Universally Composable Commitments

Ran Canetti* Marc Fischlin†

July 10, 2001

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.

*IBM T.J. Watson Research Center. Email: canetti@watson.ibm.com.

†Goethe-University of Frankfurt; part of this work done while visiting IBM T.J. Watson Research Center. Email: marc@mi.informatik.uni-frankfurt.de.

Contents

1 Introduction

Commitment is one of the most basic and useful cryptographic primitives. On top of being intriguing by itself, it is an essential building block in many cryptographic protocols, such as Zero-Knowledge protocols (e.g., [GMW91, BCC88, D89]), general function evaluation protocols (e.g., [GMW87, GHY88, G98]), contract-signing and electronic commerce, and more. Indeed, commitment protocols have been studied extensively in the past two decades (e.g., [B82, N91, DDN00, NOVY92, B96, 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 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* [B96]. 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, before the $\mathcal{C}$-commitment is opened.

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, uses a different copy of the CRS for each copy of the protocol. Said otherwise, this construction 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 single, 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 [C01]. 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 [C01].

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 [C01] 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. (Allowing the environment to freely interact with the adversary is crucial for the composability properties ot he definition.) 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 [c01] 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 [c01]. 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 $\mathcal{F}_{\text{COM}}$ that captures the expected behavior of an "ideal commitment service". Universally Composable (UC) commitment protocols are defined to be those that securely realize $\mathcal{F}_{\text{COM}}$. Our formulation of $\mathcal{F}_{\text{COM}}$ is a straightforward transcription of the "envelope paradigm": $\mathcal{F}_{\text{COM}}$ 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, $\mathcal{F}_{\text{COM}}$ 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, $\mathcal{F}_{\text{COM}}$ sends the recorded value x to R, and halts. (Some other variants of $\mathcal{F}_{\text{COM}}$ 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 $\mathcal{F}_{\text{COM}}$, 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, applying the composition theorem to protocols that securely realize $\mathcal{F}_{\text{COM}}$ as described above is highly wasteful of the reference string. In order to capture protocols where multiple commitments may use the same short reference string we formulate a natural extension of $\mathcal{F}_{\text{COM}}$ that handles multiple commitment requests. We call this extension $\mathcal{F}_{\text{MCOM}}$.

We remark that the definition allows UC commitment protocols to be computationally secret and computationally binding only, achieving neither property unconditionally. In fact, one of the constructions presented here merely attains this computational security level but is indeed universally composable.

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 $\mathcal{F}_{\text{COM}}$ 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 [C01] 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 $\mathcal{F}_{\text{COM}}$) 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 $\mathcal{F}_{\text{MCOM}}$.

In the second protocol (that securely realizes $\mathcal{F}_{\text{MCOM}}$), 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 string commitment protocols 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 [C01].

Organization. Section 2 shortly reviews the general framework of [C01] and presents the ideal commitment functionalities $\mathcal{F}_{\text{COM}}$ and $\mathcal{F}_{\text{MCOM}}$. Section 3 demonstrates that functionalities $\mathcal{F}_{\text{COM}}$ and $\mathcal{F}_{\text{MCOM}}$ cannot be realized in the plain model by a two-party protocol. Section 4 presents and proves security of the protocols that securely realize $\mathcal{F}_{\text{COM}}$ and $\mathcal{F}_{\text{MCOM}}$. Section 5 presents the application to constructing Zero-Knowledge protocols.

2 Defining UC commitments

Section 2.1 shortly summarizes the relevant parts of the general framework of [C01], including the general framework for defining security and the composition theorem. Section 2.1.2 defines the CRS model. Section 2.2 defines the ideal commitment functionalities, $\mathcal{F}_{\text{COM}}$ and $\mathcal{F}_{\text{MCOM}}$.

2.1 The general framework

As sketched in the Introduction, protocols that securely carry out a given task (or, protocol problem) are defined in three steps, as follows. First, the process of executing a protocol in the presence of an adversary and in a given computational environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. In the ideal process the parties do not communicate with each other. Instead they have access to an "ideal functionality", which is essentially an incorruptible "trusted party" that is programmed to capture the desired requirements from the task at hand. A protocol is said to securely realize a task if the process of running the protocol "emulates" the ideal process for that task. In the rest of this subsection we overview the model for protocol execution (called the *real-life model*), the ideal process, and the notion of protocol emulation.

Protocol syntax. Following [GMRa89, G95], a protocol is represented as a system of interactive Turing machines (ITMs), where each ITM represents the program to be run within a different party. 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. [C01] discusses several models of computation. We concentrate on a model where 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. Any number of parties can be corrupted. Finally, the adversary and environment are restricted to probabilistic polynomial time (or, "feasible") computation.

[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 [C01] for further discussion.

Protocol execution in the real-life model. We sketch the process 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. (It is stressed that only messages that were generated by parties can be delivered. The adversary cannot modify or duplicate messages.) 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 write information on the input tape of either one of the parties or of the adversary. That entity is activated once the activation of the environment is complete (i,e, once the environment enters a special waiting state.) If no input tape was written into then the execution halts. Once a party completes its activation the environment is activated again. Whenever the adversary delivers a message to some party P in some activation, then this party is activated next. Once P's activation is complete, the environment is activated again. If in some activation the adversary delivers no messages then the environment is activated as soon as the adversary completes its activation. Notice that this mechanism allows environment and the adversary to exchange information freely using their input and output tapes, between each two activations of some party. The output of the protocol execution is the output of $\mathcal{Z}$. (Without loss of generality $\mathcal{Z}$ outputs a single bit.)

Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \vec{r})$ 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 $\vec{r} = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1 \ldots r_n$ 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 describing $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \vec{r})$ when $\vec{r}$ 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 from $\mathcal{F}$ to the parties. It can also corrupt dummy parties, learn the information they know, and control their future activities.

The order of events in the ideal process is the same as in the real-life process, with the exception that here, if a dummy party $\tilde{P}$ is activated by an input value coming from the environment then (this value is copied to the outgoing communication tape of $\tilde{P}$ and) the ideal functionality is activated next. Once the ideal functionality completes its activation (having perhaps sent messages to the adversary or dummy parties), $\tilde{P}$ is activated one again. 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.

Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z, \vec{r})$ 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 $\vec{r} = r_{\mathcal{Z}}, r_{\mathcal{S}}, r_{\mathcal{F}}$ 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 $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z, \vec{r})$ when $\vec{r}$ 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 interacting 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 ([c01]) *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* $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$.

2.1.1 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 additions. On top of 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. It is stressed that the environment does not have direct access to the copies of $\mathcal{F}$. (Indeed, here the security definition will require that the environment will be unable to tell whether it is interacting with the real-life model or the hybrid model.)

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:

Theorem 3 *Let $\mathcal{F}$ be an ideal functionality. Let π be a protocol in the $\mathcal{F}$-hybrid model, and let ρ be a protocol that securely realizes $\mathcal{F}$. Then for any real-life adversary $\mathcal{A}$ there exists a hybrid-model adversary $\mathcal{H}$ such that for any environment machine $\mathcal{Z}$ we have* $\text{REAL}_{\pi^\rho,\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},\mathcal{Z}}$.

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 4 ([c01]) *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 $\mathcal{G}$.*

2.1.2 The common reference string (CRS) model.

In the common reference string (CRS) 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.) In the present framework we re-cast the CRS model framework as a hybrid model with ideal access to a functionality $\mathcal{F}_{\text{CRS}}$, that is parameterized by a distribution D and described in Figure 1 below.

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

$\mathcal{F}_{\text{CRS}}$ proceeds as follows, when parameterized by a distribution D.

1. When activated for the first time on input $(\texttt{value}, sid)$, choose a value $d \stackrel{\text{R}}{\leftarrow} D$ and send d back to the activating party. In each other activation return the value d to the activating party.

Figure 1: The Common Reference String functionality

Notice that this formalization has the usual properties of the CRS model. Specifically:

- In the real-life model of computation the parties have access to a common and public string that is chosen in advance according to some distribution (specified by the protocol run by the parties).
- In the ideal process for some functionality (say, for $\mathcal{F}_{\mathrm{COM}}$ defined below) there is no use of the random string. Consequently an ideal process adversary that operates by simulating a real-life adversary may play the role of $\mathcal{F}_{\mathrm{CRS}}$ for the simulated adversary. This means that the ideal process adversary may choose the common string in any way it wishes.

Furthermore, since the ideal process makes no use of the random string, the validity of the ideal process is not affected by the fact that the protocol runs in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid model. We are thus guaranteed that our notion of security remains valid.

Protocol composition in the CRS model. Some words of clarification are in order with respect to the composition theorem in the CRS model. It is stressed that each copy of protocol ρ within the composed protocol π^ρ should have its own copy of the reference string, i.e. a separate instance of $\mathcal{F}_{\mathrm{CRS}}$, (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 security requirements from 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 $\mathcal{F}_{\mathrm{COM}}$ that handles a single commitment-decommitment process, and functionality $\mathcal{F}_{\mathrm{MCOM}}$ that handles multiple such processes. Recall that the advantage of $\mathcal{F}_{\mathrm{MCOM}}$ over $\mathcal{F}_{\mathrm{COM}}$ is that protocols that securely realize $\mathcal{F}_{\mathrm{MCOM}}$ may use the same short common string for multiple commitments. (In contrast, applying the composition theorem to protocols that realize $\mathcal{F}_{\mathrm{COM}}$ requires using a different common string for each commitment.) Indeed, realizing $\mathcal{F}_{\mathrm{MCOM}}$ is more challenging than realizing $\mathcal{F}_{\mathrm{COM}}$. Some further discussion on the functionalities and possible variants appears in Section 2.2.1.

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 $\mathcal{F}_{\mathrm{COM}}$ and $\mathcal{F}_{\mathrm{MCOM}}$ 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 $\mathcal{F}_{\mathrm{COM}}$, described in Figure 2, proceeds as follows. The commitment phase is modeled by having $\mathcal{F}_{\mathrm{COM}}$ receive a value $(\mathtt{Commit}, sid, P_i, P_j, b)$, from some party P_i (the committer). Here *sid* is a Session ID used to distinguish among various copies of $\mathcal{F}_{\mathrm{COM}}$, P_j is the identity of another party (the receiver), and $b \in \{0,1\}$ is the value committed to. In response, $\mathcal{F}_{\mathrm{COM}}$ 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 $(\mathtt{Receipt}, sid, P_i, P_j)$ to P_j and $\mathcal{S}$. The opening phase is initiated by the committer sending a value $(\mathtt{Open}, sid, P_i, P_j)$ to $\mathcal{F}_{\mathrm{COM}}$. In response, $\mathcal{F}_{\mathrm{COM}}$ hands the value $(\mathtt{Open}, sid, P_i, P_j, b)$ to P_j and $\mathcal{S}$.

Functionality $\mathcal{F}_{\mathrm{MCOM}}$, presented in Figure 3, essentially mimics the operation of $\mathcal{F}_{\mathrm{COM}}$ for multiple commitments. In addition to the session ID *sid*, functionality $\mathcal{F}_{\mathrm{MCOM}}$ uses an additional identifier, a

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

$\mathcal{F}_{\text{COM}}$ 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 `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.

Figure 2: The Ideal Commitment functionality for a single commitment

Commitment ID *cid*, that is used to distinguish among the different commitments that take place within a single run of $\mathcal{F}_{\text{MCOM}}$. 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 $\mathcal{F}_{\text{MCOM}}$ allows a committer to open a commitment several times (to the same receiver).

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

$\mathcal{F}_{\text{MCOM}}$ 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.

Figure 3: The Ideal Commitment functionality for multiple commitments

Definition 5 *A protocol is a* universally composable (UC) commitment *protocol if it securely realizes functionality $\mathcal{F}_{\text{COM}}$. If the protocol securely realizes $\mathcal{F}_{\text{MCOM}}$ then it is called a* reusable-CRS UC *commitment protocol.*

2.2.1 Discussion

On duplicating commitments. Notice that functionalities $\mathcal{F}_{\text{COM}}$ and $\mathcal{F}_{\text{MCOM}}$ 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 $\mathcal{F}_{\text{COM}}$ or $\mathcal{F}_{\text{MCOM}}$. 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.)

On the difference between $\mathcal{F}_{\mathrm{COM}}$ and $\mathcal{F}_{\mathrm{MCOM}}$. Securely realizing $\mathcal{F}_{\mathrm{MCOM}}$ is considerably more demanding than securely realizing $\mathcal{F}_{\mathrm{COM}}$. In particular, a protocol that securely realizes $\mathcal{F}_{\mathrm{COM}}$ does not need to explicitly guarantee "independence" (or, "non-malleability") among different commitments: this independence is taken care of by the general composition theorem. In contrast, in order to securely realize $\mathcal{F}_{\mathrm{MCOM}}$ a protocol has to explicitly guarantee independence among the different commitments handled by the same copy of $\mathcal{F}_{\mathrm{MCOM}}$. Independence from other copies of $\mathcal{F}_{\mathrm{MCOM}}$ and from other protocols is guaranteed via the general composition theorem.

Some variants of $\mathcal{F}_{\mathrm{COM}}$ and $\mathcal{F}_{\mathrm{MCOM}}$. Functionalities $\mathcal{F}_{\mathrm{COM}}$ and $\mathcal{F}_{\mathrm{MCOM}}$ capture one standard variant of commitment protocols. Other variants are possible, providing different security properties. We sketch a few:

1. The functionalities can be modified so that the adversary does not receive the opened value x. This captures the concern that the opening of the commitment should be available only to the receiver.
2. The functionalities can be modified so that the receiver of the commitment provides the functionality with acknowledgments for obtaining the commitment and the opening, and the functionality forwards these acknowledgments to the committer. This may be useful in cases where the committer has to make sure that the receiver accepted the commitment and/or the opening.
3. The functionalities can be modified so that the adversary receives no messages whatsoever. This captures the concern that the adversary does not learn whether a commitment protocol took place at all. (This requirement has a flavor of protection against traffic analysis.)
4. Functionalities $\mathcal{F}_{\mathrm{COM}}$ and $\mathcal{F}_{\mathrm{MCOM}}$ don't specify an "error message," to be generated by the receiver, in case where the committer provides the receiver with an invalid opening of some committed value. (Instead, the current specification instructs the receiver to ignore invalid decommitments.) An alternative formulation would instruct the functionality to notify the receiver when it receives an invalid `(Open,...)` message from the committer.

3 Impossibility of UC Commitments in the Plain Model

This section demonstrates that in the plain model (i.e., without access to some ideal functionality) 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 (namely, servers), as long as a majority of the servers remain uncorrupted. This follows from a general result in [c01], 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 honest 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 honest 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 6 *There exist no bilateral, terminating protocol π that securely realizes functionality $\mathcal{F}_{\text{COM}}$ 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, and relays to the environment the messages sent to P_i. 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. In other words, 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.

Since the receiver outputs a receipt before the decommitment starts, an ideal-model adversary $\mathcal{S}$ for the pair $\mathcal{A}, \mathcal{Z}$ must send $(\texttt{Commit}, sid, P_i, P_j, b')$ to $\mathcal{F}_{\text{COM}}$ *before* learning the bit b in the decommitment step. However, the honest receiver outputs the bit b' it gets in the opening step from $\mathcal{F}_{\text{COM}}$, and this implies that a successful $\mathcal{S}$ must come up with the true bit b already at the commitment step, which contradicts the secrecy of the commitment protocol.

Formally, suppose that there is an ideal-model adversary $\mathcal{S}$ such that $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{\text{COM}},\mathcal{S},\mathcal{Z}}$. 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 $\mathcal{F}_{\text{COM}}$; 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). □

4 UC Commitment schemes in the CRS model

We present two basic approaches for constructions of UC commitment protocols in the common reference string (CRS) model. The protocol presented in Section 4.1 securely realizes functionality $\mathcal{F}_{\mathrm{COM}}$, 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 4.2 securely realizes $\mathcal{F}_{\mathrm{MCOM}}$, i.e., it reuses the public string for multiple commitments. This protocol requires potentially stronger assumptions (either the existence of claw-free pairs of trapdoor permutations or alternatively secure encryption and non-interactive perfectly-secret trapdoor commitments). Nonetheless, in the presence of an adaptive adversary this solution only works if the honest players faithfully erase some parts of their internal randomness. In Section 4.3 we give sufficient conditions under which data erasure can be avoided, and show that these conditions can be met under the Decisional Diffie-Hellman assumption for example.

4.1 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 for only one 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].

4.1.1 Preliminaries

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 the two 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.

4.1.2 Description of Commitment Scheme

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 that has a trapdoor property. Specifically, we use the Blum-Micali-Yao generator but with trapdoor permutations instead of one-way permutations [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

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

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 tell whether a given $y \in \{0,1\}^{4n}$ is in the range of 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.

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 $(\texttt{Com}, sid, y)$ to the receiver

Upon receiving $(\texttt{Com}, sid, y)$ from P_i, P_j outputs $(\texttt{Receipt}, sid, cid, P_i, P_j)$

decommitment for y:

send b, r to the receiver
receiver checks $y \stackrel{?}{=} G_{pk_b}(r)$ for $b = 0$, or $y \stackrel{?}{=} G_{pk_b}(r) \oplus \sigma$ for $b = 1$.
If the verification succeeds then P_j outputs $(\texttt{Open}, sid, P_i, P_j, b)$.

Figure 4: Commitment Scheme in the One-Time-Usable Common Reference String Model

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 4.

4.1.3 Basic Properties

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 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^* generated 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 the range of G_{pk_0} and, simultaneously, $y^* \oplus \sigma$ belongs to the range of 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.)

4.1.4 Security

To summarize, our commitment scheme supports equivocability and extraction. We are now ready to prove that the protocol securely realizes functionality $\mathcal{F}_{\text{COM}}$:

Theorem 7 *Protocol $\mathsf{UCC}_{\mathsf{OneTime}}$ securely realizes functionality $\mathcal{F}_{\text{COM}}$ in the* CRS *model.*

Proof: We describe the ideal-model adversary $\mathcal{S}$. This adversary runs an execution with the environment $\mathcal{Z}$ and, in parallel, simulates a virtual copy of the real-life adversary $\mathcal{A}$ in a black-box way. That is, $\mathcal{S}$ acts as an interface between $\mathcal{A}$ and $\mathcal{Z}$ by imitating a copy of a real execution of π for $\mathcal{A}$, incorporating $\mathcal{Z}$'s ideal-model interactions and vice versa forwarding $\mathcal{A}$'s messages to $\mathcal{Z}$. More precisely,

1. At the outset the simulator $\mathcal{S}$ prepares σ by selecting key pairs $(pk_0, td_0) \leftarrow \mathsf{KGen}(1^n)$ and $(pk_1, td_1) \leftarrow \mathsf{KGen}(1^n)$ and setting $\sigma = G_{pk_0}(r_0) \oplus G_{pk_1}(r_1)$ for random $r_0, r_1 \in \{0,1\}^n$. We call this a *fake* string σ with respect to preselected values pk_0, pk_1, $G_{pk_0}(r_0)$ and $G_{pk_1}(r_1)$. Next, $\mathcal{S}$ starts the simulation of $\mathcal{A}$ and the execution with $\mathcal{Z}$ on the fake string σ and pk_0, pk_1.

2. If at some point in the execution the environment $\mathcal{Z}$ writes a message $(\texttt{Commit}, sid, P_i, P_j, b)$ on the tape of the *uncorrupted* party $\tilde{P}_i$, and $\tilde{P}_i$ copies this to the functionality $\mathcal{F}_{\text{COM}}$, then the ideal-model simulator —who cannot read the actual bit, but is informed about the commitment by receiving $(\texttt{Receipt}, sid, P_i, P_j)$— tells $\mathcal{A}$ that P_i has sent $y = G_{pk_0}(r_0)$ to P_j.

3. If at some point in the execution $\mathcal{Z}$ instructs an *uncorrupted* party $\tilde{P}_i$ to decommit and this party has previously correctly committed to some secret bit b. Then the ideal-model adversary $\mathcal{S}$ must have sent the value $y = G_{pk_0}(r_0)$ on behalf of P_i in the black-box simulation of $\mathcal{A}$. In the ideal model, $\mathcal{S}$ now learns b from $\tilde{P}_i$ via $\mathcal{F}_{\text{COM}}$ and opens y in the simulation of $\mathcal{A}$ accordingly, using the equivocability property.

4. If the simulated adversary $\mathcal{A}$ lets some *corrupted* party P_i send $(\texttt{Com}, sid, y^*)$ to an honest party P_j then $\mathcal{S}$ verifies with the help of the trapdoor td_0 whether y^* is in the range of $G_{pk_0}(\cdot)$ or not. If so, $\mathcal{S}$ sends a message $(\texttt{Commit}, sid, P_i, P_j, 0)$ on behalf of the party to the functionality; else $\mathcal{S}$ sends $(\texttt{Commit}, sid, P_i, P_j, 1)$ to $\mathcal{F}_{\text{COM}}$.

5. If $\mathcal{A}$ tells a *corrupted* party P_i to open a valid commitment y^* correctly with bit b^*, then $\mathcal{S}$ compares b^* to the previously extracted bit and stops if they differ; otherwise $\mathcal{S}$ sends $(\texttt{Open}, sid, P_i, P_j)$ in the name of the party to $\mathcal{F}_{\text{COM}}$. If P_i is supposed to decommit incorrectly, then $\mathcal{S}$ also sends an incorrect opening to the functionality.

6. Whenever the simulated $\mathcal{A}$ demands to corrupt a party, $\mathcal{S}$ corrupts this party in the ideal model and learns all internal information of the party. Now $\mathcal{S}$ first adapts possible decommitment information about a previously given but yet unopened commitment of this party, like in the case of an honest party decommitting. After this, $\mathcal{S}$ gives all this adjusted information to $\mathcal{A}$.

In order to show that the environment's output in the real-life model is indistinguishable from its output in the ideal-process, we consider the following three random variables:[2]

Real/Genuine: The output of $\mathcal{Z}$ in a real-life execution with parties running the protocol and adversary $\mathcal{A}$. This amounts to choosing a uniformly distributed σ and random pk_0, pk_1 by running KGen and publishing this as the public string; then run the protocol in the real-life model with $\mathcal{A}$ and $\mathcal{Z}$ on this string.

Real/Fake: The output of $\mathcal{Z}$ from the following interaction. Choose a fake string σ together with random pk_0, pk_1, like the simulator, involving preselected values $G_{pk_0}(r_0)$ and $G_{pk_1}(r_1)$. Run the real-life protocol with $\mathcal{A}, \mathcal{Z}$ on the fake string; if an honest party is supposed to commit to a bit b let this party compute the commitment by using the preselected values: $y = G_{pk_0}(r_0)$ if $b = 0$ and $y = G_{pk_1}(r_1) \oplus \sigma$ for $b = 1$. If the honest party is later asked to decommit, then the opening is done by sending b and the value r_b. At the end of the execution, output whatever $\mathcal{Z}$ returns.

Ideal/Fake: The output of $\mathcal{Z}$ in an execution in the ideal process with $\mathcal{S}$ and $\mathcal{F}_{\text{COM}}$ (on a fake public string chosen by $\mathcal{S}$).

Indistinguishability of Real/Genuine and Real/Fake. Let us presume, for sake of contradiction, that $\mathcal{Z}$ tells apart the hybrids Real/Genuine and Real/Fake with non-negligible probability. From this, we construct an algorithm deciding if an input is truly random or pseudorandom. Details follow.

We are given the security parameter n, a random public key pk of a trapdoor pseudorandom generator $G_{pk} : \{0,1\}^n \rightarrow \{0,1\}^{4n}$ together with a string $z \in \{0,1\}^{4n}$, either chosen at random or produced by applying G_{pk}. We are supposed to predict in which way z has been generated.

To distinguish a random z and a pseudorandom z we use the environment $\mathcal{Z}$ distinguishing Real/Genuine and Real/Fake. For this, we generate a string σ similar to the procedure of $\mathcal{S}$, but we deploy the given string z. Then we basically emulate a real-life execution simulating all honest parties; in particular, we read all the incoming messages from $\mathcal{Z}$. More specifically,

- generation of public string:
  - pick a bit c at random and set $pk_{1-c} = pk$ for the given public key (the bit c is our guess for the bit of an honest party committing)
  - generate another key pair $(pk_c, td_c) \leftarrow \mathsf{KGen}(1^n)$
  - select $r_c \in \{0,1\}^n$ at random and set $\sigma = G_{pk_c}(r_c) \oplus z$
- emulation:
  - simulate the real-life protocol with $\mathcal{A}, \mathcal{Z}$ on σ, pk_0, pk_1

[2]Abusing notation, the same symbols will be typically used to refer to the output of $\mathcal{Z}$ from an experiment and to the experiment itself.

- if an uncorrupted party P_i is told by $\mathcal{Z}$ to commit to a bit b, then we stop immediately with output 0 if $b \neq c$ (i.e., our guess is wrong). In the case $b = c$ we send $G_{pk_c}(r_c)$ for $b = c = 0$ and z for $b = c = 1$ in the name of P_i and continue the simulation; when $\mathcal{Z}$ later instructs P_i to decommit, we transmit $b(= c)$ and r_c. Analogously, we present b, r_c to $\mathcal{A}$ if this party is corrupted before decommitting.
- if the adversary $\mathcal{A}$ corrupts the sender P_i before this party is giving the commitment, then we stop with probability 1/2 (this provides symmetry to the first case and simplifies the analysis); otherwise we go on with the real-life simulation.

• output:

- given that we have not stopped yet, simply copy $\mathcal{Z}$'s output.

To analyze the advantage of our algorithm we start with the case that z is a uniformly distributed $4n$-bit string. Then σ is also random and our prediction c is hidden information-theoretically from $\mathcal{A}$ and $\mathcal{Z}$ at the outset of the execution. Therefore, the probability that we stop prematurely with output 0 is 1/2, independent of the fact whether $\mathcal{A}$ plays the committer or lets an honest party commit. Conditioning on that we enter the final output step, it is easy to see that $\mathcal{Z}$'s output is identically distributed to a sample of Real/Genuine.

Now let z be produced by sampling $G_{pk}(\cdot)$. In this case σ corresponds to a fake string. Also, the public string does not reveal anything to $\mathcal{A}$ and $\mathcal{Z}$ about c. We conclude again that we stop early with probability 1/2, regardless of who commits. Additionally, given that we reach the final step, $\mathcal{Z}$'s output is distributed like a sample from Real/Fake.

Hence, in both experiments Real/Genuine and Real/Fake we output 1 with half the probability that $\mathcal{Z}$ returns 1. It follows that if $\mathcal{Z}$'s advantage separating Real/Genuine and Real/Fake equals

$$\begin{aligned}\epsilon(n) \quad = \quad & |\text{Prob}\,[\mathcal{Z} \text{ outputs 1 in experiment } \mathsf{Real/Genuine}] \\ & \qquad - \text{Prob}\,[\mathcal{Z} \text{ outputs 1 in experiment } \mathsf{Real/Fake}]|\,,\end{aligned}$$

then our advantage distinguishing pseudorandom from random inputs equals $\epsilon(n)/2$. In particular, if $\epsilon(n)$ is non-negligible, so is $\epsilon(n)/2$, and this contradicts the pseudorandomness of the generator.

Indistinguishability of Real/Fake and Ideal/Fake. Obviously, given that $\mathcal{A}$ does not manage to send some y^* in the range of G_{pk_0} and to open this value later correctly with $b^* = 1$, the two experiments are identical. Thus, it suffices to bound the probability for such a mismatch. We show that this probability is negligible because of the pseudorandomness of the generators.

Suppose that the probability in experiment Ideal/Fake that $\mathcal{A}$ commits for a corrupted party to y^* such that y^* and $y^* \oplus \sigma$ are images under G_{pk_0} and G_{pk_1}, respectively, is not negligible. Construct the following algorithm: the input to the algorithm is n, a public key pk and a $4n$-bit string z, and the output is a bit indicating whether z is random or pseudorandom.

1. set $pk_1 = pk$, generate another random key pair (pk_0, td_0) and define $\sigma = G_{pk_0}(r_0) \oplus z$ for random $r_0 \in \{0, 1\}^n$.
2. emulate the Ideal/Fake experiment with $\mathcal{S}, \mathcal{Z}$ on σ, pk_0, pk_1; abort if an honest party is instructed to commit.
3. if $\mathcal{A}$ lets a corrupted party commit to y^*, check —with the help of td_0— if y^* is an image under G_{pk_0}. If this corrupted party then also gives a correct opening of y^* for $b^* = 1$, then stop and output 1.

4. in any other case, return 0.

Observe that this algorithm merely returns 1 if the verification with td_0 yields a preimage r_0^* under G_{pk_0} and if the adversary also reveals r_1^* such that

$$G_{pk_0}(r_0^*) = y^* = G_{pk_1}(r_1^*) \oplus \sigma = G_{pk_1}(r_1^*) \oplus G_{pk_0}(r_0) \oplus z$$

But for random z the probability that

$$z \in \{G_{pk_0}(r_0) \oplus G_{pk_0}(r_0^*) \oplus G_{pk_1}(r_1^*) \mid r_0, r_0^*, r_1^* \in \{0,1\}^n\}$$

is at most 2^{-n}. Thus, in this case, our algorithm outputs 1 with exponentially small probability only. On the other hand, if z is pseudorandom then our algorithm outputs 1 with the same probability as the adversary $\mathcal{A}$ produces a mismatch in the experiment $\mathsf{Ideal/Fake}$. By assumption, this probability is non-negligible. Therefore, the overall advantage of our algorithm is non-negligible, too, refuting the fact that the generator is pseudorandom. This concludes the proof. □

4.2 Reusable Common Reference String: Erasing Parties

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. In the next section we present a solution that does not require data erasure.

4.2.1 Preliminaries

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 $f_{0,pk_{\mathsf{claw}}}^{-1}, f_{1,pk_{\mathsf{claw}}}^{-1}$. 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, based on the decisional Diffie-Hellman assumption, appears in [CS98]. Both schemes have errorless decryption.

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

public string:

pk_{claw} — public key for claw-free trapdoor permutation pair f_0, f_1
$pk_{\mathcal{E}}$ — public key for encryption algorithm 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 $y \stackrel{?}{=} f_b(x)$,
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.
If the verification succeeds then P_j outputs $(\mathtt{Open}, sid, cid, P_i, P_j, b)$.

Figure 5: Commitment Scheme with Reusable Reference String

4.2.2 Description of the Commitment Scheme

The commitment scheme $\mathsf{UCC}_{\mathsf{ReUse}}$ (for universally composable commitment with reusable reference string) is displayed in Figure 5. 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 obtains a value y by applying 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) to the receiver. The sender is also instructed to erase the randomness used 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.

4.2.3 Basic Properties

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 preimages 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 deny to know the randomness for the other ciphertext. If the encryption scheme is secure then it is intractable to distinguish dummy encryptions from fake ones. 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.

4.2.4 Security

We are now ready to prove that protocol $\mathsf{UCC}_{\mathsf{ReUse}}$ securely realizes functionality $\mathcal{F}_{\text{MCOM}}$:

Theorem 8 *Protocol* $\mathsf{UCC}_{\mathsf{ReUse}}$ *securely realizes functionality* $\mathcal{F}_{\text{MCOM}}$ *in the* CRS *model.*

Proof: As in the proof of Theorem 7 we present an ideal-process adversary $\mathcal{S}$ simulating a virtual copy of the real-life adversary $\mathcal{A}$ and relaying messages of $\mathcal{A}$ and the environment $\mathcal{Z}$. The ideal-process adversary is defined by the following actions:

1. the simulator $\mathcal{S}$ chooses keys $(pk_{\mathsf{claw}}, td_{\mathsf{claw}}) \leftarrow \mathsf{KGen}_{\mathsf{claw}}(1^n)$ and $(pk_{\mathcal{E}}, sk_{\mathcal{E}}) \leftarrow \mathsf{KGen}_{\mathcal{E}}(1^n)$, defines the public string to be the pair $pk_{\mathsf{claw}}, pk_{\mathcal{E}}$, and simulates an execution of $\mathcal{A}$ with $\mathcal{Z}$ on $pk_{\mathsf{claw}}, pk_{\mathcal{E}}$.

2. If during this execution the environment $\mathcal{Z}$ lets an *uncorrupted* party $\tilde{P}_i$ send a message $(\texttt{Commit}, sid, cid, P_i, P_j, b)$ to the functionality then the ideal-model simulator is informed about the commitment but not the bit itself. The simulator picks a random $x_0 \in \{0,1\}^n$, computes $y = f_0(x_0)$ and $x_1 = f_1^{-1}(y)$ as well as encryptions $c_0 \leftarrow \mathsf{Enc}(x_0, P_i)$ and $c_1 \leftarrow \mathsf{Enc}(x_1, P_i)$. Tell $\mathcal{A}$ that party P_i has sent $sid, cid, (y, c_0, c_1)$.

3. If an *uncorrupted* party $\tilde{P}_i$ is instructed by $\mathcal{Z}$ to open a commitment to some bit b, then the ideal-model adversary learns b from $\mathcal{F}_{\text{MCOM}}$. Pretend in the simulation of $\mathcal{A}$ that the previously sent (y, c_0, c_1) equals a b-commitment by sending b, x_b and the randomness to encrypt c_b; claim that the randomness for the other encryption has been deleted.

4. If the simulated $\mathcal{A}$ lets some *corrupted* party P_i commit to an honest party P_j by sending $(\texttt{Com}, sid^*, cid^*, (y^*, c_0^*, c_1^*))$, then $\mathcal{S}$ decrypts c_0^* with $sk_{\mathcal{E}}$ to (x^*, P_i^*) and checks whether $P_i^* = P_i$ and if cid^* has not been used in a commitment of P_i to P_j before; if either condition is violated then ignore this message. Else, $\mathcal{S}$ sends a message $(\texttt{Commit}, sid^*, cid^*, P_i, P_j, b)$ on behalf of P_i to the functionality, where the bit b is determined as follows. If c_0^* was previously used in a (simulated) commitment (y, c_0^*, c_1) or (y, c_0, c_0^*) of P_i when P_i was still uncorrupted, then the bit b is set to the bit that this previous commitment was opened to (either by an instruction of $\mathcal{Z}$ or upon corruption of P_i); otherwise, if $f_0(x^*) = y^*$ then $b = 0$, else $b = 1$.

5. If $\mathcal{A}$ tells a *corrupted* party P_i to open a commitment $(\texttt{Com}, sid^*, cid^*, (y^*, c_0^*, c_1^*))$ correctly with bit b^*, then $\mathcal{S}$ compares b^* to the previously extracted bit for these IDs and aborts if the

bits are different; in case of equality $\mathcal{S}$ sends ($\mathtt{Open}$, sid, cid, P_i, P_j) in the name of the party to $\mathcal{F}_{\text{MCOM}}$. If $\mathcal{A}$ lets P_i give an incorrect opening, then $\mathcal{S}$ can ignore this message because the functionality does not open it.

6. Assume that $\mathcal{A}$ demands to corrupt a party in the black-box simulation. Then $\mathcal{S}$ gets all internal information from this party by corrupting it in the ideal model. $\mathcal{S}$ modifies all decommitment information about unopened commitments of this party to match the received data and hands this modified internal information to $\mathcal{A}$.

The proof that the $\mathcal{Z}$'s output in the real-life is indistinguishable from its output in the ideal process is in the line of the proof for Theorem 7. We investigate again three hybrid variables:

Real/Genuine: The output of $\mathcal{Z}$ of an interaction in the real-life model with adversary $\mathcal{A}$ and parties running the protocol.

Real/Fake: The output of $\mathcal{Z}$ from the following hybrid interaction in the real-life model with adversary $\mathcal{A}$. The interaction is identical to Real/Genuine, except that honest parties use the following way to commit to a bit b: instead of sending correct values (y, c_0, c_1) the honest player now sends $y = f_0(x_0)$ for random x_0, $c_b \leftarrow \mathsf{Enc}(x_0, P_i)$ and $c_{1-b} \leftarrow \mathsf{Enc}(x_1, P_i)$ where $x_1 = f_1^{-1}(y)$. (The randomness used for generating c_{1-b} is erased.) The opening for this commitment consists of b, x_b and the randomness used to encrypt c_b.

Ideal/Fake: The output of $\mathcal{Z}$ in an execution in the ideal process with $\mathcal{S}$ and $\mathcal{F}_{\text{MCOM}}$.

Suppose that the extreme hybrids Real/Genuine and Ideal/Fake are distinguishable. This means either that the hybrids Real/Genuine and Real/Fake iare distinguishable or that the hybrids Real/Fake and Ideal/Fake are distinguishable. We will show that this leads to a contradiction to the claw-freeness or to the chosen ciphertext security of the encryption scheme.

Real/Genuine and Real/Fake are indistinguishable. Assume that the variables Real/Genuine and Real/Fake are distinguishable. The only difference between the two executions is that honest parties in Real/Fake send encryptions of claws instead of encryptions of 0^n. But since the encryption scheme is secure this difference should be negligible. We prove this rigorously.

We remark that our analysis uses an alternative (but equivalent) formalization of IND-CCA2 security. This formalization has been introduced by Bellare et al. [BDJR97] in the private-key setting under the name left-or-right security against chosen ciphertext attacks, and has been shown to be equivalent to IND-CCA2 in the public-key model in [BBM00]. Basically, security is defined as follows: the adversary gets a public key $pk_{\mathcal{E}}$ and is allowed to query adaptively a so-called left-or-right encryption oracle for pairs of messages (m_0, m_1). This left-or-right oracle answers with an encryption of m_{CB} under $\mathsf{Enc}_{pk_{\mathcal{E}}}(\cdot)$, where the secret challenge bit CB is randomly chosen at the beginning but is fixed throughout the whole attack. The adversary is also given access to the decryption oracle $\mathsf{Dec}_{sk_{\mathcal{E}}}(\cdot)$; as usual, the adversary is not allowed to query the decryption oracle for ciphertexts obtained from the left-or-right encryption oracle. Finally, the adversary is supposed to output a guess for CB. For such an LR-CCA2 scheme the prediction probability of any polynomially-bounded adversary should not exceed 1/2 by a non-negligible amount.

Given environment $\mathcal{Z}$ that distinguishes between Real/Genuine and Real/Fake, we construct a successful distinguisher for the LR-CCA2 property of the encryption scheme $\mathcal{E}$; in fact, this distinguisher never queries the decryption oracle, so left-or-right security against chosen plaintext attacks (CPA) [BBM00] suffices in this step.

Distinguisher $\mathcal{D}_{CPA}$ gets 1^n and a random public key $pk_{\mathcal{E}}$ obtained by running $\mathsf{KGen}_{\mathcal{E}}(1^n)$ as input. Let CB be the random bit that determines if the left-or-right encryption oracle returns ciphertexts of the left (CB $= 0$) or the right (CB $= 1$) messages. $\mathcal{D}_{CPA}$ tries to predict CB by simulating a real-life execution:

1. $\mathcal{D}_{CPA}$ picks $(pk_{\mathsf{claw}}, td_{\mathsf{claw}}) \leftarrow \mathsf{KGen}_{\mathsf{claw}}(1^n)$
2. $\mathcal{D}_{CPA}$ imitates a real-life execution of $\mathcal{A}$ with $\mathcal{Z}$ on $pk_{\mathsf{claw}}, pk_{\mathcal{E}}$. In particular, $\mathcal{D}_{CPA}$ plays all honest parties and reads all the messages sent from $\mathcal{Z}$ to the other parties.
3. if an honest party P_i is told to commit to a bit b then $\mathcal{D}_{CPA}$ —who knows b— selects $x_b \in \{0,1\}^n$ at random, and computes $y = f_b(x_b)$, $x_{1-b} = f_{1-b}^{-1}(y)$ as well as $c_b \leftarrow \mathsf{Enc}(x_b, P_i)$. Then, $\mathcal{D}_{CPA}$ gives the pair $(0^n, P_i)$, (x_{1-b}, P_i) (in this order) to the left-or-right encryption oracle. Denote the answer by c_{1-b}. Send (y, c_0, c_1) on behalf of the honest party.
4. if the honest party is asked to decommit (or, similarly, is corrupted before decommitting) then $\mathcal{D}_{CPA}$ presents b, x_b and the randomness for producing c_b.
5. at the end, copy $\mathcal{Z}$'s output

If the left-or-right oracle always encrypts the left message $(0^n, P_i)$ then $\mathcal{D}_{CPA}$ simulates a real-life execution with correctly behaving honest parties. We conclude that the probability that $\mathcal{D}_{CPA}$ outputs 1 in this case equals the probability that $\mathcal{Z}$ returns 1 in experiment Real/Genuine. Also, if the oracle has encrypted all the right messages (x_{1-b}, P_i) then $\mathcal{D}_{CPA}$ simulates the experiment Real/Fake and outputs 1 exactly if $\mathcal{Z}$ gives output 1 in this experiment. Hence,

$$\begin{aligned}
&\mathrm{Prob}[\mathcal{D}_{CPA} \text{ outputs CB}] \\
&\quad= \mathrm{Prob}[\mathrm{CB} = 1 \wedge \mathcal{Z} \text{ outputs } 1] + \mathrm{Prob}[\mathrm{CB} = 0 \wedge \mathcal{Z} \text{ outputs } 0] \\
&\quad= \tfrac{1}{2} \cdot \mathrm{Prob}[\mathcal{Z} \text{ outputs 1 in experiment } \mathsf{Real/Fake}] \\
&\qquad\quad + \tfrac{1}{2} \cdot \mathrm{Prob}[\mathcal{Z} \text{ outputs 0 in experiment } \mathsf{Real/Genuine}] \\
&\quad= \tfrac{1}{2} \cdot \mathrm{Prob}[\mathcal{Z} \text{ outputs 1 in experiment } \mathsf{Real/Fake}] \\
&\qquad\quad + \tfrac{1}{2} \cdot (1 - \mathrm{Prob}[\mathcal{Z} \text{ outputs 1 in experiment } \mathsf{Real/Genuine}]) \\
&\quad= \tfrac{1}{2} + \tfrac{1}{2} \cdot (\mathrm{Prob}[\mathcal{Z} \text{ outputs 1 in experiment } \mathsf{Real/Fake}] \\
&\qquad\qquad\qquad - \mathrm{Prob}[\mathcal{Z} \text{ outputs 1 in experiment } \mathsf{Real/Genuine}])
\end{aligned}$$

$\mathcal{D}_{CPA}$'s prediction probability is therefore bounded away from 1/2 by a non-negligible function, contradicting the left-or-right property of the encryption scheme $\mathcal{E}$.

Real/Fake and Ideal/Fake are indistinguishable. The only point in which the two experiments could diverge is if during the simulation $\mathcal{S}$ hands $\mathcal{F}_{\text{MCOM}}$ a value b in the name of some corrupted party, and later this corrupted party manages to successfully decommit to $b^* \neq b$. More precisely, define the following bad event $\mathcal{B}$: Event $\mathcal{B}$ occurs if during the run of $\mathcal{S}$ the following happens: *(a)* The simulated $\mathcal{A}$ generates a commitment $(\mathtt{Com}, sid, cid, (y, c_0, c_1))$ in the name of some corrupted party P_i, *(b)* $\mathcal{S}$ hands $\mathcal{F}_{\text{MCOM}}$ a value $(\mathtt{Commit}, sid, cid, b)$, and *(c)* The simulated $\mathcal{A}$ later generates a valid opening of $(\mathtt{Com}, sid, cid, (y, c_0, c_1))$ to a value $b^* \neq b$. Then, as long as event $\mathcal{B}$ does not occur the view of $\mathcal{Z}$ in experiment Real/Fake is identical to its view in Ideal/Fake. So it remains to demonstrate that event $\mathcal{B}$ occurs with negligible probability.

We would like to demonstrate that last statement via reduction to the security of the claw-free pair (f_0, f_1). However, a direct reduction does not seem to work. We thus first show that if event $\mathcal{B}$ occurs in $\mathsf{Ideal/Fake}$ with non-negligible probability, then this should also be true if we replace the simulated commitments of honest parties in $\mathcal{A}$'s simulation with commitments where we correctly put a dummy ciphertext into the tuple instead of an encryption of (x_{1-b}, P_i). Call this new experiment $\mathsf{Ideal/Genuine}$. That is, experiment $\mathsf{Ideal/Genuine}$ is identical to experiment $\mathsf{Ideal/Fake}$ with the exception that in $\mathsf{Ideal/Genuine}$ the simulator $\mathcal{S}$ 'magically knows' the real values committed to by the uncorrupted parties, and generates genuine commitments for these values. We show that if the probability of event $\mathcal{B}$ in the two experiments differs by non-negligible amount then it is possible to break the CCA security of the encryption scheme $\mathcal{E}$.

Claim 9 *The probability of event $\mathcal{B}$ in experiment* $\mathsf{Ideal/Fake}$ *differs from the probability of event $\mathcal{B}$ in experiment* $\mathsf{Ideal/Genuine}$ *by at most a negligible amount.*

Proof: We first observe that in order for event $\mathcal{B}$ to happen, the simulated adversary $\mathcal{A}$ must generate a message $(\mathtt{Com}, sid, cid, (y, c_0, c_1))$ such that c_0 decrypts to (x_0, P_i), c_1 decrypts to (x_1, P_i), $f_0(x_0) = f_1(x_1) = y$, and cid was never used before for a commitment of P_i to P_j. If this event occurs then we say that $\mathcal{A}$ has $\mathsf{found\ a\ claw}$.

Assume towards contradiction that there exist an environment $\mathcal{Z}$ and adversary $\mathcal{A}$ such that the probabilities that $\mathcal{A}$ finds a claw in the two interactions differ by a non-negligible amount. From this we devise a distinguisher $\mathcal{D}_{CCA}$ for $\mathcal{E}$ that works similarly to the distinguisher $\mathcal{D}_{CPA}$ above, but runs an adaptive chosen ciphertext attack against the left-or-right security. $\mathcal{D}_{CCA}$ gets 1^n and a random public key $pk_{\mathcal{E}}$ obtained by running $\mathsf{KGen}_{\mathcal{E}}(1^n)$ as input, together with oracle access to a left-or-right encryption oracle initialized with random bit CB, and to the decryption oracle $\mathsf{Dec}(\cdot)$.

1. $\mathcal{D}_{CCA}$ generates $(pk_{\mathsf{claw}}, td_{\mathsf{claw}}) \leftarrow \mathsf{KGen}_{\mathsf{claw}}(1^n)$
2. $\mathcal{D}_{CCA}$ follows the pattern of a ideal-model execution of $\mathcal{S}$ with $\mathcal{Z}$ on keys $pk_{\mathsf{claw}}, pk_{\mathcal{E}}$; $\mathcal{D}_{CCA}$ also executes a black-box simulation of $\mathcal{A}$. In contrast to $\mathcal{S}$, who cannot read $\mathcal{Z}$'s messages to honest parties, $\mathcal{D}_{CCA}$ gets to know all messages.
3. Whenever an uncorrupted party P_i commits to a value b, $\mathcal{D}_{CCA}$ does the following: first select a random $x_b \in \{0,1\}^n$ and compute $y = f_b(x_b)$, $x_{1-b} = f_{1-b}^{-1}(y)$ and $c_b \leftarrow \mathsf{Enc}(x_b, P_i)$. Next the distinguisher queries the left-or-right encryption oracle about (x_{1-b}, P_i) and $(0^n, P_i)$ in this order and stores the answer in c_{1-b}. Finally, $\mathcal{D}_{CCA}$ sends (y, c_0, c_1) in the name of the honest party.
4. If an uncorrupted party P_i is asked to decommit (or corrupted before opening) then $\mathcal{D}_{CCA}$ presents the corresponding values of b, x_b and the randomness for c_b.
5. If the simulated $\mathcal{A}$ lets some *corrupted* party P_i commit to an honest party P_j by sending $(\mathtt{Com}, sid^*, cid^*, (y^*, c_0^*, c_1^*))$, then $\mathcal{D}_{CCA}$ proceeds as follows:
   (a) If c_0^* has not been returned from the left-or-right encryption oracle of $\mathcal{D}_{CCA}$ before, then $\mathcal{D}_{CCA}$ asks its decryption oracle to decrypt c_0^* and proceeds like the ideal-model adversary $\mathcal{S}$.
   (b) Otherwise, if c_0^* has been returned from the left-or-right encryption oracle, then $\mathcal{D}_{CCA}$ has sent this value in a commitment (y, c_0^*, c_1) or (y, c_0, c_0^*) in the name of some honest party. If this has been a different party than P_i then ignore the adversary's message. Else

(c_0^* appeared in a commitment of P_i before P_i was corrupted), recall the corresponding bit from the previous commitment and proceed like the ideal-model adversary $\mathcal{S}$.

6. If $\mathcal{A}$ tells a corrupted party to open a commitment $(\texttt{Com}, sid^*, cid^*, (y^*, c_0^*, c_1^*))$ correctly with bit b^*, then $\mathcal{D}_{CCA}$ compares b^* to the previously extracted bit for sid^*, cid^* and halts with output 1 if they are distinct; Otherwise $\mathcal{D}_{CCA}$ proceeds as the ideal-model adversary.

7. If $\mathcal{A}$ halts without finding a claw then output 0 and halt.

The analysis of $\mathcal{D}_{CCA}$ is almost identical to the case of distinguisher $\mathcal{D}_{CPA}$ above and is omitted. This completes the proof of Claim 9. □

It remains to prove that $\mathcal{A}$ finds claws in experiment Ideal/Genuine with negligible probability only. But this follows from the claw-freeness of the trapdoor permutation pair. To be more precise, given an environment $\mathcal{Z}$ and adversary $\mathcal{A}$ that find claws in Ideal/Genuine, we construct an algorithm that finds claws in the claw-free pair: Given 1^n and a random pk_{claw}, generate $(pk_{\mathcal{E}}, sk_{\mathcal{E}})$; simulate the experiment Ideal/Genuine by reading $\mathcal{Z}$'s commitment instructions to honest parties and giving a correct commitment, involving a dummy encryption. For $\mathcal{A}$ committing in the black-box simulation extract the bit using the secret key $sk_{\mathcal{E}}$. If at some step $\mathcal{A}$ generates a claw by outputting a preimage x_{b^*} under f_{b^*} for some y^* for which we have extracted a preimage x_{1-b^*} under f_{1-b^*} before, then we output this pair and stop. If this event would occur with non-negligible probability it would render the claw-freeness wrong. □

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 (aka. trapdoor) commitments a la [BCC88]. That is, any non-interactive perfectly-secret trapdoor commitment works. Such commitments exist for instance under the hardness of the discrete logarithm or factoring problem. Further relaxing the underlying hardness assumptions is an interesting open problem..

4.3 Reusable Common Reference String: Non-Erasing Parties

A careful look at the proof of Theorem 8 shows that, instead of letting the sender generate a ciphertext and erase the randomness, it is sufficient to enable the parties to obliviously generate a "ciphertext-like" string without knowing the plaintext, but such that a simulator can produce a correct ciphertext and a fake random string suggesting that the ciphertext has been obtained by the oblivious sampling procedure. Then the honest parties can use the sampling mechanism to produce the dummy ciphertext, while the simulator is still able to place the fake encryption into the commitment and to find fake randomness making it look like a dummy ciphertext. We show how this can be done under certain conditions, and show that these conditions can be met if the encryption scheme in use is that of [CS98].

4.3.1 Preliminaries: Obliviously Samplable Encryption Scheme

We formalize the requirement for the oblivious sampling procedure of the encryption scheme in the following definition:

Definition 10 *A public-key encryption scheme $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is* obliviously samplable *with respect to chosen-plaintext attacks if there are probabilistic polynomial-time algorithms* sample, fake

such that for any probabilistic polynomial-time algorithm $\mathcal{A}$ the probability that Experiment$_{\mathcal{A}}(1^n) = 1$ is negligibly close to $1/2$, where
Experiment$_{\mathcal{A}}(1^n)$:

- *a secret random bit* $\mathrm{CB} \in \{0,1\}$ *is chosen*
- *generate a key pair* $(pk, sk) \leftarrow \mathsf{KGen}(1^n)$
- *invoke $\mathcal{A}$ on pk to obtain a message m*
- *generate the challenge:*
  - *if* $\mathrm{CB} = 0$ *then sample a pseudo-ciphertext* $c_{\mathsf{sample}} \leftarrow \mathsf{sample}(pk, 0^{|m|})$ *(with randomness r_{sample}) and return $(c_{\mathsf{sample}}, r_{\mathsf{sample}})$ to $\mathcal{A}$. (Note: this case corresponds to the adversary's view when the committer is honest.)*
  - *if* $\mathrm{CB} = 1$ *then encrypt m to $c \leftarrow \mathsf{Enc}(pk, m)$, calculate $r_{\mathsf{fake}} \leftarrow \mathsf{fake}(pk, c)$ and hand (c, r_{fake}) to $\mathcal{A}$ (Note: this case corresponds to the adversary's view when the committer is played by the simulator.)*
- *output 1 if and only if $\mathcal{A}$'s output equals* CB

If the probability for Experiment$_{\mathcal{A}}(1^n) = 1$ remains negligibly close to $1/2$ even if $\mathcal{A}$ is additionally allowed to query the decryption oracle $\mathsf{Dec}(sk, \cdot)$ *during the attack for any values different than the challenge, then the scheme is called obliviously samplable with respect to chosen-ciphertext attacks.*

In particular, it should hold that sample maps to c under randomness r_{fake} for fake's output (c, r_{fake}) with overwhelming probability, i.e., the fake output should look like an oblivious sample. Note that the sample algorithm gets the length of m as additional input, since the length of a message can be deduced from the ciphertext. Also note that an obliviously samplable encryption scheme is semantically secure against the corresponding type of attack.

4.3.2 Example of Obliviously Samplable Encryption Scheme

In the Cramer-Shoup encryption scheme [CS98] the public key consists of a group G of prime order q, two generators g_1, g_2 of G and three group elements c, d, h as well as a universal one-way hash function H. To encrypt a message $m \in G$ compute

$$u_1 = g_1^r,\ u_2 = g_2^r,\ e = h^r m,\ \alpha = H(u_1, u_2, e),\ v = c^r d^{r\alpha}$$

and output the ciphertext (u_1, u_2, e, v).

Let us assume that $p = qw + 1$ for some w not divisible by q, and that G is a subgroup of order q in $\mathcal{Z}_p^*$ (and that w is public). Then in order to obliviously sample a random group element in G we first generate a random element in $\mathcal{Z}_p^*$ by picking a random bit string of length $2|p|$ and interpreting it as a number between 1 and $p-1$ by reduction modulo p of the bit string viewed as an integer. Then we raise this element to the w-th power and return it. We remark that this element is statistically close to a uniformly chosen one from G. We call this sampling process the oblivious element generation for G.

The oblivious element generation for G is invertible in the sense that, given a random group element $h \in G$ we can efficiently generate a random element h_p in $\mathcal{Z}_p^*$ (and a corresponding bit string of length $2|p|$) mapping to h if raised to the w-th power. Namely, let g be a generator of

$\mathcal{Z}_p^*$. Solve the equation $xw = 1 \bmod q$ for x, pick a random integer i between 0 and $w-1$ and define the element $h_p := h^x g^{iq} \bmod p$. Since the g^{iq}'s are w-th roots of unity, it is readily verified that indeed $h_p^w = h \bmod p$ and that h_p is uniformly distributed among the preimages of h under exponentiation with w. Adding for random j between 0 and $p-1$ the value jp to h_p over the integers gives a $2|p|$-bit string whose distribution is statistically close to the uniform distribution on bit strings mapping to h with the oblivious element generation.

We describe our algorithms sample and fake. Algorithm sample on input $pk, 0^{|m|}$ simply generates four random group elements u_1, u_2, e, v with independent executions of the element generation procedure for G and returns them, together with all the randomness for these executions. Algorithm fake, on the other side, given pk and a correct ciphertext $c = (u_1, u_2, e, v)$, runs the inverse process to the element generation for G as described above for each element and returns the derived bit strings.

The fact that the outputs of sample and fake are indistinguishable under the Decisional Diffie-Hellman assumption follows from the proof in [CS98]. This is true even if the adversary has access to the decryption oracle. Altogether, the Cramer-Shoup scheme is obliviously samplable with respect to chosen-ciphertext attacks.

4.3.3 Description and Security of Commitment Scheme

Besides being obliviously samplable with respect to adaptive chosen-ciphertext attacks, we again presume that the encryption scheme $\mathcal{E}$ is uniquely decipherable. Modify the scheme $\mathsf{UCC}_{\mathsf{ReUse}}$ insofar as the sender does not compute the dummy ciphertext $c_{1-b} \leftarrow \mathsf{Enc}(0^n, P_i)$ and then erases the randomness, but rather samples $c_{1-b} \leftarrow \mathsf{sample}(pk_{\mathcal{E}}, 0^\ell)$ (where ℓ denotes the length of $(0^n, P_i)$) with randomness r_{sample} obliviously. In the decommitment step or if corrupted, the sender reveals r_{sample} for this part of the commitment. Call this scheme $\mathsf{UCC}_{\mathsf{ReUse/NotErase}}$.

Theorem 11 *Protocol* $\mathsf{UCC}_{\mathsf{ReUse/NotErase}}$ *securely realizes functionality* $\mathcal{F}_{\text{MCOM}}$ *in the* CRS *model.*

Proof: The proof of the theorem is almost identical to the one in the case of erasing parties. Only this time the ideal-model simulator works slightly different when an uncorrupted party commits or is corrupted or decommits. Namely, for a commitment the simulator in Theorem 8 sends encryptions of (x_0, P_i) and (x_1, P_i) in the name of this party; after having learned the actual bit b in case of corruption or decommitment, the simulator there then claims to have erased the randomness for the wrong value x_{1-b}. In our case, the simulator also encrypts both values in the commitment phase, but in the reveal step it invokes algorithm fake on public key $pk_{\mathcal{E}}$ and the ciphertext for the wrong value x_{1-b} to produce a fake random string. Besides the true randomness used to produce the encryption of x_b, the simulator hands the fake randomness to the adversary in order to prove that the ciphertext for x_{1-b} has been sampled obliviously.

In the proof of Theorem 8, the indistinguishability of the simulator's way to commit and decommit on behalf of honest parties and the behavior of the actual sender relies on the indistinguishability of fake and dummy encryptions. Specifically, we have reduced indistinguishability of simulations twice to the left-or-right security of the encryption system, one time in a chosen-plaintext attack and the other time in a chosen-ciphertext attack. In these reductions the left-or-right oracle encrypts either all left messages $(0^n, P_i)$ or all right messages (x_{1-b}, P_i). Which messages are encrypted, the left or right ones, corresponds to the behavior of honest parties or the simulator.

Except for the reductions to left-or-right security the proof of Theorem 8 remains unchanged. In particular, the behavior of $\mathcal{S}$ in case that the committer is corrupted remains unchanged. The simulation remains valid since the encryption scheme remains uniquely decipherable.

To adapt the proof to the simulation here it is sufficient to extend the notion of left-or-right security to obliviously samplable encryption schemes. Namely, the so-called sample-or-encrypt oracle is initialized with a challenge bit CB and the adversary is given the public key pk and is allowed to hand messages m to the oracle and either receives a sample $(c_{\mathsf{sample}}, r_{\mathsf{sample}})$ if $\mathrm{CB} = 0$ or a ciphertext c of m with fake randomness r_{fake} if $\mathrm{CB} = 1$. The adversary is supposed to predict CB with non-negligible advantage. If for any efficient adversary mounting a chosen-plaintext attack the advantage predicting CB is negligible, then the scheme is called sample-or-encrypt secure against chosen-plaintext attacks. If the adversary is also allowed to submit queries to the decryption oracle —all different from the answers of the sample-or-encrypt oracle— then the scheme is said to be sample-or-encrypt secure against chosen-ciphertext attacks.

In analogy to the proof in [BBM00] it follows that the encryption scheme is sample-or-encrypt secure against chosen-plaintext attacks if the encryption system if obliviously samplable with respect to chosen-plaintext attacks. Additionally, if the encryption scheme is obliviously samplable with respect to chosen-ciphertext attacks, then the system is sample-or-encrypt secure against such attacks.

Here, instead of passing $(0^n, P_i)$ and (x_{1-b}, P_i) to the left-or-right oracle, we forward the message (x_{1-b}, P_i) to the sample-or-encrypt oracle to obtain either an oblivious sample c_{sample} and the randomness r_{sample}, or a ciphertext of the message (x_{1-b}, P_i) together with a fake random string. Denote the answer by (c_{1-b}, r_{1-b}) and let the simulator transmit c_{1-b} as part of the commitment. Later, in the decommitment phase or upon corruption, the simulator reveals r_{1-b} on behalf of the sender. As the choice of the sample-or-encrypt oracle determines whether we simulate honest parties (if the oracle returns oblivious samples) or the simulator (if the oracle produces correct ciphertexts and fake randomness), it is easy to see that the proof of Theorem 8 carries over to this case. $\square$

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, $\mathcal{F}_{\mathrm{ZK}}$, that implies the notion of Zero-Knowledge in a very strong sense. (In fact, $\mathcal{F}_{\mathrm{ZK}}$ implies concurrent and non-malleable Zero-Knowledge proofs of knowledge.) We then show that in the $\mathcal{F}_{\mathrm{COM}}$-hybrid model (i.e., in a model with ideal access to $\mathcal{F}_{\mathrm{COM}}$) there is a 3-round protocol that securely realizes $\mathcal{F}_{\mathrm{ZK}}$ with respect to any NP relation. Using the composition theorem of [C01], we can replace $\mathcal{F}_{\mathrm{COM}}$ with any UC commitment protocol. (This of course requires using the CRS model, unless we involve third parties in the interaction. Also, using functionality $\mathcal{F}_{\mathrm{MCOM}}$ instead of $\mathcal{F}_{\mathrm{COM}}$ is possible and results in a more efficient use of the common string.)

Functionality $\mathcal{F}_{\mathrm{ZK}}$, described in Figure 6, is parameterized by a binary relation $R(x, w)$.. It first waits to receive a message $(\mathtt{verifier}, id, P_i, P_j, x)$ 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, $\mathcal{F}_{\mathrm{ZK}}$ 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.)

We demonstrate a protocol for securely realizing $\mathcal{F}_{\mathrm{ZK}}^{R}$ with respect to 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 $\mathcal{F}_{\mathrm{COM}}$. The

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

$\mathcal{F}_{\text{ZK}}$ 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 $(\texttt{verifier}\ldots)$ 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.

Figure 6: The Zero-Knowledge functionality, $\mathcal{F}_{\text{ZK}}$

protocol (in the $\mathcal{F}_{\text{COM}}$-hybrid model) is presented in Figure 7.

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

Let $\mathcal{F}_{\text{ZK}}^H$ denote functionality $\mathcal{F}_{\text{ZK}}$ parameterized by the Hamiltonicity relation H. (I.e., $H(G, h) = 1$ iff h is a Hamiltonian cycle in graph G.)

Theorem 12 *Protocol* HC *securely realizes $\mathcal{F}_{\text{ZK}}^H$ in the $\mathcal{F}_{\text{COM}}$-hybrid model.*

Proof (sketch): Let $\mathcal{A}$ be an adversary that operates against protocol HC in the $\mathcal{F}_{\text{COM}}$-hybrid model. We construct an ideal-process adversary (i.e., a simulator) $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and HC in the $\mathcal{F}_{\text{COM}}$-hybrid model or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{ZK}}^H$.

Simulator $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$. Messages received from $\mathcal{Z}$ are forwarded to the simulated $\mathcal{A}$, and messages sent by the simulated $\mathcal{A}$ to its environment are forwarded to $\mathcal{Z}$. In addition:

1. If $\mathcal{A}$, controlling a corrupted party P, starts an interaction as a prover with an uncorrupted party V, then $\mathcal{S}$ records the values that $\mathcal{A}$ sends to $\mathcal{F}_{\text{COM}}$, plays the role of V (i.e., $\mathcal{S}$ provides $\mathcal{A}$ with a random set of bits $c_1, ..., c_n$), and records $\mathcal{A}$'s responses. Now $\mathcal{S}$ simulates V's decision algorithm and if V accepts then $\mathcal{S}$ finds a Hamiltonian cycle h in G and hands g to $\mathcal{F}_{\text{ZK}}^H$. Else $\mathcal{S}$ hands an invalid cycle h' in G (say, the all-zero cycle) to $\mathcal{F}_{\text{ZK}}^H$. It remains to describe how $\mathcal{S}$ finds a Hamiltonian cycle h in G. This is done as follows: $\mathcal{S}$ looks for a k such that $c_k = 1$ and the cycle h decommitted to by $\mathcal{A}$, combined with the committed permutation π_k, point to a Hamiltonian cycle in G. If such a k is not found then $\mathcal{S}$ aborts; but, as claimed below, this will occur only with probability $2^{-n/2}$.
2. If an uncorrupted party P starts an interaction with a corrupted party V then $\mathcal{S}$ learns from $\mathcal{F}_{\text{ZK}}^H$ whether V should accept or reject, and simulates the view of $\mathcal{A}$ accordingly. Notice that $\mathcal{S}$ has no problem carrying out the simulation since it simulates for $\mathcal{A}$ an interaction with $\mathcal{F}_{\text{COM}}$ where $\mathcal{F}_{\text{COM}}$ is played by $\mathcal{S}$ himself. Thus, $\mathcal{S}$ is not bound by the "commitments" and can "open" them in whichever way it pleases.

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 $\mathcal{F}_{\text{COM}}$, 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 $\mathcal{F}_{\text{COM}}$, where $e = 1$ if there is an edge between $\pi_k(i)$ and $\pi_k(j)$ in G, and $e = 0$ otherwise. (Here the value (i,j,k) serves as the session ID for the commitment.)
   (c) Using $\mathcal{F}_{\text{COM}}$, commit to the permutation π_k. That is, for $l = 1, ..., L$ send `(Commit,` $(l,k), P, V, p_l$`)` to $\mathcal{F}_{\text{COM}}$ 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 $\mathcal{F}_{\text{COM}}$, 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 $\mathcal{F}_{\text{COM}}$ for all $i,j = 1, ..., n$ and $l = 1, ..., L$.
   (b) If $c_k = 1$ then send `(Open,` $(i,j,k), P, V$`)` to $\mathcal{F}_{\text{COM}}$ 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 $\mathcal{F}_{\text{COM}}$, 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.

Figure 7: The protocol for proving Hamiltonicity in the $\mathcal{F}_{\text{COM}}$-hybrid model

3. If two uncorrupted parties P and V interact then $\mathcal{S}$ simulates for $\mathcal{A}$ the appropriate protocol messages. This case is very similar to the case of corrupted verifier, since this is an Arthur-Merlin protocol.

4. Party corruptions are dealt with in a straightforward way. Corrupting the verifier provides the adversary with no extra information (again, since the protocol is Arthur-Merlin). When the prover is corrupted $\mathcal{S}$ corrupts the prover in the ideal process, obtains w, and generates an internal state of the prover that matches the protocol stage and whether $R(x,w)$ holds. Generating such a state is not problematic since $\mathcal{S}$ is not bound by any "commitments", and it can freely choose $\pi_1, ...\pi_k$ to match the (simulated) conversation up to the point of corruption.

Given that $\mathcal{S}$ does not abort in Step 1, the validity of the simulation is straightforward. We show that $\mathcal{S}$ aborts with probability at most $2^{-n/2}$. Say that index $k \in [n]$ is **valid** if applying the kth committed permutation to the input graph G results in the kth committed graph. If less than $n/2$ of the indices are valid then V accepts with probability at most $2^{-n/2}$. However, if at least $n/2$ of the indices are valid then with probability at least $1 - 2^{-n/2}$ V has $c_k = 1$ for at least one valid index k. In this case, $\mathcal{S}$ will not fail since V accepts only if the decommitted cycle h, together with the permutation π_k, points to a Hamiltonian cycle in G. □

Remark: Notice that Theorem 12 holds even if the environment and the real-life adversary are allowed to be *computationally unbounded.* In this case, the complexity of $\mathcal{S}$ is polynomial in the complexity of $\mathcal{A}$ (and is independent of the complexity of $\mathcal{Z}$). This means that the only place where cryptographic assumptions are needed is in realizing $\mathcal{F}_{\text{COM}}$.

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].) We also thank Roger Fischlin for help with the oblivious element generation in case of non-erasing parties.

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.

[B96] 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.

[B82] M. Blum, "Coin flipping by telephone", IEEE Spring COMPCOM, pp. 133-137, Feb. 1982.

[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.

[C01] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols", in *42nd FOCS,* 2001. Also available at http://eprint.iacr.org/2000/067. (Previous versions of this work appeared under the title "A unified framework for analyzing security of Protocols.")

[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, "Parallel Reducibility for Information-Theoretically 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.