Universally Composable Commitments Using Random Oracles

Dennis Hofheinz[†] and Jörn Müller-Quade[†]

Abstract. In the setting of universal composability [Can01], commitments cannot be implemented without additional assumptions such as that of a publicly available *common reference string* [CF01]. Here, as an alternative to the commitments in the common reference string model, the use of *random oracles* to achieve universal composability of commitment protocols is motivated. Special emphasis is put on the security in the situation when the additional "helper functionality" is replaced by a realizable primitive. This contribution gives two constructions which allow to turn a given non-interactive commitment scheme into a non-interactive universally composable commitment scheme in the random oracle model. For both constructions the binding and the hiding property remain valid when *collision-free hash functions* are used instead of random oracles. Moreover the second construction in this case even preserves the property of perfect binding.
Keywords: cryptographic protocols, universal composition, commitment, random oracle.

1 Introduction

The framework [Can01] for multi-party computations allows to formulate the security and, in particular, the composition of multi-party protocols in a very general way. It is possible to treat security notions for rather different multi-party tasks in a common way. For this, protocols are compared to idealized versions of the respective protocol task. If a protocol "behaves" exactly like this idealization with respect to any attacker and in any environment, it is considered a secure realization of the protocol task in question. In the setting of [Can01] an arbitrary environment surrounding the protocol execution is mimicked by an *environment machine* $\mathcal{Z}$. Furthermore the environment machine $\mathcal{Z}$ serves as a distinguisher between a real protocol and the idealized version. A protocol is securely realizing an ideal functionality if no environment $\mathcal{Z}$ can distinguish between an execution of the real protocol with a real adversary and a run of the ideal functionality together with a simulator trying to mimic the effect of the real attack. For the purpose of distinguishing the environment machine may choose the inputs for all parties, may see the outputs of all parties, and may interact with the adversary at any time during the protocol.

[†] IAKS, Arbeitsgruppe Systemsicherheit, Prof. Dr. Th. Beth, Fakultät für Informatik, Universität Karlsruhe, Germany. E-Mail: {hofheinz,muellerq}@ira.uka.de.

This notion of security, which implies *universal composability* [Can01], is very strict and it was shown in [CF01], that an idealization of a *commitment* task cannot be securely realized in this sense *without additional assumptions*. (With additional assumptions, we mean special facilities protocol participants may use and which itself may not be securely realizable; as an example, consider public information ideally chosen from some predefined distribution. See below for details.)

However, in [CF01,DN02,CLOS02], several protocols for securely realizing an idealization of a commitment functionality are presented; all of them are formulated in the *common reference string model*, i. e., all of them expect access to public information ideally drawn from some predefined distribution.

The selection of this common reference string is crucial for the security of the commitment protocol. In particular, "imperfect" selections possibly influenced by an adversary may affect the security of the commitment protocol in a very severe way, as will be discussed in Section 3.1. The common reference string in [CF01] serves as a public key to which the corresponding secret key is unknown by assumption. If, in the worst case, the adversary were allowed to choose the common reference string by himself then the binding property as well as the hiding property of a commitment scheme built on this common reference string would be compromised (an analogous statement holds for the constructions in [DN02,CLOS02]). This is especially dangerous as this security leak cannot be "detected", because the public key is chosen with the appropriate distribution. As a different approach, we consider the use of *random oracles* for building bit commitment protocols. Of course, like the common reference string, random oracles are not realizable and the property of universal composability is lost when concrete functions replace the random oracle calls. This is in accordance with other results like [CGH98,Nie02,GTK03,BBP03]. These contributions show explicitly that there are protocols which can be proven secure in the random oracle model, yet lose this security completely when instantiating the random oracles. In contrast to that, we show that there is a construction which turns a given bit commitment protocol into a protocol which is universally composable in the random oracle model and which *remains* binding and hiding when substituting the random oracles with a special class of functions (namely, collision-free hash functions).

As a first solution one might think of using random oracles to derive a common reference string with which universally composable bit commitment can be obtained. But if a random oracle would be replaced by any real hash function no general guarantee for the derived common reference string could be given and all protocols on its basis would be critical. To ensure the common reference string to be chosen at random one could think of deriving it by an interactive protocol which still ensures randomness of the common reference string when random oracles are replaced by real hash functions. But this is not the approach chosen here as this additional interactive protocol reduces the efficiency.

In this contribution we use random oracles in a different way to obtain universal composability which ensures the properties binding and hiding even if the

random oracles are replaced by arbitrary collision free hash functions. A random oracle will be used as a function which can be evaluated by every participant of the protocol, but which is not accessible to the environment machine. The equivocability of a bit commitment, which is important for simulatability, can then easily be obtained as in the ideal protocol no random oracle exists and the ideal adversary can determine the outcome of (simulated) evaluations of the random oracle. (Note that in [Nie02], a similar method was used to obtain non-committing encryption in the random oracle model.)

Furthermore this limitation put up on the environment machine will make it impossible for the environment machine to generate commitments to strings unknown to the attacker thereby preventing attacks where the environment machine uses a corrupted party as a *relay* to insert such bit commitments into the protocol. Specifically, we give two constructions to convert a bit commitment scheme into a universally composable bit commitment scheme using random oracles. Both constructions yield bit commitments which remain binding and hiding if the random oracle used is replaced by an arbitrary collision free hash function. The first and more simple construction however does not conserve the property of being perfectly binding, whereas the second construction yields a commitment scheme which is perfectly binding if the original commitment scheme was.

2 Preliminaries

2.1 The General Framework

To start, we shortly outline the framework for multi-party protocols defined in [Can01]. First of all, *parties* (denoted by P_1 through P_n) are modeled as *interactive Turing machines (ITMs)* (cf. [Can01]) and are supposed to run some (fixed) protocol π. There also is an *adversary* (denoted $\mathcal{A}$ and modeled as an ITM as well) carrying out attacks on protocol π. Therefore, $\mathcal{A}$ may corrupt parties (in which case it learns the party's current state and the contents of all its tapes, and controls its future actions), and intercept or, when assuming unauthenticated message transfer, also fake messages sent between parties. If $\mathcal{A}$ corrupts parties only *before* the actual protocol run of π takes place, $\mathcal{A}$ is called *non-adaptive*, otherwise $\mathcal{A}$ is said to be *adaptive*. The respective local inputs for protocol π are supplied by an *environment machine* (modeled as an ITM and denoted $\mathcal{Z}$), which may also read all outputs locally made by the parties and communicate with the adversary. Here we will only deal with environments guaranteeing a polynomial (in the security parameter) number of total steps all participating ITMs run. For more discussion on this issue, cf. [HMQS03b].

The model we have just described is called the *real* model of computation. In contrast to this, the *ideal* model of computation is defined just like the real model, with the following exceptions: we have an additional ITM called the *ideal functionality* $\mathcal{F}$ and being able to send messages to and receive messages from the parties privately (i. e., without the adversary being able to even intercept these messages). The ideal functionality may not be corrupted by the adversary, yet may send messages to and receive messages from it. Furthermore, the parties

$P_1, \ldots, P_n$ are replaced by *dummy parties* $\tilde{P}_1, \ldots, \tilde{P}_n$ which simply forward their respective inputs to $\mathcal{F}$ and take messages received from $\mathcal{F}$ as output. Finally, the adversary in the ideal model is called the *simulator* and denoted $\mathcal{S}$. The only means of attack the simulator has in the ideal model are those of corrupting parties (which has the same effect as in the real model), delaying or even suppressing messages sent from $\mathcal{F}$ to a party, and all actions that are explicitly specified in $\mathcal{F}$. However, $\mathcal{S}$ has no access to the contents of the messages sent from $\mathcal{F}$ to the dummy parties (except in the case the receiving party is corrupted) nor are there any messages actually sent between (uncorrupted) parties $\mathcal{S}$ could intercept. Intuitively, the ideal model of computation (or, more precisely, the ideal functionality $\mathcal{F}$ itself) should represent what we ideally expect a protocol to do. In fact, for a number of standard tasks, there are formulations as such ideal functionalities (see, e. g., [Can01]).

To decide whether or not a given protocol π does what we would ideally expect some ideal functionality $\mathcal{F}$ to do, the framework of [Can01] uses a *simulatability*-based approach: at a time of its choice, $\mathcal{Z}$ may enter its halt state and leave output on its output tape. The random variable describing the first bit of $\mathcal{Z}$'s output will be denoted by $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ when $\mathcal{Z}$ is run on *security parameter* $k \in \mathbb{N}$ and initial input $z \in \{0,1\}^*$ (which may, in case of a non-uniform $\mathcal{Z}$, depend on k) in the real model of computation, and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)$ when $\mathcal{Z}$ is run in the ideal model. Now if for any adversary $\mathcal{A}$ in the real model, there exists a simulator $\mathcal{S}$ in the ideal model such that for *any* environment $\mathcal{Z}$ and *any* initial input z, we have that

$$|\mathbf{P}(\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z) = 1) - \mathbf{P}(\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z) = 1)| \tag{1}$$

is a negligible[1] function in k, then protocol π is said to *securely realize* functionality $\mathcal{F}$.[2] Intuitively, this means that any attack carried out by adversary $\mathcal{A}$ in the real model can also be carried out in the idealized modeling with an ideal functionality by the simulator $\mathcal{S}$ (hence the name), such that no environment is able to tell the difference. By definition, the trivial protocol which does not generate output realizes *any* ideal functionality securely. (The corresponding simulator just has to suppress delivery of messages from the ideal functionality to the parties.) To avoid such trivial realizations, we will only consider *terminating* protocols, which eventually generate output when all protocol messages in the real model are delivered.

To allow for a modular protocol design, in [Can01] also the *$\mathcal{F}$-hybrid model of computation* (for an arbitrary ideal functionality $\mathcal{F}$) is introduced. Briefly, this model is identical to the real model of computation, but the parties have access to an unbounded number of instances of $\mathcal{F}$, each one identified via a *session identifier* (SID). The modularity of the hybrid model is legitimated by the fundamental *composition theorem* of [Can01]. Summarizing, it states that

[1] A function $f : \mathbb{N} \to \mathbb{R}$ is called *negligible*, if for any $c \in \mathbb{N}$, there is a $k_0 \in \mathbb{N}$ such that $|f(k)| < k^{-c}$ for all $k > k_0$.

[2] The formulation in [Can01] is slightly different, but equivalent to the one chosen here which allows to simplify our presentation.

once protocol τ securely realizes functionality $\mathcal{F}$, in any protocol π running in the $\mathcal{F}$-hybrid model, a polynomial number of instances of $\mathcal{F}$ can be substituted by invocations of τ *without losing security*. Specifically, for every real-life adversary $\mathcal{A}$, there is a hybrid-model adversary $\mathcal{H}$ such that no environment can tell whether it is interacting with $\mathcal{A}$ and π (with $\mathcal{F}$-instances substituted by invocations of τ) in the real model, or with $\mathcal{H}$ and π in the $\mathcal{F}$-hybrid model.

2.2 The Common Reference String Model

To catch the notion of information publicly known to all protocol participants, the modeling of [Can01] can be extended to give any participant (including the adversary) access to a *common reference string*, initially chosen from some distribution D. This can be cast as the $\mathcal{F}_{\text{CRS}}$-hybrid model, where $\mathcal{F}_{\text{CRS}}$ denotes the ideal functionality that in its first activation chooses a value d from a distribution D (the latter over which $\mathcal{F}_{\text{CRS}}$ is parameterized). From this point on, it replies to any request from a party P_i or from the adversary with this value d.

2.3 Collision-Free Hash Functions

A family $\mathcal{H} = \{H_k\}_{k\in\mathbb{N}}$ of functions $H_k : \{0,1\}^* \to \{0,1\}^k$ is called a *family of collision-free hash functions*, if the following requirements are met:

- There is a probabilistic algorithm A computing H_k in time polynomial in both k and input length.
- There is *no* probabilistic algorithm B being able to find $x, y \in \{0,1\}^*$ sufficing $x \neq y$ and $H_k(x) = H_k(y)$ in polynomial time with non-negligible probability.

Using the argument in the proof of [Dam90, Lemma 2.1], one can derive a certain *one-way* property: for a family of collision-free hash functions $\mathcal{H} = \{H_k\}$ as above, there can be *no* probabilistic algorithm C which, on input $y = H_k(x) \in \{0,1\}^k$ for uniformly selected $x \in \{0,1\}^{k+1}$, succeeds with non-negligible probability to find $x' \in \{0,1\}^{k+1}$ sufficing $H(x') = y$ in polynomial time.

2.4 The Random Oracle Model

The *random oracle model* (see, e. g., [BR93]) captures an idealization of a hash function. In particular, the idealized version allows only black-box access and cannot be "predicted" without explicitly evaluating it. Moreover, the function values are uniformly selected random k-bit strings. Using the terminology just described, the random oracle model can be modeled in the setting of [Can01] as the $\mathcal{F}_{\text{RO}}$-hybrid model for the ideal functionality $\mathcal{F}_{\text{RO}}$ given in Figure 1. In the presence of more than one party, $\mathcal{F}_{\text{RO}}$ cannot be realized securely *without* interparty communication. (In this case, its very definition forces any protocol aimed at realizing $\mathcal{F}_{\text{RO}}$ to behave like an "almost" deterministic function evaluation; yet such a—by construction easily computable and explicitly given—function can

Functionality $\mathcal{F}_{\mathrm{RO}}$

$\mathcal{F}_{\mathrm{RO}}$ proceeds as follows, running on security parameter k, with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$.

1. $\mathcal{F}_{\mathrm{RO}}$ keeps a list L (which is initially empty) of pairs of bitstrings.
2. Upon receiving a value (sid, m) (with $m \in \{0,1\}^*$) from some party P_i or from $\mathcal{S}$, do:
   - If there is a pair $(m, \tilde{h})$ for some $\tilde{h} \in \{0,1\}^k$ in the list L, set $h := \tilde{h}$.
   - If there is no such pair, choose uniformly $h \in \{0,1\}^k$ and store the pair (m, h) in L.

   Once h is set, reply to the activating machine (i. e., either P_i or $\mathcal{S}$) with (sid, h).

Fig. 1. Functionality $\mathcal{F}_{\mathrm{RO}}$

be distinguished easily from $\mathcal{F}_{\mathrm{RO}}$, which chooses its return values completely at random in each run.) In particular, one cannot hope to securely realize $\mathcal{F}_{\mathrm{RO}}$ by, e. g., a family of collision-free hash functions. Below we will investigate possible consequences of such "imperfect" realizations of $\mathcal{F}_{\mathrm{RO}}$.

2.5 Security Notions for Commitments

First a general remark: for any probabilistic algorithm A we write $A(x; r)$ to indicate execution of A on input $x \in \{0,1\}^*$ and with explicitly supplied random coins $r \in \{0,1\}^*$. Now a *non-interactive string commitment scheme* $C = \{C_k, V_k\}$, indexed by a security parameter $k \in \mathbb{N}$, is a family of polynomial-time (in both k and input length) algorithms C_k and V_k, where the C_k may be probabilistic. We mandate that C_k outputs a tuple (com, dec) of bitstrings com and dec on input $m \in \{0,1\}^*$, while V_k generates output $m \in \{0,1\}^* \cup \{\perp\}$ on input (com, dec). Furthermore, we require:

1. (Meaningfulness.) $V_k(C_k(m)) = m$ for all $k \in \mathbb{N}$ and $m \in \{0,1\}^*$.
2. (Hiding property.) For any $m \in \{0,1\}^*$, let $\{com(m)\}$ denote the distribution $\{com; (com, dec) \leftarrow C_k(m)\}$. We require that for arbitrary $m_1, m_2 \in \{0,1\}^*$ with $|m_1| = |m_2|$, the distributions $\{com(m_1)\}$ and $\{com(m_2)\}$ are computationally indistinguishable. If any two such distributions are also indistinguishable for computationally *unbounded* algorithms, we say that the scheme is *unconditionally hiding*.
3. (Binding property.) There is *no* probabilistic, polynomial-time (in k) algorithm B which is able to produce with non-negligible probability (in k) a tuple (com, dec_1, dec_2) such that $\perp \neq V_k(com, dec_1) \neq V_k(com, dec_2) \neq \perp$. If this holds even for computationally unbounded B, then the scheme is said to be *unconditionally binding*.

It will be convenient to denote by $com_k^C(m)$ (resp. $dec_k^C(m)$) the first (resp., the second) component of C_k's output when run on input m; in particular, com_k^C and dec_k^C can be viewed as probabilistic algorithms.

3 Commitment in the Random Oracle Model

3.1 Motivation

The common reference string model proved extremely useful for realizing general ideal functionalities: in [CLOS02], it is shown that almost any two-party ideal functionality $\mathcal{F}$ can be realized in the $\mathcal{F}_{\text{CRS}}$-hybrid model, under the assumption that trapdoor permutations and augmented two-party non-committing encryption protocols exist. It is also shown there that this result can be extended to the multi-party case when we additionally assume a broadcast channel available (which can also be modeled as an ideal functionality). A key point in the constructions of [CLOS02] is the realization of the commitment functionality $\mathcal{F}_{\text{MCOM}}$ (see also Appendix A) in the $\mathcal{F}_{\text{CRS}}$-hybrid model. Since $\mathcal{F}_{\text{MCOM}}$ cannot be securely realized as a two-party computation in the real model (see [CF01]), one must assume some "helper functionality" such as $\mathcal{F}_{\text{CRS}}$ available. Indeed, in [CF01,DN02,CLOS02], several realizations of different commitment functionalities are described in the common reference string model.

Let's shortly recall which additional features the common reference string is to give us, when having in mind securely realizing, e. g., $\mathcal{F}_{\text{MCOM}}$ (cf. also the discussion in [CLOS02, Section 5]). First, note that at a time one party initiates a commitment in the ideal model, the simulator $\mathcal{S}$ must be able to supply the environment $\mathcal{Z}$ with a valid commitment *without* knowing to which value P_i is actually committed. Furthermore, in case of a corrupted committer, $\mathcal{S}$ must be able to *extract* the committed bit out of a valid commitment. Since alone *choosing* the common reference string must enable the simulator do to so, the whole security of a commitment protocol formulated in the common reference string model relies on the fact that $\mathcal{F}_{\text{CRS}}$ chooses the common reference string *ideally* and in a trusted manner.

Moreover, once we assume "imperfect" implementations of the ideal functionality $\mathcal{F}_{\text{CRS}}$ (i. e., publicly available random strings whose choice may somehow be influenced by an adversary), any protocol which realizes $\mathcal{F}_{\text{MCOM}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model may get insecure in a fatal way: in the extreme case in which an adversary may freely *choose* the common reference string, it can generate "fake" commitments which it can later open as 0 *or* 1, as well as "look into" legitimately generated commitments at wish. Specifically, such imperfect common reference strings can damage the security of the general constructions in [CLOS02] in a serious way. That is, a protocol which is to realize some ideal functionality $\mathcal{F}$ using commitments loses not only its universal composability property, but also may lose security in a very "intuitive" way, since the underlying commitment scheme does so.

One way to avoid this is to use another "helper functionality". In this contribution we will present two constructions in the $\mathcal{F}_{\text{RO}}$-hybrid model where a random oracle $\mathcal{F}_{\text{RO}}$ is available. The constructions of this work allow to turn a given non-interactive string commitment scheme into a non-interactive string commitment scheme in the $\mathcal{F}_{\text{RO}}$-hybrid model which is universally composable. Moreover the constructions ensure the hiding and binding proprties even when

the random oracles are replaced by arbitrary collision free hash functions. The second construction even preserves the property of perfect binding. In fact, when implementing the ideal commitment functionality via a hiding and unconditionally binding commitment scheme, the construction in [CLOS02] (formulated in the framework of [Can01]) for realizing general ideal functionalities is essentially the one presented in [GMW87,Gol02] in the special case of a *secure function evaluation*.

3.2 A Universally Composable Commitment Scheme

First, let's have a look at the abovementioned functionality $\mathcal{F}_{\text{SCOM}}$, which is derived from the functionality $\mathcal{F}_{\text{MCOM}}$ of [CLOS02] (the latter which is also given in Appendix A). A description of $\mathcal{F}_{\text{SCOM}}$ is given in Figure 2. The commitment phase described is different than that of $\mathcal{F}_{\text{MCOM}}$. This is to take into account the following attack, which is described for the case of key exchange in [HMQS03a], and goes back to an argument of Damgård for the case of bit commitment. For this attack, one party P_i is invoked with input b by the environment. Then, before any messages are delivered, the environment instructs the adversary to corrupt P_i and to let it perform the protocol from the start, but using input $b' \neq b$. In the ideal model, P_i's input b is already forwarded to $\mathcal{F}_{\text{MCOM}}$ and can *not* be changed anymore, although P_i gets corrupted later on. On the other hand, in the real model, the bit committed to will be $b' \neq b$, as P_i is "reset" when corrupted and no messages were delivered before. This allows for a "trivial" distinction of real and ideal model for *any* protocol aimed at realizing $\mathcal{F}_{\text{MCOM}}$; we solved this situation by a modified functionality $\mathcal{F}_{\text{SCOM}}$ which lets the adversary decide on the point in time when a `commit` input is accepted. An alternative to our formulation would be to change the framework to let the adversary also delay messages sent from parties *to* the ideal functionality. This approach was taken in [CLOS02, revision dated July 14th].

Notice that different commitments are handled via different subsession identifiers, each one of them handling at most one commitment per committer–receiver pair. Furthermore, $\mathcal{F}_{\text{SCOM}}$ allows committing to a string of bits rather than only to a single bit. It is worthwhile to point out that *no* information (not even length information) about the string m committed to is given to the adversary.

Now assume that $C = \{C_k, V_k\}$ is a non-interactive string commitment scheme as described in Section 2.5. Consider protocol HC_C given in Figure 3. This protocol is formulated in the $\mathcal{F}_{\text{RO}}$-hybrid model and aimed at realizing the ideal functionality $\mathcal{F}_{\text{SCOM}}$.

Proposition 1. *Assuming authenticated links, protocol* HC_C *securely realizes* $\mathcal{F}_{\text{SCOM}}$ *with respect to adaptive adversaries as soon as* $C = \{C_k, V_k\}$ *is a non-interactive string commitment scheme.*

Proof. For any adversary $\mathcal{H}$ mounting attacks on protocol HC_C in the $\mathcal{F}_{\text{RO}}$-hybrid model, we describe a simulator $\mathcal{S} = \mathcal{S}_{\mathcal{H}}$ emulating such attacks in the ideal model. $\mathcal{S}$ internally keeps a *complete* simulation of a run of $\mathcal{H}$ in the $\mathcal{F}_{\text{RO}}$-hybrid model. That is, $\mathcal{S}$ keeps a simulation of parties $P_1^{(s)}$ through $P_n^{(s)}$ running

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

$\mathcal{F}_{\text{SCOM}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$:

- **Commit Phase**:
  1. When receiving a message $(\texttt{commit}, sid, ssid, P_i, P_j, m)$ from P_i, where $m \in \{0,1\}^*$, first send the message $(\texttt{request}, sid, ssid, P_i, P_j)$ to the adversary $\mathcal{S}$ and, if $\mathcal{S}$ then issues a corresponding `ready` message (see below), proceed to the third step.
  2. When receiving $(\texttt{ready}, sid, ssid, P_i, P_j, \ell)$ from the adversary, and at least ℓ messages $(\texttt{commit}, sid, ssid, P_i, P_j, m)$ (possibly with different m's) have been received from P_i, perform the third step described below with the message m contained in the ℓth of these messages.
  3. Record the tuple $(ssid, P_i, P_j, m)$ and send $(\texttt{receipt}, sid, ssid, P_i, P_j)$ to P_j. Ignore any future `commit` messages with the same $ssid$ from P_i to P_j.
- **Reveal Phase**: Upon receiving a message $(\texttt{reveal}, sid, ssid, P_j)$ from P_i: If a tuple $(ssid, P_i, P_j, m)$ was previously recorded, then send the message $(\texttt{reveal}, sid, ssid, P_i, P_j, m)$ to P_j and $\mathcal{S}$. Otherwise, ignore.

Fig. 2. Functionality $\mathcal{F}_{\text{SCOM}}$

Protocol HC_C

These are instructions for parties P_1 through P_n to carry out commitments. The parties expect to be run in the $\mathcal{F}_{\text{RO}}$-hybrid model. For ease of notation, here $\mathcal{O}_{sid}(x)$ denotes the reply of the $\mathcal{F}_{\text{RO}}$-instance with session ID sid to the query x.

- When activated with input $(\texttt{commit}, sid, ssid, P_i, P_j, m)$, where $m \in \{0,1\}^*$, P_i computes $(com_1, dec_1) \leftarrow C_k(\mathcal{O}_{sid}(ssid, i, j, m, r_1); r_2)$ for a uniformly chosen k-bit string r_1. Note that by $r_2 \in \{0,1\}^*$, we denote the random coins used by C_k during this process. Then, P_i computes $(com_2, dec_2) \leftarrow C_k(\mathcal{O}_{sid}(r_2))$ and sends the message $(sid, ssid, com_1, com_2)$ to P_j while storing $(ssid, j, m, r_1, r_2, dec_2)$. Further $(\texttt{commit}, sid, ssid, P_i, P_j, \cdot)$ inputs are ignored.
- When receiving $(sid, ssid, com_1, com_2)$ from P_i, where $com_1, com_2 \in \{0,1\}^*$ and $ssid$ is a subsession ID under which P_j did not yet get such a message from any party, P_j stores the pair $(ssid, i, com_1, com_2)$ and locally outputs $(\texttt{receipt}, sid, ssid, P_i, P_j)$. Any future messages $(sid, ssid, com_1', com_2')$ with the same subsession ID $ssid$ from P_i are ignored.
- When activated on input $(\texttt{reveal}, sid, ssid, P_j)$, party P_i checks if it has a tuple $(ssid, j, m, r_1, dec_2)$ (for any m, r_1, dec_2) stored. If so, P_i sends the tuple $(sid, ssid, m, r_1, r_2, dec_2)$ to P_j. Further inputs $(\texttt{reveal}, sid, ssid, P_j)$ are ignored.
- When receiving $(sid, ssid, m, r_1, r_2, dec_2)$ with $m, dec_2, r_1, r_2 \in \{0,1\}^*$ from P_i while already having received a value $(sid, ssid, com_1, com_2)$ also from P_i, P_j first computes $o_2 \leftarrow V_k(com_2, dec_2)$. Then, if $o_2 = \mathcal{O}_{sid}(r_2)$, P_j checks if $com_k^C(\mathcal{O}_{sid}(ssid, i, j, m, r_1); r_2)$ equals com_1. If so, P_j locally outputs $(\texttt{reveal}, sid, ssid, P_i, P_j, m)$ and ignores all further $(sid, ssid, \ldots)$ messages. In any other case, P_j does nothing.

Fig. 3. Protocol HC_C

protocol HC_C, a simulation of $\mathcal{H}$ interacting with these parties, and (as needed) simulated instances of the ideal functionality $\mathcal{F}_{\text{RO}}$. Communication of $\mathcal{H}$ with the environment is forwarded to the (non-simulated) environment $\mathcal{Z}$ with which $\mathcal{S}$ is to interact. Similarly, messages from $\mathcal{Z}$ to $\mathcal{S}$ are forwarded to the adversary $\mathcal{H}$ in the simulation.

Of course, $\mathcal{S}$ still needs to keep its simulation consistent with all inputs the dummy parties receive from $\mathcal{Z}$; similarly, the output behaviour of the dummy parties has to be the same as that of the simulated parties. (Note that generally, $\mathcal{S}$ has no information about inputs and only existence information about outputs of the dummy parties, unless the ideal functionality explicitly informs $\mathcal{S}$ about incoming input, or about output sent to the parties.) Therefore, $\mathcal{S}$ acts as follows:

- When the simulated $\mathcal{H}$ corrupts a simulated party $P_i^{(s)}$, $\mathcal{S}$ first corrupts the corresponding dummy party P_i and modifies $P_i^{(s)}$'s state to account for ignored inputs possibly handed from $\mathcal{Z}$ to P_i. (Upon corruption of P_i, $\mathcal{S}$ gets to know about such messages when receiving the state of P_i.)
- Upon receiving $(\texttt{receipt}, sid, ssid, P_i, P_j)$ from $\mathcal{F}_{\text{SCOM}}$ (in which case P_i and thus $P_i^{(s)}$ must be uncorrupted), $\mathcal{S}$ picks $s \in \{0,1\}^k$ uniformly and computes $com_1 \leftarrow com_k^C(s; r_2)$ and $com_2 \leftarrow com_k^C(\mathcal{O}_{sid}(r_2); r_3)$ for a uniformly chosen r_3 (here the simulated random oracle $\mathcal{F}_{\text{RO}}$ is queried). Then, $\mathcal{S}$ simulates a message $(sid, ssid, com_1, com_2)$ from $P_i^{(s)}$ to $P_j^{(s)}$ and stores this message together with r_2, r_3, and s.
- When $\mathcal{H}$ delivers a message $(sid, ssid, com_1, com_2)$ from $P_i^{(s)}$ to $P_j^{(s)}$, and $P_j^{(s)}$ did not yet get such a message with subsession ID $ssid$ from P_i, $\mathcal{S}$ proceeds as follows: if $P_i^{(s)}$ (and hence P_i as well) is uncorrupted, $\mathcal{S}$ sends $(\texttt{ready}, sid, ssid, P_i, P_j, 1)$ to $\mathcal{F}_{\text{SCOM}}$, and delivers the `receipt` message then sent from $\mathcal{F}_{\text{SCOM}}$ to P_j as soon as $P_j^{(s)}$ outputs such a `receipt` message (i. e., immediately afterwards).

  If, on the other hand, $P_i^{(s)}$ (and thus P_i) is corrupted, then first the message m committed to may have to be extracted from com_1 and com_2 which could have been supplied by $\mathcal{Z}$ without appropriate commit input. By looking up all queries to the simulated $\mathcal{F}_{\text{RO}}$-instance with session ID sid, we can reduce to a polynomial number of possible m-r_1-r_2-combinations. Hence by verifying whether com_1 equals $com_k^C(\mathcal{O}_{sid}(ssid, i, j, m, r_1); r_2)$, $\mathcal{S}$ can extract m alone from com_1, provided that the commitment can be unveiled according to HC and $\mathcal{F}_{\text{RO}}$ did never output the same value twice. (Note that here we use the binding property of C.) In these latter cases, it suffices to set m to 0 (or any other value), since $\mathcal{F}_{\text{RO}}$ produces collisions only in a negligible fraction of runs and commitments generated without explicitly querying the random oracle can be unveiled only with negligible probability. (Here it is important that since the subsession identifier sid and the party identities i and j are hashed together with m, hash values cannot be "re-used" in a different subsession.)

  Once m is determined, $\mathcal{S}$ sends $(\texttt{commit}, sid, ssid, P_i, P_j, m)$ in the name of the corrupted relay P_i to functionality $\mathcal{F}_{\text{SCOM}}$, followed by a corresponding

(`ready`, sid, $ssid$, P_i, P_j, ℓ) signal. Here ℓ denotes the number of already received `request` notifications plus 1, and thus indicates that the message m just sent is to be committed to. This causes the ideal functionality to send a `receipt` message to P_j, which then can be delivered by $\mathcal{S}$ as soon as $P_j^{(s)}$ generates output in the simulation.

– Upon receiving (`reveal`, sid, $ssid$, P_i, P_j, m) from $\mathcal{F}_{\text{SCOM}}$ (which means that P_i is still uncorrupted), $\mathcal{S}$ lets $P_i^{(s)}$ compute a commitment to m at P_j (under session ID sid and subsession ID $ssid$), but forces
  - the simulated $\mathcal{F}_{\text{RO}}$ with session ID sid to output s when queried by $P_i^{(s)}$ with $(ssid, i, j, m, r_1)$ (this is not possible when $\mathcal{F}_{\text{RO}}$ was queried on $(ssid, i, j, m, r_1)$ before; yet, since r_1 is chosen uniformly from $\{0,1\}^k$ by $P_i^{(s)}$, this only occurs with negligible probability)
  - $P_i^{(s)}$ to use the values r_2 and r_3 which $\mathcal{S}$ stored together with the message $(sid, ssid, j, com_1, com_2)$.

  Now $\mathcal{S}$ dismisses the actual commitment message sent from $P_i^{(s)}$ to $P_j^{(s)}$ (note that by construction, this message is exactly the message simulated by $\mathcal{S}$) and modifies $P_i^{(s)}$'s internal state so as to look as if this commitment had been performed exactly at the time the corresponding message $(sid, ssid, com_1, com_2)$ was simulated from $P_i^{(s)}$ to $P_j^{(s)}$.
  Finally, P_i is fed with input (`reveal`, sid, $ssid$, P_i, P_j, m) to reflect in the simulation the actual decommitment operation $\mathcal{S}$ was informed about. The `reveal` message sent from $\mathcal{F}_{\text{SCOM}}$ to P_j is delivered as soon as $P_j^{(s)}$ generates as output the corresponding `reveal` message.
– The same procedure is applied to all commitments P_i has not yet opened when $P_i^{(s)}$ gets corrupted. (Note that then, $\mathcal{S}$ gets to know the corresponding messages committed to by receiving P_i's state.)
– Finally, if $P_j^{(s)}$ generates `reveal` output while uncorrupted, the corresponding dummy party P_j has to generate output as well. The steps above take care that this output is the same in the ideal model as in the simulation, except in a negligible fraction of runs. Particularly, $\mathcal{S}$ only needs to deliver the corresponding `reveal` message from $\mathcal{F}_{\text{SCOM}}$ to P_j when the corresponding committer P_i was uncorrupted at the time it got its `reveal` input. If, on the other hand, P_i was corrupted at that time, $\mathcal{S}$ must first supply $\mathcal{F}_{\text{SCOM}}$ with the corresponding `reveal` input via the corrupted relay P_i. In this case, $\mathcal{S}$ has already extracted the message m needed for this `reveal` input either from P_i's state (when P_i's corruption took place *after* the delivery of the commitment), or, otherwise, from the actual commitment message delivered to $P_j^{(s)}$. (Here we use that except with negligible probability, there is no efficient way to unveil a commitment in more than one way; by construction of protocol HC_C, this follows from the "collision-freeness" of $\mathcal{F}_{\text{RO}}$.)

By construction, $\mathcal{S}$ provides $\mathcal{Z}$ with a view *identical* to one of a run in the hybrid model, until a `reveal` output of a party P_j differs from that of the respective simulated party $P_j^{(s)}$. However, this can only happen when $\mathcal{S}$ is unable to extract

a message out of a commitment sent from a corrupted committer $P_i^{(s)}$ to an uncorrupted receiver $P_j^{(s)}$, or when $\mathcal{S}$ cannot unveil a commitment generated by $\mathcal{S}$ itself. As reasoned above, the probability for any of these is only negligible, henceforth we are done. Note that we did *not* use the hiding property of C. $\square$

For achieving universal composability, we had to incorporate subsession identifiers and the identities of committer and receiver into the message actually committed to. To allow for statements independent of such protocol-inherent information, we drop that requirement on the format of the message and, to be able to formally view the protocol HC_C as a non-interactive string commitment scheme, we set $\text{HC}_C = \{\mathcal{C}_k, \mathcal{V}_k\}$. Here algorithm $\mathcal{C}_k$ computes $com \leftarrow (com_k^C(\mathcal{O}(m, r_1); r_2), com_k^C(\mathcal{O}(r_2); r_3))$ for uniformly chosen $r_1 \in \{0,1\}^k$ and random coins $r_2, r_3 \in \{0,1\}^*$, then $dec \leftarrow (m, r_1, r_2, dec_k^C(\mathcal{O}(r_2); r_3))$ and returns (com, dec). On input (com, dec) of the form $com = (com_1, com_2)$ and $dec = (m, r_1, r_2, dec_2)$, algorithm $\mathcal{V}_k$ computes $o_2 \leftarrow V_k(com_2, dec_2)$ and, if $o_2 = \mathcal{O}(r_2)$, checks whether com_1 equals $C_k(\mathcal{O}(m, r_1); r_2)$. Only in this case $\mathcal{V}_k$ returns m, otherwise it returns $\perp$.

As mentioned above, we would like to be able to deduce security properties of the scheme HC_C even when having substituted all random oracles $\mathcal{O}$ by suitable hash functions. Let therefore $\text{HC}_{C,\mathcal{H}}$ denote the scheme which is identical to HC_C, except that all $\mathcal{O}$-queries are replaced by evaluations of H_k, where $\mathcal{H} = \{H_k\}$ is a family of collision-free hash functions as defined in the preliminaries.

Proposition 2. *Once $\mathcal{H} = \{H_k\}$ is a family of collision-free hash functions and $C = \{C_k, V_k\}$ is a non-interactive string commitment scheme, the scheme $\text{HC}_{C,\mathcal{H}} = \{\mathcal{C}_k, \mathcal{V}_k\}$ (as described above) is also a non-interactive string commitment scheme. Furthermore, if C is unconditionally hiding, then so is $\text{HC}_{C,\mathcal{H}}$.*

Proof. Meaningfulness, binding and hiding properties of $\text{HC} = \text{HC}_{C,\mathcal{H}}$ need to be checked. The meaningfulness of HC follows directly from that of C. Furthermore, every algorithm B which supplies a HC-commitment together with two decommitments yielding different messages has to supply in particular a C-commitment together with two C-decommitments yielding messages $H_k(m, r_1)$, resp. $H_k(m', r_1')$. If both are equal, B has found a H_k-collision (since $m \neq m'$ by assumption); if the hash values are different, then B breaks the binding property of C. In either case, we have shown the binding property of HC.

Now for the hiding property, consider the scheme $\text{HC}' = \{\mathcal{C}_k', \mathcal{V}_k'\}$, which is identical to HC, except that the C-commitment to $H_k(r_2)$ (where r_2 denotes the random coins used in the C-commitment to $H_k(m, r_1)$) is replaced by a C-commitment to 0^k. More formally, $\mathcal{C}_k'(m)$ computes to (com', dec') with $com' \leftarrow (com_k^C(H_k(m, r_1); r_2), com_k^C(0^k))$ and $dec' \leftarrow (m, r_1, r_2)$; the definition of $\mathcal{V}_k'$ is obvious.

Let A be a probabilistic, polynomial-time algorithm which breaks the computational hiding property of HC. More specifically, say that there are messages $m_1, m_2 \in \{0,1\}^*$, such that the difference

$$\mathbf{Adv}(A, \text{HC}, m_1, m_2) := \mathbf{P}(A(com_k^{\text{HC}}(m_1)) \to 1) - \mathbf{P}(A(com_k^{\text{HC}}(m_2)) \to 1)$$

is a non-negligible function in k. Assume first that

$$\mathbf{Adv}(A, \text{HC}', m_1, m_2) := \mathbf{P}(A(com_k^{\text{HC}'}(m_1)) \to 1) - \mathbf{P}(A(com_k^{\text{HC}'}(m_2)) \to 1)$$

is non-negligible in k as well. Since by construction, a C-commitment to a message $H_k(m, r_1)$ can be extended to an HC′-commitment to the message m *without* knowledge of m or r_1, it follows that from A, we can build a probabilistic, polynomial-time algorithm A_1 with

$$\mathbf{P}(A_1(com_k^C(H_k(m_1, r_1)) \to 1)) - \mathbf{P}(A_1(com_k^C(H_k(m_2, r_1)) \to 1)$$

non-negligible in k for a certain, *fixed* $r_1 \in \{0,1\}^k$. Such an A_1 would break the hiding property of C, thereby yielding a contradiction.

On the other hand, suppose that $\mathbf{Adv}(A, \text{HC}', m_1, m_2)$ is negligible in k. As then,

$$\begin{aligned} & \mathbf{Adv}(A, \text{HC}, m_1, m_2) - \mathbf{Adv}(A, \text{HC}', m_1, m_2) \\ = \quad & \Big(\mathbf{P}(A(com_k^{\text{HC}}(m_1)) \to 1) - \mathbf{P}(A(com_k^{\text{HC}'}(m_1)) \to 1)\Big) \\ - \quad & \Big(\mathbf{P}(A(com_k^{\text{HC}}(m_2)) \to 1) - \mathbf{P}(A(com_k^{\text{HC}'}(m_2)) \to 1)\Big) \end{aligned}$$

is non-negligible, at least one of the addends on the right-hand side of the equation must be as well. So say that $\mathbf{P}(A(com_k^{\text{HC}}(m_i)) \to 1) - \mathbf{P}(A(com_k^{\text{HC}'}(m_i)) \to 1)$ is non-negligible (with fixed $i \in \{1,2\}$). Then there have to be certain, *fixed* r_1, r_2 for which A can distinguish tuples $(com_k^C(H_k(m_i, r_1); r_2), com_k^C(H_k(r_2)))$ from tuples $(com_k^C(H_k(m_i, r_1); r_2), com_k^C(0^k))$. Hence from A we can construct a probabilistic, polynomial-time algorithm A_2 with

$$\mathbf{P}(A_2(com_k^C(H_k(r_2)) \to 1)) - \mathbf{P}(A_2(com_k^C(0^k)) \to 1)$$

non-negligible in k, thereby breaking the hiding property of C. So in either case, we have a contradiction and there can be no such algorithm A; consequently, HC must be computationally hiding. As this reduction also applies to computationally unbounded algorithms A, a possible unconditional hiding property of C is preserved. □

3.3 Preserving Unconditional Binding

Although we have shown that the construction $\text{HC}_{C,\mathcal{H}}$ preserves hiding and computational binding properties of C, this is *not* true for a potential *unconditional* binding property: Any algorithm breaking the collision-freeness of $\mathcal{H}$ can be used to generate HC-commitments together with multiple decommitments to different messages. (Note that with HC, we are actually committed only to the hash value of a message.)

An unconditionally binding string commitment scheme cannot completely hide the length of the message committed to; to reflect this in an idealization, we define the ideal functionality $\mathcal{F}_{\text{BSCOM}}$ to be identical to $\mathcal{F}_{\text{SCOM}}$ (cf. Figure 2),

except that $\mathcal{F}_{\text{BSCOM}}$ supplies the simulator $\mathcal{S}$ upon a commitment to a message m with the bit length $|m|$ of this message. This length information is included in the respective `request` message sent to the simulator.

Let $F_{\mathcal{H}}$ be the following probabilistic, polynomial-time (in both the security parameter k and its input length) algorithm, where $\mathcal{H} = \{H_k\}_k$ is a family of functions $H_k : \{0,1\}^* \to \{0,1\}^k$ which in turn are computable in polynomial time. Upon input $m = m_1 \cdots m_n \in \{0,1\}^n$, $F_{\mathcal{H}}$ uniformly selects $s_1, \ldots, s_n \in \{0,1\}^k$ with each s_i satisfying $s_i H_k(s_i) \neq H_k(s_i) s_i$ and outputs

$$F_{\mathcal{H}}(m) = \pi(m_1, s_1, H_k(s_1)) \cdots \pi(m_n, s_n, H_k(s_n)),$$

where $\pi(0, s, t) = (s, t)$ and $\pi(1, s, t) = (t, s)$ for arbitrary $s, t \in \{0,1\}^*$. Algorithm $F_{\mathcal{H}}$ will be used to encode a message in a (for the simulator) equivocable yet (for parties) binding way. To extract the encoded message, we will use the following polynomial-time computable function $F_{\mathcal{H}}^{-1}$. We set

$$F_{\mathcal{H}}^{-1}(m) = \begin{cases} 0 \text{ if } r = H_k(l) \text{ and } l \neq H_k(r) \\ 1 \text{ if } r \neq H_k(l) \text{ and } l = H_k(r) \\ \perp \text{ else} \end{cases}$$

for $m = lr$ with $l, r \in \{0,1\}^k$. For $m = m_1 \cdots m_\ell$ with $\ell > 1$ and all $m_i \in \{0,1\}^{2k}$, we define $F_{\mathcal{H}}^{-1}(m) = F_{\mathcal{H}}^{-1}(m_1) \cdots F_{\mathcal{H}}^{-1}(m_\ell)$. In all other cases, we set $F_{\mathcal{H}}^{-1}(m) = \perp$.

Given a non-interactive string commitment scheme C, consider a protocol BHC_C, whose "infrastructure" is identical to that of protocol HC_C, but the commitment and decommitment messages differ slightly, as does the verification procedure. Protocol BHC_C is described in Figure 4.

Proposition 3. *Assuming authenticated links, protocol* BHC_C *securely realizes* $\mathcal{F}_{\text{BSCOM}}$ *with respect to adaptive adversaries as soon as* $C = \{C_k, V_k\}$ *is a non-interactive string commitment scheme.*

Proof. The proof is very similar to the one of Proposition 1, and we will only describe the necessary modifications to the simulator $\mathcal{S}$. For generating an equivocable commitment to a message of length ℓ (here we use the length information with which $\mathcal{F}_{\text{BSCOM}}$ supplies $\mathcal{S}$ upon commitment requests), $\mathcal{S}$ picks $s \in \{0,1\}^k$, $f \in \{0,1\}^{2\ell k}$, then generates C-commitments $C_k(s, f; r_2)$ and $C_k(\mathcal{O}_{sid}(r_2))$ and with these simulates a commitment as before. Later, when being forced to unveil this commitment as a commitment to $m \in \{0,1\}^\ell$, $\mathcal{S}$ lets the simulated committer $P_i^{(s)}$ perform a commitment to m as before, *but* forces

- $P_i^{(s)}$ to compute $F_{\mathcal{O}_{sid}}(m)$ as the given f by altering $P_i^{(s)}$'s random tape, resp. the random tape of $\mathcal{F}_{\text{RO}}$
- the simulated $\mathcal{F}_{\text{RO}}$ to output s when queried by $P_i^{(s)}$ on $(ssid, i, j, f)$.

Again, this "tampering" with $\mathcal{F}_{\text{RO}}$ is possible only if $\mathcal{F}_{\text{RO}}$ was not queried on any of these values before. By the hiding property of C and the randomization

Protocol BHC_C

These are instructions for parties P_1 through P_n to carry out commitments in the $\mathcal{F}_{\text{RO}}$-hybrid model.

- When activated with input $(\texttt{commit}, sid, ssid, P_i, P_j, m)$, where $m \in \{0,1\}^*$, P_i computes $f \leftarrow F_{\mathcal{O}_{sid}}(m)$ and $(com_1, dec_1) \leftarrow C_k(\mathcal{O}_{sid}(ssid, i, j, f), f; r_2)$. Then, P_i computes $(com_2, dec_2) \leftarrow C_k(\mathcal{O}_{sid}(r_2))$ and sends the message $(sid, ssid, com_1, com_2)$ to P_j while storing $(ssid, j, f, r_2, dec_2)$. Any further $(\texttt{commit}, sid, ssid, P_i, P_j, \cdot)$ inputs are ignored.
- When receiving $(sid, ssid, com_1, com_2)$ from P_i, where $com_1, com_2 \in \{0,1\}^*$ and $ssid$ is a subsession ID under which P_j did not yet get such a message from any party, P_j stores the pair $(ssid, i, com_1, com_2)$ and locally outputs $(\texttt{receipt}, sid, ssid, P_i, P_j)$. Any future messages $(sid, ssid, com_1', com_2')$ from P_i (with the same sid and $ssid$) are ignored.
- When activated on input $(\texttt{reveal}, sid, ssid, P_j)$, P_i checks if it has stored a tuple $(ssid, j, f, r_2, dec_2)$ (for any f, dec_2). If so, P_i sends $(sid, ssid, f, dec_2)$ to P_j. Any future inputs $(\texttt{reveal}, sid, ssid, P_j)$ are ignored.
- When receiving $(sid, ssid, f, r_2, dec_2)$ with $f, r_2, dec_2 \in \{0,1\}^*$ from P_i while already having received a value $(sid, ssid, com_1, com_2)$ also from P_i, party P_j first computes $o_2 \leftarrow V_k(com_2, dec_2)$. Then, if $o_2 = \mathcal{O}_{sid}(r_2)$, P_j checks if $com_k^C(\mathcal{O}_{sid}(ssid, i, j, m, f), f; r_2)$ equals com_1 and if $m \neq \perp$ for $m \leftarrow F_{\mathcal{H}}^{-1}(f)$. If so, P_j locally outputs $(\texttt{reveal}, sid, ssid, P_i, P_j, m)$ and ignores further $(sid, ssid, \ldots)$ messages. In any other case, P_j does nothing.

Fig. 4. Protocol BHC_C

of $F_{\mathcal{O}_{sid}}$, this is guaranteed to occur only negligibly often. Note here also that both hash values s and f are valid for only one subsession (i. e., for one single commitment).

The extraction of the message committed to from a commitment that was generated according to BHC_C is straightforward; again, if a commitment was *not* generated as specified by BHC_C, it can only be unveiled with negligible probability. With these changes, the proof of Proposition 1 applies. □

Once again, by dropping protocol-inherent information, protocol BHC_C may formally be regarded as a non-interactive commitment scheme. Therefore, we set $\text{BHC}_C = \{\mathcal{C}_k, \mathcal{V}_k\}$, where algorithm $\mathcal{C}_k$ computes $f \leftarrow F_{\mathcal{O}}(m)$, then $com \leftarrow (com_k^C(\mathcal{O}(f), f; r_2), com_k^C(\mathcal{O}(r_2); r_3))$ and $dec \leftarrow (f, r_2, dec_k^C(\mathcal{O}(r_2); r_3))$ for random coins $r_2, r_3 \in \{0,1\}^*$, and finally returns (com, dec). On input (com, dec) of the form $com = (com_1, com_2)$ and $dec = (f, r_2, dec_2)$, algorithm $\mathcal{V}_k$ computes $o_2 \leftarrow V_k(com_2, dec_2)$ and, if $o_2 = \mathcal{O}(r_2)$, checks whether com_1 equals $C_k(\mathcal{O}(f), f; r_2))$. Only in this case $\mathcal{V}_k$ returns $F_{\mathcal{O}}^{-1}(f)$, otherwise it returns $\perp$.

As will be shown, BHC_C preserves not only hiding properties of C, but also a possible unconditional binding property, albeit for the price of being less efficient than HC_C and leaking information about the length $|m|$ of the message being committed to.

Proposition 4. *Once $\mathcal{H} = \{H_k\}$ is a family of collision-free hash functions for which $F_{\mathcal{H}}$ is efficiently computable and $C = \{C_k, V_k\}$ is a non-interactive string commitment scheme, the scheme $\text{BHC}_{C,\mathcal{H}} = \{\mathcal{C}_k, \mathcal{V}_k\}$ (as described above) is also a non-interactive string commitment scheme. If C is unconditionally hiding, then so is $\text{BHC}_{C,\mathcal{H}}$. If C is unconditionally binding, then so is $\text{BHC}_{C,\mathcal{H}}$.*

Proof. The meaningfulness of $\text{BHC}_{C,\mathcal{H}} = \text{BHC}$ follows from that of C. The computational binding property of BHC follows as in the proof of Proposition 2; furthermore, as $F_{\mathcal{H}}^{-1}(F_{\mathcal{H}}(m)) = m$ for any $\mathcal{H}, m$, the same argument shows that BHC is unconditionally binding if C is. Also for the hiding properties of BHC, the argument of the proof of Proposition 2 applies when we set $\text{BHC}' = \{\mathcal{C}'_k, \mathcal{V}'_k\}$. When being run on input m, algorithm $\mathcal{C}'_k$ computes the tuple (com', dec') with $com' \leftarrow (com_k^C(H_k(f), f; r_2), com_k^C(0^k; r_3))$ and $dec' \leftarrow (f, r_2)$ for $f \leftarrow F_{\mathcal{H}}(m)$. The definition of $\mathcal{V}'_k$ is obvious. □

4 Conclusions

In the model of [Can01] bit commitment cannot be securely realized without additional assumptions, e. g. the availability of an additional functionality like a common reference string or, as proposed in this work, a random oracle. As a motivation for the use of random oracles we discussed difficulties which may arise when a common reference string functionality is replaced by a cryptographic primitive which is realizable from scratch.

This contribution gave two constructions which allow to turn a given non-interactive bit commitment into a universally composable commitment scheme in the random oracle model. The resulting commitment schemes remain binding and hiding even if the random oracles are replaced by collision resistant hash functions. The second construction even preserves the property of perfect binding.

One referee pointed out that a separation of the random oracle model and the CRS model is a consequence of our result. Namely, [DG03] showed that from the existence of a universally composable bit commitment in a CRS model, a secure key exchange protocol can be derived. However, in the random oracle model, we proved that a binding and concealing bit commitment can be transformed into a universally composable one. So if a random oracle could be implemented using a common reference string (drawn from a suitable distribution), the existence of a binding and concealing bit commitment alone would imply a secure key exchange protocol. (In fact, it seems that a universally composable bit commitment can be implemented in the random oracle model without any further assumptions, thus yielding a stronger separation.) On the other hand, implementing a common reference string in the random oracle model can be—depending on the distribution of the reference string—non-trivial.

It is an interesting open question how the constructions given here affect the *non-malleability* of a given commitment scheme. To the best of our knowledge it is not clear how relations among committed values behave with respect to the use of hash functions in the given constructions.

Acknowledgements

The authors would like to thank Rainer Steinwandt and Dominique Unruh for interesting and valuable discussions, and the anonymous referees for helpful comments.

References

[BBP03] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. IACR ePrint Archive, August 2003. Online available at `http://eprint.iacr.org/2003/077.ps`.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security, Proceedings of CCS 1993*, pages 62–73. ACM Press, 1993. Full version online available at `http://www.cs.ucsd.edu/users/mihir/papers/ro.ps`.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001. Full version online available at `http://www.eccc.uni-trier.de/eccc-reports/2001/TR01-016/revisn01.ps`.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology, Proceedings of CRYPTO 2001*, number 2139 in Lecture Notes in Computer Science, pages 19–40. Springer-Verlag, 2001. Full version online available at `http://eprint.iacr.org/2001/055.ps`.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Thirtieth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1998*, pages 209–218. ACM Press, 1998. Preliminary version, extended version online available at `http://eprint.iacr.org/1998/011.ps`.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2002*, pages 494–503. ACM Press, 2002. Extended abstract, full version online available at `http://eprint.iacr.org/2002/140.ps`.

[Dam90] Ivan Bjerre Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology, Proceedings of CRYPTO '89*, number 435 in Lecture Notes in Computer Science, pages 416–427. Springer-Verlag, 1990.

[DG03] Ivan Damgård and Jens Groth. Non-interactive and reusable non-malleable commitment schemes. In *35th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2003*, pages 426–437. ACM Press, 2003. Full version online available at `http://eprint.iacr.org/2003/080.ps`.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In Moti Yung, editor, *Advances in Cryptology, Proceedings of CRYPTO 2002*, number 2442 in Lecture Notes in Computer Science,

pages 581–596. Springer-Verlag, 2002. Full version online available at `http://eprint.iacr.org/2001/091`.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game—a completeness theorem for protocols with honest majority. In *Nineteenth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1987*, pages 218–229. ACM Press, 1987. Extended abstract.

[Gol02] Oded Goldreich. Secure multi-party computation. Unpublished, online available at `http://www.wisdom.weizmann.ac.il/~oded/PS/prot.ps`, October 2002.

[GTK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2003*, pages 102–113. IEEE Computer Society, 2003. Full version online available at `http://eprint.iacr.org/2003/034.ps`.

[HMQS03a] Dennis Hofheinz, Jörn Müller-Quade, and Rainer Steinwandt. Initiator-resilient universally composable key exchange. In Einar Snekkenes and Dieter Gollmann, editors, *Computer Security, Proceedings of ESORICS 2003*, number 2808 in Lecture Notes in Computer Science, pages 61–84. Springer-Verlag, 2003. Online available at `http://eprint.iacr.org/2003/063.ps`.

[HMQS03b] Dennis Hofheinz, Jörn Müller-Quade, and Rainer Steinwandt. On modeling IND-CCA security in cryptographic protocols. IACR ePrint Archive, February 2003. Online available at `http://eprint.iacr.org/2003/024.ps`.

[Nie02] Jesper B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology, Proceedings of CRYPTO 2002*, number 2442 in Lecture Notes in Computer Science, pages 111–126. Springer-Verlag, 2002.

A The functionality $\mathcal{F}_{\text{MCOM}}$

For convenience, here we reproduce the description of the ideal functionality $\mathcal{F}_{\text{MCOM}}$ from [CLOS02]:

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

$\mathcal{F}_{\text{MCOM}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$:

- **Commit Phase**: Upon receiving a message `(commit,`sid`,`$ssid$`,`P_i`,`P_j`,`b`)` from P_i, where $b \in \{0,1\}$, record the tuple $(ssid, P_i, P_j, b)$ and send the message `(receipt,`sid`,`$ssid$`,`P_i`,`P_j`)` to P_j and $\mathcal{S}$. Ignore any future `commit` messages with the same $ssid$ from P_i to P_j.
- **Reveal Phase**: Upon receiving a message `(reveal,`sid`,`$ssid$`)` from P_i: If a tuple $(ssid, P_i, P_j, b)$ was previously recorded, then send the message `(reveal,`sid`,`$ssid$`,`P_i`,`P_j`,`b`)` to P_j and $\mathcal{S}$. Otherwise, ignore.