, P_i)$ with randomness $r_b$;
    compute $c_{1-b} \leftarrow \mathsf{Enc}(0^n, P_i)$ with randomness $r_{1-b}$;
    erase $r_{1-b}$;
    send $(\mathtt{Com}, sid, cid, (y, c_0, c_1))$, and record $(sid, cid, b, x, r_b)$.
    Upon receiving $(\mathtt{Com}, sid, cid, (y, c_0, c_1))$ from $P_i$,
        $P_j$ outputs $(\mathtt{Receipt}, sid, cid, P_i, P_j))$

decommitment for $(P_i, P_j, sid, cid, b, x, r_b)$:
    Send $(\mathtt{Dec}, sid, cid, b, x, r_b)$ to $P_j$.
    Upon receiving $(\mathtt{Dec}, sid, cid, b, x, r_b)$, $P_j$ verifies that $y \overset{?}{=} f_b(x)$,
        that $c_b$ is encryption of $(x, P_i)$ under randomness $r_b$
        where $P_i$ is the committer's identity
        and that $cid$ has not been used with this committer before.
```

Fig. 4. Commitment Scheme with Reusable Reference String

deny to know the randomness for the other ciphertext. If the encryption scheme is secure then it is intractable to distinguish dummy and such fake encryptions. Hence, this procedure is indistinguishable from the actual steps of the honest parties.

Analogously to the extraction procedure for the commitment scheme in the previous section, here an ideal-process adversary can also deduce the bit from an adversarial commitment (y^*, c_0^*, c_1^*) if it knows the secret key of the encryption scheme. Specifically, decrypt c_0^* to obtain (x_0^*, P_i^*); if x_0^* maps to y^* under f_0 then let the guess be 0, else predict 1. This decision is only wrong if the adversary has found a claw, which happens only with negligible probability. The proof of the following theorem appears in [CF01].

Theorem 3. *Protocol $\mathsf{UCC}_{\mathsf{ReUse}}$ securely realizes functionality $\mathcal{F}_{mcom}$ in the* CRS *model.*

A solution for non-erasing parties. The security of the above scheme depends on the ability and good-will of parties to securely erase sensitive data (specifically, to erase the randomness used to generate the dummy ciphertext). A careful look shows that it is possible to avoid the need to erase: It is sufficient to be able to generate a ciphertext without knowing the plaintext. Indeed, it would be enough to enable the parties to obliviously generate a string that is indistinguishable from a ciphertext. Then the honest parties can use this mechanism to produce the dummy ciphertext, while the simulator is still able to place the fake encryption into the commitment. For example, the Cramer-Shoup system in subgroup G of $\mathcal{Z}_p^*$ has this property under the decisional Diffie-Hellman assumption: To generate a dummy ciphertext simply generate four random elements in G.

Relaxing the need for claw-free pairs. The above scheme was presented and proven using any claw-free pair of trapdoor permutations. However, it is easy to see that the claw-free pair can be substituted by chameleon commitments a la [BCC88], thus basing the security of the scheme on the hardness of the discrete logarithm or factoring. Further relaxing the underlying hardness assumptions is an interesting task.

4 Impossibility of UC Commitments in the Plain Model

This section demonstrates that in the plain model there cannot exist universally composable commitment protocols that do not involve third parties in the interaction and allow for successful completion when both the sender and the receiver are honest. This impossibility result holds even under the more liberal requirement that for any real-life adversary and any environment there should be an ideal-model adversary (i.e., under a relaxed definition where the ideal-model simulator may depend on the environment).

We remark that universally composable commitment protocols exist in the plain model if the protocol makes use of third parties, as long as a majority of the parties remain uncorrupted. This follows from a general result in [C00a], where it is shown that practically any functionality can be realized in this setting.

Say that a protocol π between n parties $P_1, \ldots, P_n$ is *bilateral* if all except two parties stay idle and do not transmit messages. A bilateral commitment protocol π is called *terminating* if, with non-negligible probability, the receiver P_j accepts a commitment of the honest sender P_i and outputs $(\texttt{Receipt}, sid, P_i, P_j)$, and moreover if the receiver, upon getting a valid decommitment for a message m and sid from the honest sender, outputs $(\texttt{Open}, sid, P_i, P_j, m)$ with non-negligible probability.

Theorem 4. *There exists no bilateral, terminating protocol π that securely realizes functionality $\mathcal{F}_{com}$ in the plain model. This holds even if the ideal-model adversary $\mathcal{S}$ is allowed to depend on the environment $\mathcal{Z}$.*

Proof. The idea of the proof is as follows. Consider a protocol execution between an adversarially controlled committer P_i and an honest receiver P_j, and assume that the adversary merely sends messages that are generated by the environment. The environment secretly picks a random bit b at the beginning and generates the messages for P_i by running the protocol of the honest committer for b and P_j's answers. In order to simulate this behavior, the ideal-model adversary $\mathcal{S}$ must be able to provide the ideal functionality with a value for the committed bit. For this purpose, the simulator has to "extract" the committed bit from the messages generated by the environment, without the ability to rewind the environment. However, as will be seen below, if the commitment scheme allows the simulator to successfully extract the committed bit, then the commitment is not secure in the first place (in the sense that a corrupted receiver can obtain the value of the committed bit from interacting with an honest committer).

More precisely, let the bilateral protocol π take place between the sender P_i and the receiver P_j. Consider the following environment $\mathcal{Z}$ and real-life adversary $\mathcal{A}$. At the outset of the execution the adversary $\mathcal{A}$ corrupts the committer P_i. Then, in the sequel, $\mathcal{A}$ has the corrupted committer send every message it receives from $\mathcal{Z}$, and reports any reply received by P_j to $\mathcal{Z}$. The environment $\mathcal{Z}$ secretly picks a random bit b and follows the program of the honest sender to commit to b, as specified by π. Once the the honest receiver has acknowledged the receipt of a commitment, $\mathcal{Z}$ lets $\mathcal{A}$ decommit to b by following protocol π. Once the receiver outputs $(\texttt{Open}, sid, P_i, P_j, b')$, $\mathcal{Z}$ outputs 1 if $b = b'$ and outputs 0 otherwise.

Formally, suppose that there is an ideal-model adversary $\mathcal{S}$ such that $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{com},\mathcal{S},\mathcal{Z}}$. Then we construct a new environment $\mathcal{Z}'$ and a new real-life adversary $\mathcal{A}'$ for which there is no appropriate ideal-model adversary for π. This time, $\mathcal{A}'$ corrupts the receiver P_j at the beginning. During the execution $\mathcal{A}'$ obtains messages form the honest committer P_i and feeds these messages into a virtual copy of $\mathcal{S}$. The answers of $\mathcal{S}$, made on behalf of an honest receiver, are forwarded to P_i in the name of the corrupted party P_j. At some point, $\mathcal{S}$ creates a submission $(\texttt{Commit}, sid, P_i, P_j, b')$ to $\mathcal{F}_{com}$; the adversary $\mathcal{A}'$ outputs b' and halts. If $\mathcal{S}$ halts without creating such a submission then $\mathcal{A}'$ outputs a random bit and halts.

The environment $\mathcal{Z}'$ instructs the honest party P_i to commit to a randomly chosen secret bit b. (No decommitment is ever carried out.) Conclusively, $\mathcal{Z}'$ outputs 1 iff the adversary's output b' satisfies $b = b'$.

By the termination property, we obtain from the virtual simulator $\mathcal{S}$ a bit b' with non-negligible probability. This bit is a good approximation of the actual bit b, since $\mathcal{S}$ simulates the real protocol π except with negligible error. Hence, the guess of $\mathcal{A}'$ for b is correct with 1/2 plus a non-negligible probability. But for a putative ideal-model adversary $\mathcal{S}'$ predicting this bit b with more than non-negligible probability over 1/2 is impossible, since the view of $\mathcal{S}'$ in the ideal process is statistically independent from the bit b. (Recall that the commitment to b is never opened).

5 Application to Zero-Knowledge

In order to exemplify the power of UC commitments we show how they can be used to construct simple Zero-Knowledge (ZK) protocols with strong security properties. Specifically, we formulate an ideal functionality, $\mathcal{F}_{zk}$, that captures the notion of Zero-Knowledge in a very strong sense. (In fact, $\mathcal{F}_{zk}$ implies concurrent and non-malleable Zero-Knowledge proofs of knowledge.) We then show that in the $\mathcal{F}_{com}$-hybrid model (i.e., in a model with ideal access to $\mathcal{F}_{com}$) there is a 3-round protocol that securely realizes $\mathcal{F}_{zk}$ with respect to any NP relation. Using the composition theorem of [c00a], we can replace $\mathcal{F}_{com}$ with any UC commitment protocol. (This of course requires using the CRS model, unless we involve third parties in the interaction. Also, using functionality $\mathcal{F}_{mcom}$ instead of $\mathcal{F}_{com}$ is possible and results in a more efficient use of the common string.)

Functionality $\mathcal{F}_{zk}$, described in Figure 5, is parameterized by a binary relation $R(x, w)$. It first waits to receive a message $(\texttt{verifier}, id, P_i, P_j, x)$ from some party P_i, interpreted as saying that P_i wants P_j to prove to P_i that it knows a value w such that $R(x, w)$ holds. Next, $\mathcal{F}_{zk}$ waits for P_j to explicitly provide a value w, and notifies P_i whether $R(x, w)$ holds. (Notice that the adversary is notified whenever either the prover or the verifier starts an interaction. It is also notified whether the verifier accepts. This represents the fact that ZK is not traditionally meant to hide this information.)

Functionality $\mathcal{F}_{zk}$

$\mathcal{F}_{zk}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$. The functionality is parameterized by a binary relation R.

1. Wait to receive a value $(\texttt{verifier}, id, P_i, P_j, x)$ from some party P_i. Once such a value is received, send $(\texttt{verifier}, id, P_i, P_j, x)$ to $\mathcal{S}$, and ignore all subsequent $(\texttt{verifier...})$ values.
2. Upon receipt of a value $(\texttt{prover}, id, P_j, P_i, x', w)$ from P_j, let $v = 1$ if $x = x'$ and $R(x, w)$ holds, and $v = 0$ otherwise. Send (id, v) to P_i and $\mathcal{S}$, and halt.

Fig. 5. The Zero-Knowledge functionality, $\mathcal{F}_{zk}$

We demonstrate a protocol for securely realizing $\mathcal{F}_{zk}^R$ for any NP relation R. The protocol is a known one: It consists of n parallel repetitions of the 3-round protocol of Blum for graph Hamiltonicity, where the provers commitments are replaced by invocations of $\mathcal{F}_{com}$. The protocol (in the $\mathcal{F}_{com}$-hybrid model) is presented in Figure 6.

We remark that in the $\mathcal{F}_{com}$-hybrid model the protocol securely realizes $\mathcal{F}_{zk}$ *without any computational assumptions,* and even if the adversary and the environment are computationally unbounded. (Of course, in order to securely realize $\mathcal{F}_{com}$ the adversary and environment must be computationally bounded.) Also, in the $\mathcal{F}_{com}$-hybrid model there is no need in a common reference string. That is, the CRS model is needed only for realizing $\mathcal{F}_{com}$.

Let $\mathcal{F}_{zk}^H$ denote functionality $\mathcal{F}_{zk}$ parameterized by the Hamiltonicity relation H. (I.e., $H(G, h) = 1$ iff h is a Hamiltonian cycle in graph G.) The following theorem is proven in [CF01].

Theorem 5. *Protocol* HC *securely realizes $\mathcal{F}_{zk}^H$ in the $\mathcal{F}_{com}$-hybrid model.*

Acknowledgements. We thank Yehuda Lindell for suggesting to use non-malleable encryptions for achieving non-malleability of commitments in the common reference string model. This idea underlies our scheme that allows to reuse the common string for multiple commitments. (The same idea was independently suggested in [DKOS01].)

Protocol Hamilton-Cycle (HC)

1. Given input `(Prover,`id, P, V, G, h`)`, where G is a graph over nodes $1, ..., n$, the prover P proceeds as follows. If h is not a Hamiltonian cycle in G, then P sends a message `reject` to V. Otherwise, P proceeds as follows for $k = 1, ..., n$:
   a) Choose a random permutation π_k over $[n]$.
   b) Using $\mathcal{F}_{com}$, commit to the edges of the permuted graph. That is, for each $(i, j) \in [n]^2$ send `(Commit,`$(i, j, k), P, V, e$`)` to $\mathcal{F}_{com}$, where $e = 1$ if there is an edge between $\pi_k(i)$ and $\pi_k(j)$ in G, and $e = 0$ otherwise.
   c) Using $\mathcal{F}_{com}$, commit to the permutation π_k. That is, for $l = 1, ..., L$ send `(Commit,`$(l, k), P, V, p_l$`)` to $\mathcal{F}_{com}$ where $p_1, ..., p_L$ is a representation of π_k in some agreed format.
2. Given input `(Verifier,`id, V, P, G`)`, the verifier V waits to receive either `reject` from P, or `(Receipt,`$(i, j, k), P, V$`)` and `(Receipt,`$(l, k), P, V$`)` from $\mathcal{F}_{com}$, for $i, j, k = 1, ..., n$ and $l = 1, ..., L$. If `reject` is received, then V output 0 and halts. Otherwise, once all the `(Receipt,...)` messages are received V randomly chooses n bits $c_1, ..., c_n$ and sends to P.
3. Upon receiving $c_1, ..., c_n$ from V, P proceeds as follows for $k = 1, ..., n$:
   a) If $c_k = 0$ then send `(Open,`$(i, j, k), P, V$`)` and `(Open,`$(l, k), P, V$`)` to $\mathcal{F}_{com}$ for all $i, j = 1, ..., n$ and $l = 1, ..., L$.
   b) If $c_k = 1$ then send `(Open,`$(i, j, k), P, V$`)` to $\mathcal{F}_{com}$ for all $i, j = 1, ..., n$ such that the edge $\pi_k(i), \pi_k(j)$ is in the cycle h.
4. Upon receiving the appropriate `(Open,...)` messages from $\mathcal{F}_{com}$, the verifier V verifies that for all k such that $c_k = 0$ the opened edges agree with the input graph G and the opened permutation π_k, and for all k such that $c_k = 1$ the opened edges are all 1 and form a cycle. If verification succeeds then output 1, otherwise output 0.

Fig. 6. The protocol for proving Hamiltonicity in the $\mathcal{F}_{com}$-hybrid model

References

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

[B99] D. Beaver, "Adaptive Zero-Knowledge and Computational Equivocation", *28th Symposium on Theory of Computing (STOC),* ACM, 1996.

[BBM00] M. Bellare, A. Boldyreva and S. Micali, "Public-Key Encryption in a Multi-user Setting: Security Proofs and Improvements," *Eurocrypt 2000, pp. 259–274, Springer LNCS 1807,* 2000.

[BDJR97] M Bellare, A. Desai, E. Jokipii and P. Rogaway, "A concrete security treatment of symmetric encryption: Analysis of the DES modes of operations," 38*th Annual Symp. on Foundations of Computer Science (FOCS),* IEEE, 1997.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway, "Relations among notions of security for public-key encryption schemes", *CRYPTO '98,* 1998, pp. 26-40.

[BM84] M.Blum, S.Micali: How to Generate Cryptographically Strong Sequences of Pseudorandom Bits, *SIAM Journal on Computation, Vol. 13, pp. 850–864*, 1984.

[BCC88] G. Brassard, D. Chaum and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *JCSS*, Vol. 37, No. 2, pages 156–189, 1988.

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

[C00a] R. Canetti, "A unified framework for analyzing security of Protocols", manuscript, 2000. Available at http://eprint.iacr.org/2000/067.

[CF01] R. Canetti and M. Fischlin, "Universally Composable Commitments". Available at http://eprint.iacr.org/2001.

[CS98] R. Cramer and V. Shoup, "A paractical public-key cryptosystem provably secure against adaptive chosen ciphertext attack", *CRYPTO '98,* 1998.

[D89] I. Damgard, On the existence of bit commitment schemes and zero-knowledge proofs, Advances in Cryptology - Crypto '89, pp. 17–29, 1989.

[D00] I. Damgard. Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. *Eurocrypt 00*, LNCS, 2000.

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

[DKOS01] G. Di Crecenzo, J. Katz, R. Ostrovsky and A. Smith. Efficient and Perfectly-Hiding Non-Interactive, Non-Malleable Commitment. *Eurocrypt '01,* 2001.

[DM00] Y. Dodis and S. Micali, "Secure Computation", *CRYPTO '00,* 2000.

[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),* ACM, 1991.

[DNRS99] C. Dwork, M. Naor, O. Reingold, and L. Stockmeyer. Magic functions. In *40th Annual Symposium on Foundations of Computer Science*, pages 523–534. IEEE, 1999.

[FS90] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd STOC*, pages 416–426, 1990.

[FF00] M. Fischlin and R. Fischlin, "Efficient non-malleable commitment schemes", *CRYPTO '00, LNCS 1880*, 2000, pp. 413-428.

[GHY88] Z. Galil, S. Haber and M. Yung, Cryptographic computation: Secure faut-tolerant protocols and the public-key model, *CRYPTO '87, LNCS 293*, Springer-Verlag, 1988, pp. 135-155.

[G95] O. Goldreich, *"Foundations of Cryptography (Fragments of a book)"*, Weizmann Inst. of Science, 1995. (Avaliable at http://philby.ucsd.edu)

[G98] O. Goldreich. *"Secure Multi-Party Computation"*, 1998. (Avaliable at http://philby.ucsd.edu)

[GMW91] O. Goldreich, S. Micali and A. Wigderson, "Proofs that yield nothing but their validity or All Languages in NP Have Zero-Knowledge Proof Systems", *Journal of the ACM,* Vol 38, No. 1, ACM, 1991, pp. 691–729. Preliminary version in *27th Symp. on Foundations of Computer Science (FOCS),* IEEE, 1986, pp. 174-187.

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

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

[GMRa89] S. Goldwasser, S. Micali and C. Rackoff, “The Knowledge Complexity of Interactive Proof Systems”, *SIAM Journal on Comput.,* Vol. 18, No. 1, 1989, pp. 186-208.

[GMRi88] S.Goldwasser, S.Micali, R.Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks, *SIAM Journal on Computing, Vol. 17, No. 2, pp. 281–308*, 1988.

[L00] Y. Lindell, private communication, 2000.

[MR91] S. Micali and P. Rogaway, “Secure Computation”, unpublished manuscript, 1992. Preliminary version in *CRYPTO ’91, LNCS 576,* Springer-Verlag, 1991.

[N91] M.Naor: Bit Commitment Using Pseudo-Randomness, *Journal of Cryptology, vol. 4, pp. 151–158*, 1991.

[NOVY92] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung, Perfect zero-knowledge arguments for NP can be based on general complexity assumptions, Advances in Cryptology - Crypto ’92, pp. 196–214, 1992.

[PW94] B. Pfitzmann and M. Waidner, “A general framework for formal notions of secure systems”, Hildesheimer Informatik-Berichte 11/94, Universität Hildesheim, 1994. Available at http://www.semper.org/sirene/lit.

[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, 2001. See also IBM Research Report RZ 3304 (#93350), IBM Research, Zurich, December 2000.

[RS91] C. Rackoff and D. Simon, “Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack”, *CRYPTO ’91,* 1991.

[Y82] A. Yao, Theory and applications of trapdoor functions, In *Proc. 23rd Annual Symp. on Foundations of Computer Science (FOCS),* pages 80–91. IEEE, 1982.