Karlsruhe Institute of Technology

KASTEL

Composable Definitions of Long-Term Security for Commitment Schemes and their Applications

Master's Thesis

KIT – Karlsruher Institut für Technologie
ITI – Institut für Theoretische Informatik
Forschungsruppe Kryptographie und Sicherheit

Sarai Eilebrecht

28. Februar 2021

Verantwortlicher Betreuer: Prof. Dr. Jörn Müller-Quade
Betreuender Mitarbeiter: Jeremias Mechler, M. Sc.
Astrid Ottenhues, M. Sc.

Abstract.

What happens if a cryptographic assumption turns out not to hold and in which way does it affect the security of cryptographic protocols?

One might consider on updating the security assumption and prove the security of the updated protocol with this new security assumption including the update procedure. But how to prove the security of the updated protocol and the update procedure? One way might be to prove the protocol in question as long-term UC secure, a security framework which assumes that an adversary is unbounded after the computation of the protocol is completed and therefore computational assumptions do not hold once the protocol has finished its computation. Additionally, for long-term UC security are impossibility results shown, especially for commitment protocols. Therefore the notion of long-term UC security may be too strong if one wants to prove security against adversaries that increase their computational power during the computation since such an adversary is still computationally bounded, even after the computation of a protocol is completed.

In this thesis, we define a relaxed notion of long-term security, called $\mathcal{F}^{\mathbf{post}}$ security.

Additionally, we show a combination of commitment and coin toss protocols to toss a new CRS of an updated commitment scheme such that the commitment scheme is $\mathcal{F}^{\mathbf{post}}$ secure.

Deutsche Zusammenfassung.

Was passiert, falls eine kryptographische Annahme als nicht mehr sicher gilt und in welcher Weise betrifft dies die Sicherheit von kryptographischen Protkollen?

In dieser Hinsicht mag man sich überlegen, die Sicherheitsannahme zu aktualisieren und die Sicherheit des aktualisierten Protokolls inklusive der Aktualisierungsprozedur nachzuweisen. Wie jedoch lässt sich die Sicherheit des aktualisierten Protokolls und der Aktualisierungsprozedur nachweisen?

Eine Möglichkeit wäre zu beweisen, dass das gegebene Protokoll nachweisbar langfristig UC-sicher ist, ein Sicherheitsbegriff bei dem angenommen wird dass der Angreifer nach Protokollausführung unbeschränkt ist und daher nach Protokollausführung keine Komplexitätsannahmen gelten. Zudem wurden Unmöglichkeitsresultate gezeigt, insbesondere für Commitmentprotokolle. Daher kann der Begriff der langfristigen UC-Sicherheit etwas zu stark sein, wenn man die Sicherheit gegenüber Angreifern nachweisen möchte, die zwar während der Protokollausführung die Rechenkapazität erhöht, diese aber limitiert bleibt, auch nach der Ausführung des Protkolls.

In dieser Arbeit definieren wir einen gelockerten Begriff der langfristigen UC-Sicherheit, den wir $\mathcal{F}^{\mathbf{post}}$-Sicherheit nennen.

Darüber hinaus möchten wir zeigen, wie man ein $\mathcal{F}^{\mathbf{post}}$-sicheres Commitment-Schema verwenden kann, um einen Common Reference String (CRS) eines anderen Commitments zu aktualisieren.

Contents

1 Introduction

In today's world, computational power increases very fast. Algorithms that took days to compute a certain function some years ago are now able to compute the same functions only within hours or even less. For the security of cryptographic protocols this can lead to a problem: Some of the cryptographic assumptions like the length of a encryption scheme's private key turn out to be not sufficiently long enough, which makes the protocol itself insecure. Therefore one might ask: Is it possible to update a cryptographic assumption and how can be the security of such an updated protocol and the update process be proven?

A good start point for defining a security notion for updated protocols is the long-term UC security introduced by Müller-Quade and Unruh [MU10]. This notion of security combines two strong notions of security: long-term security which provides a definition of security which considers adversaries that may get unbounded after a protocol is completed and the notion of universal composability [Canoo] which is a strong tool for proving security of protocols in complex environments.

Yet, the long-term UC security may be a too strong notion if we only consider adversaries that increase their computational power during the computation of a protocol but remain computationally bounded after the computation is completed. Therefore some protocols can be secure against such adversaries but fail to be proven long-term UC secure.

In this work we want to find a way to relax the notion of long-term UC security such that protocols whose security assumptions are updated can be proven as secure against adversaries that increase their computational power during the computation. Thereby we will focus on the security of commitment schemes.

Furthermore, we want to show how to update a CRS of a commitment scheme using a process of commitments and coin tosses.

1.1 The New Framework

The updatability of a commitment is based on a new framework, called $\mathcal{F}^{\text{post}}$ which is a relaxation of long-term universal composability framework [MU10]. Before we shortly recap the long-term universal composability framework and explain our relaxed framework, we

want to shortly explain the universal composability framework on which long-term UC and (and also $\mathcal{F}^{\text{post}}$) is based on:

The universal composability framework defines security by comparing a real world in which a protocol is computed by two or more ITMs, called the parties, in a distributed manner and an ideal world where the same function is computed by a single machine, called the *functionality*. More precisely, in both worlds there is an environment which tries to distinguish between those two worlds. In the real world we have a protocol that is computed by two or more machines, called that parties, and an adversarial machine that may try to control one or more parties and potentially tries to instruct them to deviate from the protocol. In the ideal world, we have a single and incorruptible machine that computes the protocol's function and a machine called the *simulator* which tries to simulate the execution of the protocol by simulating the communication with the corrupted parties. By showing the indistinguishability of those two worlds, we are able to show that an adversary in the real world can not do more harm than in the ideal world where the function is computed in an incorruptible manner.

In the original universal composability framework the environment, the adversary and the simulator are defined as computationally bounded machines. This definition of security is not strong enough if we assume that an adversary may be computationally unbounded after the protocol execution: For example, the witness of a zero-knowledge proof can be learned by such an adversary. Therefore [MU10] have defined a stronger framework that is secure against this type of attackers: To achieve long-term UC security, the view of the environment (which is the environment's output after the execution protocol) in the real world and in the ideal world is *statistically* indistinguishable. This framework therefore defines long-term security for UC secure protocols. It assumes adversaries that may be potentially unbounded after the computation of the protocol and protocols that securely long-term UC realize a functionality $\mathcal{F}$ guarantee everlasting input privacy.

For a setup that assumes an adversary that remains computationally bounded after protocol execution but potentially increases its computational power during the computation, this definition of security seems to be too strong, especially protocols in the $\mathcal{F}_{\text{CRS}}$-Hybrid model cannot be proven long-term UC secure. The $\mathcal{F}^{\text{post}}$ framework therefore is a relaxation of the long-term UC framework: We define a protocol to $\mathcal{F}^{\text{post}}$-realize a given functionality if the environment's view in the ideal world is computationally indistinguishable from the environment's view in the real world iff the old assumption does not hold and the new asumption is believed to computationally hold.

1.2 Updating the Security Assumption of a CRS For a UC-Secure Commitment Scheme

The update procedure we present in this work is based on [BMM21]. Their result is a composable multi-party protocol based on timed commitments [BNoo]. As in our results they construct a coin toss protocol in the plain model for which the corruption of one party is simulatable. Using this protocol, they instantiate the CRS of a commitment scheme. Unlike in their work we do not use timed commitments which can be forcefully opened (with some effort) by the recipient using an algorithm that takes at most a certain time to open the commitment. Instead we use commitments generated by the commitment scheme that will be updated based on the weaker complexity assumption which are not hiding once the adversary increases its computational power.

Our main result is updating a CRS of a commitment scheme whose complexity assumption has to be updated. To update the CRS we will instantiate a coin toss protocol based on the new complexity assumption and sample a new CRS based on the new complexity assumption using the coin toss protocol. The update process of the CRS can roughly described as follows: In the initial step, the commitment scheme whose setup assumption has to be updated is used to create commitments. Those commitments are then used for several bit coin tosses. Those bit coin tosses are then used to instantiate the CRS of a single-use commitment scheme which is based on the new complexity assumption. The commitments generated with the stronger commitment scheme can then be used in a coin toss to sample the new CRS of the commitment scheme whose setup assumption is then updated. Yet, this instantiating process only suffices if at the beginning of the update process the amount of commitments generated with the updated commitment scheme is known. Therefore we additionally describe how the update process can be extended by using a multi-use commitment. By using a multi-use commitment, one can generate several commitments using the same instance of the multi-use commitment scheme and therefore it is possible to instantiate several CRS of the updated commitment scheme in a flexible way.

For the update process, we make some requirements on the protocols that are involved: First of all, we want to assure that at the initial state of the update process the commitments generated by the commitment scheme that is still based on the setup assumption to be secure against the adversary. Therefore we require the initial commitment scheme (whose setup assumption will be updated) to be long-term UC secure. This means that only statistically binding commitment schemes or statistically hiding commitment schemes can be updated. Since the commitment scheme we update has a CRS, this scheme cannot be proven to be long-term UC secure (against $\mathcal{F}_{\mathbf{com}}$) since the CRS generated in the CRS can only be computationally

indistinguishable from the CRS generated in the real world. To ensure long-term security of the commitment scheme in question we will define two relaxed variants of $\mathcal{F}_{\text{com}}$ such that unconditionally binding UC secure commitment schemes and unconditionally hiding UC secure commitment can be proven long-term UC secure compared to the new functionalities we define. Those unconditionally binding or unconditionally hiding commitment schemes can be candidates for being updated. For the update procedure of the CRS we will instantiate a coin toss. When building the coin toss protocol, we will see that a $\mathcal{F}^{\text{post}}$-secure coin toss using both of the weakened commitment functionalities is impossible. A short explanation explanation for this is that the new functionalities we define in this work are actually too weak to be used in a UC (and hence $\mathcal{F}^{\text{post}}$) secure composition. Yet, there is a way to fix the coin toss such that this coin toss protocol is at least stand-alone secure. Therefore this coin toss protocol can only be used in a non-modular way. For creation of the two commitment schemes based on the stronger setup assumption we modify two UC secure commitment schemes: one single- commitment scheme whose CRS can be split in two parts and is uniformly random to ensure that the commitment scheme can be used with the modified functionality and one multi-use commitment scheme whose CRS can also be uniformly random.

1.3 Overview

In Chapter 2, we will shortly recap the definitions needed for our work, including the UC and long-term UC framework.

In Chapter 3 we will define the new $\mathcal{F}^{\text{post}}$ framework and define commitment functionalities that are used for showing long-term UC security of updatable commitment schemes and show two candidates that long-term UC securely realize those functionalities and therefore can assumed to be updated.

The impossibility of a $\mathcal{F}^{\text{post}}$ secure toss using our newly defined functionalities is shown in Chapter 4 as well as a way to redefine one of the functionalities. Furthermore, we give one example of a $\mathcal{F}^{\text{post}}$-secure commitment protocol that is usable in a $\mathcal{F}^{\text{post}}$ secure coin toss protocol in order to sample the new CRS.

In Chapter 5 we show how a multi-use commitment has to be modified to be able to toss several CRS for the updated commitment protocol when the multi-use commitment is composed with the commitment protocol defined in Chapter 4.

We conclude this work in 6 by recapping the results of this work.

2 Preliminaries

In this chapter, we define the notions of tools we will need throughout this work. Many of the standard definitions use uniform probabilistic polynomial-time algorithms. Since our work is based on the Universal composability framework whose definition uses *non-uniform* probabilistic polynomial-time algorithms, we will cite many of the security definitions with a modification.

If we cite a definition verbatim or without a modification from another work, then we will give a keyword "see" or give the citation in brackets. If we modify a definition according to our needs, then we will use the keyword "compare" or "cmp." followed by the citation we modified.

First, we will define the notations we will use throughout this work in 2.1.

In 2.2 we will recap the definitions of trapdoor permutations, enhanced trapdoor permutations and trapdoor permutations with dense public keys.

In 2.3 we will define commitment schemes, followed by recapping coin tosses in 2.4.

In the last two sections, we will shortly explain the two security frameworks on which our new framework is based on: The universal composability framework is described in 2.5 and a short recap on long-term universal composability can be seen in 2.6.

2.1 Notations

Throughout this work we will let $\oplus$ denote as the bitwise XOR operator.

We call a function f *negligible* if for every positive polynomial p and all sufficiently large $n \in \mathbb{N}$ we have $f(n) \leq \frac{1}{p(n)}$.

We now define probability distribution ensembles and indistinguishability as defined in [Canoo], Section 4.2 and [Lin17], Section 2. We call $\{X(n,z)_{n\in, z\in\{0,1\}^*}\}$ a probability distribution ensemble as an infinite set of probability distributions. Each distribution $X(n,z)$ is associated with a $n \in \mathbb{N}$ and $z \in \{0,1\}^*$.

Now we want to define two notions of indistinguishability. Therefore we will cite the definition of computational indistinguishability in [Lin17], Section 2 and [Canoo], Definition 4 for a formal definition of statistical indistinguishability:

Definition 2.1 (Computational indistinguishability, [Lin17], Section 2) *Let* $X = \{X(n,z)_{n\in\mathbb{N}, z\in\{0,1\}^*}\}$ *and* $Y = \{Y(n,z)_{n\in\mathbb{N}, z\in\{0,1\}^*}\}$ *be two probability distribution ensembles. We call* X *and* Y computationally indistinguishable *if for any non-uniform polynomial-time algorithm* A *there exists a negligible function* $\mu(\cdot)$ *and an* $n_0 \in \mathbb{N}$ *such that for any security parameter* $n < n_0$ *and any* $z \in \{0,1\}^*$ *we have*

$$Pr\Big|\big[A(X(n,z)) = 1\big] - Pr\big[A(Y(n,z)) = 1\big]\Big| < \mu(n)$$

Definition 2.2 (Statistical indistinguishability, [Canoo], Definition 4) *Let* $X = \{X(n,z)_{n\in\mathbb{N}, z\in\{0,1\}^*}\}$ *and* $Y = \{Y(n,z)_{n\in\mathbb{N}, z\in\{0,1\}^*}\}$ *be two probability distribution ensembles. We call* X *and* Y statistically indistinguishable *if there exists a negligible function* $\mu(\cdot)$ *and an* $n_0 \in \mathbb{N}$ *such that for any security parameter* $n < n_0$ *and any* $z \in \{0,1\}^*$ *we have*

$$Pr\Big|\big[X(n,z) = 1\big] - Pr\big[Y(n,z) = 1\big]\Big| < \mu(n)$$

2.2 Trapdoor Permutations

This section is based on [Golo1], Section 2.4.4.1, [Golo4], Section C.1 and [DP92]. Section 3.

Since trapdoor permutations will be used in many of our protocols, we will shortly recap its definition.

Informally a trapdoor permutation is a function whose domain is the same as its image. Additionally, there is a public key *pk* (in the following definition denoted as index) together with a private key *sk* for this function f such that, given the public key *pk* the evaluation $f(x)$ can be computed within polynomial time for each value x out of f's domain. Yet, this function is hard to invert (i. e. given a random value y from the image of f, the probability to find an x such that $y = f(x)$ within polynomial time is negligible) unless one has the private key *sk*. This makes *sk* a *trapdoor* for the permutation f. More specifically, given *sk* one can compute for every given y from the image of f the inverse of f (i. e x such that $x = f^{-1}(y)$) within polynomial time.

For a formal definition, we cite [Golo1], Definition 2.4.5 (in this definition we omit the standard version of the inversion hardness property since we will focus in this work on non-uniform adversaries):

Definition 2.3 (Trapdoor permutation ([Golo1], Definition 2.4.5)) *Let* $\bar{I} \subseteq \{0,1\}*$ *and* $\bar{I}_n \overset{def}{=} \bar{I} \cap \{0,1\}^n$. *A* collection of permutations *with* indices *in* $\bar{I}$ *is a set* $\{f_i : D_i \to D_i\}_{i\in\bar{I}}$ *such that* f_i *is* $1-1$ *on the corresponding* D_i. *Such a collection is called a trapdoor permutation if there exist four probabilistic polynomial-time algorithms I, D, F and* F^{-1} *such that the following five conditions hold:*

1. Index and trapdoor selection*: for every n,*

$$Pr[I(1^n) \in \bar{I}_n \times \{0,1\}^*] > 1 - 2^{-n}$$

2. Selection in domain*: For every* $n \in \mathbb{N}$ *and* $i \in \bar{I}_n$

   a) $Pr[D(i) \in D_i] > 1 - 2^{-n}$

   b) *Conditioned on* $D(i)$*, the output is uniformly distributed in* D_i*. That is, for every* $x \in D_i$*,*

$$Pr[D(i) = x | D(i) \in D_i] = \frac{1}{|D_i|}$$

   Thus, $D_i \subseteq \cup_{m \leq poly(|i|)}\{0,1\}^m$*. Without loss of generality,* $D_i \in \{0,1\}^{poly(|i|)}$*.*

3. Efficient evaluation*: For every* $n \in \mathbb{N}$*,* $i \in \bar{I}_n$ *and* $x \in D_i$*,*

$$Pr[F(i,x) = f_i(x)] > 1 - 2^{-n}$$

4. Hard to invert*: Let* I_n *be a random variable describing the distribution of the first element in the output of* $I(1^n)$ *and* $X_n \stackrel{def}{=} D(I_n)$*. For every family of polynomial-sized circuits* $\{C_n\}_{n\in\mathbb{N}}$*, every positive polynomial* $p(\cdot)$*, and all sufficiently large n's*

$$Pr[C_n(I_n, f_{I_n}(X_n)) = X_n] < \frac{1}{p(n)}$$

5. Inverting with trapdoor*: For every* $n \in \mathbb{N}$*, any pair* (i,t) *in the range of* $I(1^n)$ *such that* $i \in I_n$*, and every* $x \in D_i$*,*

$$Pr[F^{-1}(t, f_i(x)) = x] < 1 - 2^{-n}$$

An *enhanced trapdoor permutation* is a trapdoor permutation which has some "enhancement". This enhancement is the following: given the index and the randomness used for sampling the domain of f_i, it is infeasible to invert the function f_i on a random value $x \leftarrow D_i$ drawn from the domain of f_i.

In the definition of enhanced trapdoor permutations, there will be used a varied definition of the sampling algorithm D: an algorithm D' that additionally gets the random coins from a distribution R_n as an auxiliary input.

More formally, we define enhanced trapdoor permutations as in [Golo4], Definition C.1.1:

Definition 2.4 (Enhanced trapdoor permutations ([Golo4], Definition C.1.1)) *Let* $\{f_i : D_i \rightarrow D_i\}$ *be a collection of trapdoor permutations. We say that this collection is* enhanced (and call it an enhanced collection of trapdoor permutations) *if for every non-uniform probabilistic polynomial-time algorithm A, every positive polynomial p and all sufficiently large n's*

$$Pr[A(I_1(1^n), R_n) = f_{I_1(1^n)}{}^{-1}(D'(I_1(1^n), R_n))] < \frac{1}{p(n)},$$

where D' is the residual two-input algorithm obtained from D when treating the coins of D as an auxiliary input and R_n denotes the distribution of the coins of D on n-bit long inputs.

The last variant of a trapdoor functions we will use in this work, we have trapdoor functions with dense public keys. Such trapdoor functions have public keys which are uniformly distributed. This concludes that if a random element is drawn from $\{0,1\}^k$ where k is the length of public keys for f, then with non-negligible this public key is a secure public key and has a corresponding private key.

For the definition of a trapdoor permutation dense public keys, we recap the informal definition of De Santis et al. [DP92] in a modified way for trapdoor permutations. We modify the definition minimally since the original definition was defined for encryption schemes.

Definition 2.5 (Trapdoor permutation with dense public keys, compare [DP92], Section 3) *Let k be the length of the public keys used for a trapdoor permutations and $d \in \mathbb{N}$.*

A trapdoor permutation with dense public keys is a trapdoor permutation such that

- *Each k-bit string defines a public key of the trapdoor permutation*
- *If an k-bit string is picked at random, then there is a non-negligible probability of at least k^{-d} that a secure public key is picked*
- *It is possible to generate uniformly distributed k-bit public keys along with their corresponding secret keys.*

Remark 2.6 *Please note that the definition of trapdoor permutations with dense public keys is not strong enough for our purposes: need the probability to pick secure public key randomly with overwhelmingly probability, not with a probability of k^{-d}. One solution for this problem may be combining several trapdoor permutations with dense public keys to ensure that the probability of generating a secure public key is high enough.*

2.3 Commitment Schemes

Since we will focus on commitment schemes in this thesis, we will shortly recap the definition of commitment schemes, compare [Golo1], Section 4.4.1:

A commitment scheme is a two-party protocol in which a party, called the committer C, can commit itself on a value whereas the recipient R only learns the committed value when the committed value is revealed. More precisely, it is a two-phase protocol consisting of a

committing phase (in which the committer can commit itself to a value) and a unveil phase in which the recipient learns the committed value.

For a formal definition, we will cite [BHY09], Definition 3 in a modified way: in the original definition the machine A (the adversarial machine) is defined as a uniform machine. Since we will use commitment schemes in the UC framework where the adversary is defined as a *non-uniform* machine, we will accommodate the definition of A.

We now formally cite a [BHY09], Definition 3 in a modified way for a formal definition of a commitment scheme:

Definition 2.7 (Commitment scheme, (compare [BHY09], Definition 3)) *For a pair of PPT machines $Com = (C, R)$ and a non-uniform machine A, consider the following experiments:*

Experiment $\mathbf{Exp}^{binding}_{Com,A}(n)$	**Experiment** $\mathbf{Exp}^{hiding-b}_{Com,A}(n)$
run $\langle R(\mathsf{recv}), A(\mathsf{com})\rangle$ $m'_0 \leftarrow \langle R(\mathsf{open}), A(\mathsf{open},0)\rangle$ rewind A and R back to after step 1 $m'_1 \leftarrow \langle R(\mathsf{open}),A(\mathsf{open},1)\rangle$ return 1 iff $\bot \neq m'_0 \neq m'_1 \neq \bot$	$(m_0, m_1) \leftarrow A(\mathsf{choose})$ return $\langle A(\mathsf{recv}), C(\mathsf{com},m_b)\rangle$

In this, $\langle A,C\rangle$ denotes the output of A after interacting with C and $\langle R,A\rangle$ denotes the output of R after interacting with A. We say that Com *is a* commitment scheme *iff the following holds:*

Syntax *For any $m \in \{0,1\}^n$, $C(\mathsf{com},m)$ first interacts with $R(\mathsf{recv})$. We call this the* commit phase. *After that, $C(\mathsf{open})$ interacts again with $R(\mathsf{open})$, and R finally outputs a value $m' \in \{0,1\}^n \cup \{\bot\}$. We call this the opening phase.*

Correctness *We have $m' = m$ always for all m.*

Hiding *For a non-uniform PPT machine A, let*

$$\mathbf{Adv}^{hiding}_{com,A}(n) := Pr[\mathbf{Exp}^{hiding-0}_{Com,A}](n) - Pr[\mathbf{Exp}^{hiding-1}_{Com,A}](n),$$

where $\mathbf{Exp}^{hiding-b}_{Com,A}$ is depicted above. For Com to be hiding, we demand that $\mathbf{Adv}^{hiding}_{com,A}(n)$ is negligible for all non-uniform PPT A that satisfy $m_0, m_1 \in \{0,1\}^n$ always.

Binding *For a machine A, consider the experiment $\mathbf{Adv}^{binding}_{com,A}(n)$ above. For Com to be binding, we require that $\mathbf{Adv}^{binding}_{com,A}(n) = Pr[\mathbf{Exp}^{binding}_{Com,A}(n) = 1]$ is negligible for all non-uniform PPT A.*

Further, we say that Com is perfectly binding iff $\mathbf{Adv}_{com,A}^{binding}(n) = 0$ *for all non-uniform (not necessarily PPT) A. We say that Com is statistically hiding iff* $\mathbf{Adv}_{com,A}^{hiding}(n)$ *is negligible for all non-uniform (not necessarily PPT) A.*

2.4 Coin Toss

In this Section, we will shortly recap the notion of a cryptographically secure coin toss.

The first cryptographically secure coin toss protocol was introduced by Manuel Blum in [Blu81]. The idea for this coin toss protocol was to securely flip a coin even when one of the two participants acts maliciously. The protocol is a simple three-step protocol and uses a commitment protocol.

A coin toss protocol is used to sample a random bit or a random string in a distributed manner.

This protocol which can be seen in Figure 2.1 is still nowadays the standard protocol for tossing a coin or a random string, yet the commitment protocol for executing the coin toss protocol is chosen individually according to the security requirements of the protocol.

Protocol $\Pi_{\text{Blum-CT}}$

- P_1 draws a random bit $b_1 \leftarrow \{0,1\}$, generates a commitment c sends c to P_2.
- When getting a commitment c, P_2 sends bit $b_2 \leftarrow \{0,1\}$ to P_1.
- P_1 then unveils c to b_1.
- When getting the unveil information to c, both P_1 and P_2 compute $b = b_1 \oplus b_2$.

Figure 2.1: The protocol $\mathbf{\Pi}_{\mathbf{Blum-CT}-bit}$ for tossing a coin.

Assumed that the commitment scheme is secure, the coin toss protocol outputs a uniformly random bit.

2.5 Universal Composability

In this section we will recap the universal composability framework which is a powerful tool for defining cryptographic protocols and proving their security. The whole section is based on [Canoo], Sections 4 and 5 as well as [MU10], Section 3.1.

First, we will give an overview of the UC framework. Then we will describe the real-world model and the ideal-world model is described at last.

2.5.1 Overview

The Universal Composability framework [Canoo] defined by Ran Canetti is a very powerful framework: As the name already suggests, a protocol sufficing the UC security can be composed in any way with another protocol sufficing the UC security without losing its security, even when concurrently composed.

The security is proven by comparing the real execution of the protocol in presence of an adversary with an ideal and non-corruptible computation of the protocol's function in presence of a simulator that simulates the behavior of the real-world adversary. This definition looks quite familiar compared to the stand-alone model where a protocol can be composed with other protocols as long as the composition is done sequentially, i. e. it is guaranteed that only one instance of the protocol is run at a time. The difference to the stand-alone security is formulated by an environment that is able to interact during the execution of the protocol with the adversary (whereas in the stand-alone model the adversary gets its instructions at the beginning of the protocol and at the end of the computation the adversary sends only the transcript to the environment). This makes the simulator in the UC setting impossible to rewind a party or the internal simulation of an adversary since the environment can notice this behavior and therefore easily distinguish the real-world execution from the ideal-world simulation.

Real Execution

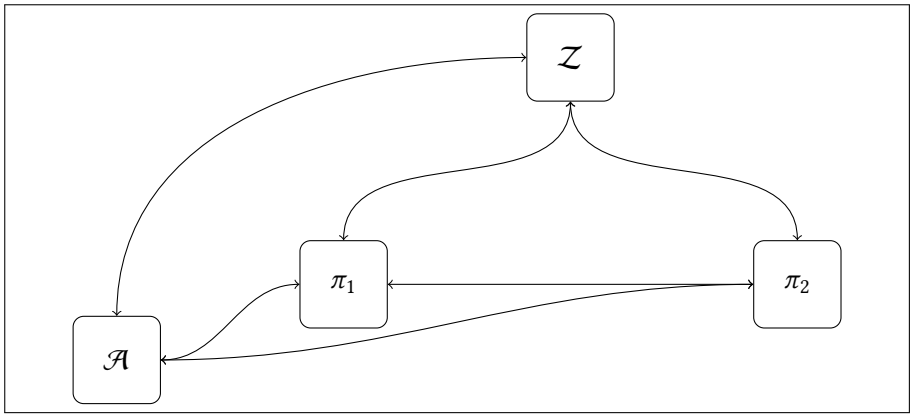

Figure 2.2: The real world with environment $\mathcal{Z}$, functionality $\mathcal{F}$, adversary $\mathcal{S}$ and parties π_1 and π_2 computing protocol Π, see [Canoo], Figure 2.

We define the real execution of a protocol as the distributed computation of a specific

function by several machines (called the parties) in presence of an adversarial machine, called the adversary $\mathcal{A}$. The adversary may behave in several ways: It may read the messages sent by and delivered to the corrupted parties (such an adversary is called a *semi-honest* or *passive* adversary). It additionally run arbitrary code on the corrupted parties (this type of adversary is called a *malicious* or *active* adversary). First of all, we define three types of ITMs which are acting in the real execution: the environment, the adversary and the computing parties which are executing the protocol.

The environment machine is a non-uniform ITM that represents all possible machines that interact with the parties during the computation of the protocol. It additionally provides all parties with input and receives their outputs.

Then there is the adversary: it is a non-uniform ITM that is able to corrupt any party and is able to read the messages of the parties it corrupts (if the adversary instructs each corrupted party to follow the instructions of the protocol, then the adversary is passive) and maybe additionally tries to bias the outcome of the computation (then the adversary is active).

The last type of machines we have in the real world are the computing parties: These are ITMs which compute together a certain function, where a subset of the parties follows the instructions of the adversary instead of the instructions of the protocol. We call this subset of parties *corrupted parties*.

The adversary may corrupt parties only at the beginning of the execution of the protocol and then is not allowed to change the set of corrupted parties (i. e. corrupt additional parties) during the computation. In this case the adversary is called a *static* adversary. In some models the adversary may corrupt additional parties during the execution. This type of adversary is called an *adaptive* adversary. In our work we will only focus on static adversaries.

Ideal Execution

In the ideal world we do not have parties that execute the protocol but a central machine that computes securely the function the protocol intends to compute: the ideal functionality $\mathcal{F}$.

We also have the environment machine that represents protocols the functionality interacts with. It further sends inputs to the functionality and receives output from the functionality.

As in the real world, there exists an ideal-world adversary, called the simulator $\mathcal{S}$. Since in the ideal world are no computing parties, the ideal-world adversary only sends to and receives messages from the functionality and the environment.

To show that a certain real-world protocol Π securely realizes a given functionality $\mathcal{F}$, we show that there exists an ideal protocol Φ (which is computed by the functionality $\mathcal{F}$ and has the same amount of parties which only directly deliver messages coming from the environment

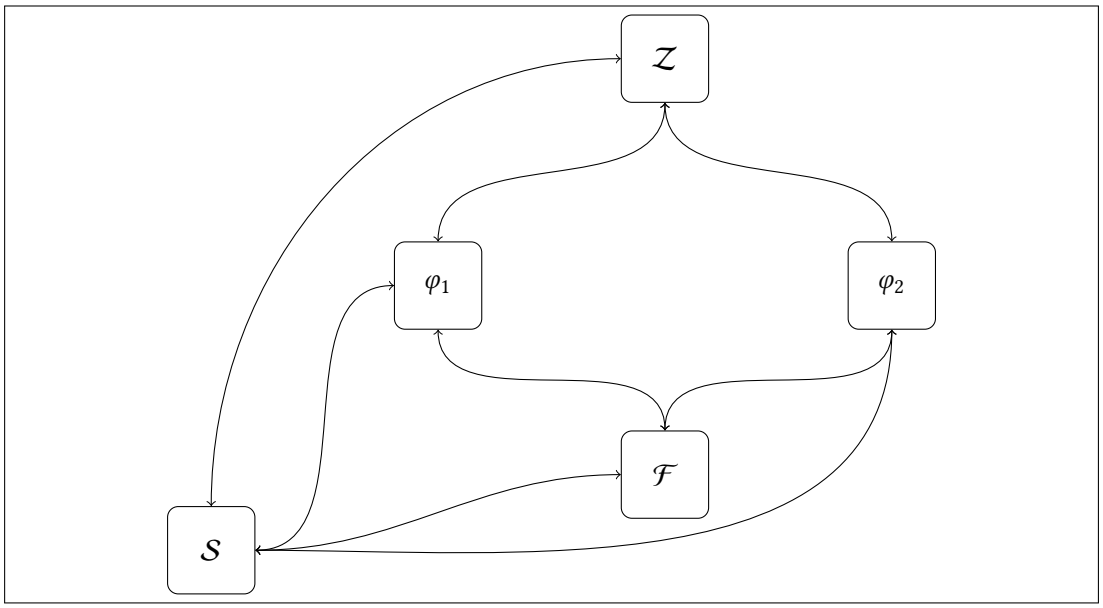

Figure 2.3: The ideal world with environment $\mathcal{Z}$, functionality $\mathcal{F}$, simulator $\mathcal{S}$ and dummy parties φ_1 and φ_2, see [Canoo], Figure 3.

to $\mathcal{F}$ and vice versa, called the *dummy parties*) and show that for any adversary there exists a simulator such that no environment can distinguish between the real-world protocol Π and the ideal-world protocol Φ.

More formally, we define $\{EXEC_{\Pi,\mathcal{A},\mathcal{Z}}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^{poly(n)}}$ to be the distribution of $\mathcal{Z}$'s outputs in the real world when interacting with the parties computing Π and the real-world adversary $\mathcal{A}$ with security parameter $n \in \mathbb{N}$ on input $z \in \{0,1\}^{poly(n)}$ and analogously define $\{EXEC_{\Phi,\mathcal{S},\mathcal{Z}}(n,z)\}_{\in\mathbb{N},z\in\{0,1\}^{poly(n)}}$ to be the distribution of $\mathcal{Z}$'s output in the real-world when interacting with the dummy parties and the ideal-world adversary $\mathcal{S}$ with security parameter $n \in \mathbb{N}$ on input $z \in \{0,1\}^{poly(n)}$. For better readability, we will abbreviate $\{EXEC_{\Phi,\mathcal{S},\mathcal{Z}}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^{poly(n)}}$ as $\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^{poly(n)}}$. This makes the distribution easier to recognize the functionality.

For a formal definition we cite Definition 3.1 in [MU10]:

Definition 2.8 (UC-realization ([MU10], Definition 3.1)) *A protocol Π UC realizes a protocol Φ, if for any polynomial-time adversary $\mathcal{A}$ there exists a polynomial-time simulator $\mathcal{S}$ such that for any polynomial-time environment $\mathcal{Z}$ the families of random variables $\{EXEC_{\Pi,\mathcal{A},\mathcal{Z}}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^{poly(n)}}$ and $\{EXEC_{\varphi,\mathcal{S},\mathcal{Z}}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^{poly(n)}}$ are computationally indistinguishable.*

Remark 2.9 *As stated as in [MU10], Section 3.4, we will assume throughout this work that the adversary is informed of every invocation of the ideal functionality $\mathcal{F}$ and the functionality only delivers outputs if the adversary allows this delivery. We will shortly formulate the phrase "Upon input a from party P_1 [...] generate a publicly delayed output y to P_2", meaning the phrase "Upon input a from party P_1, deliver message* (`i-th input from` P_1) *to $\mathcal{A}$. Upon message* (`deliver` i), *$\mathcal{F}$ outputs y to P_2."*

Dummy Adversaries

Throughout this work we will work with a so-called "dummy adversary", which does nothing except passing messages it gets from $\mathcal{Z}$ directly to the corrupted parties and vice versa. This enables us to create simulators that do not depend on the adversary's behavior. This is different from the original definition of UC-realization since according to the definition, the simulator has to depend on the adversary (since we define the simulator's behavior by the adversary's behavior). Even if the simulator is quantified over this specific dummy adversary, the notion of security remains the same. The idea is the following: by assuming the dummy adversary, we assume the "hardest adversary to simulate" since the environment is able to simulate every adversary and therefore is able to control all communication (see [Canoo], Section 4.4.1).

For a formal definition of UC emulation with the dummy adversary, we cite [Canoo], Claim 10:

Theorem 2.10 (UC-emulation with dummy adversaries ([Canoo], Claim 10)) *Let Π and Φ be protocols. Then Π UC-emulates Φ if and only if it UC-emulates Φ with respect to the dummy adversary.*

Well-Formed Functionalities

Since throughout this work we will work in many cases with well-formed functionalities, we want to informally describe a well-formed functionality, which is informally explained in [Can+02], Section 3.3: A well-formed functionality is a functionality that consists of two levels of procedures: the core procedure which computes the function and the shell procedure which forwards all communication coming from the parties to the core procedure except the information which of the parties is corrupted. This guarantees that the functionality's function does not depend on the information which party is corrupted.

2.5.2 Composability

Now we will look at one main tool of the UC framework: the composability.

Generally, all protocols that are proven UC secure are simultaneously proven as concurrently secure, meaning that even several instances of the same protocol run concurrently, the security of the protocols still remains. The security remains even if the protocol is run as a subroutine of another protocol, which is the composability of a protocol.

Hybrid Model

In a hybrid model, we have the real-world model consisting of the environment, the adversary and the parties computing a protocol Π. Additionally to these we have a functionality $\mathcal{F}$ which the parties have access to. More specifically, the parties have access to an unbounded number of copies of $\mathcal{F}$, called *instances of* $\mathcal{F}$. We then say Π is a protocol in the $\mathcal{F}$-Hybrid model.

Composition Theorem

If we replace in the $\mathcal{F}$-hybrid model the functionality $\mathcal{F}$ by a protocol Π that UC-realizes $\mathcal{F}$, we can show that this composition of protocols is secure as the protocol σ in the $\mathcal{F}$-hybrid model. An intuitive idea for this is to view the protocol σ as the *environment of* Π. Therefore we view Π as the real-world protocol and view $\mathcal{F}$ as the functionality in the ideal world. Since Π UC-realizes $\mathcal{F}$, no environment can distinguish between the real-world protocol and the ideal-world and security follows.

Informally the composition theorem states that if a protocol σ is defined in an Π-hybrid model (i. e. it uses the protocol Φ) and the protocol Φ is replaced by a protocol Π that UC-securely realizes Φ, then the protocol σ^{Π} (that uses Π instead of Φ) remains secure.

For a more formal statement, we cite the composition theorem as stated in [MU10], Theorem 3.2:

Theorem 2.11 (Universal composition theorem, ([MU10], Theorem 3.2)) *Let Φ, Π and σ be polynomial-time protocols. Assume that Π UC realizes Φ. Then protocol σ^{Π} UC-realizes σ^{Φ}.*

As we will see later on, this theorem will be used several times.

2.6 Long-Term Universal Composability

In this section which is based on [MU10], Section 1.1 and 3.2 we will describe the long-term universal composability framework as defined in [MU10].

A problem that arises with the UC framework is that the security of the protocols is not guaranteed against adversaries that increase their computational power after the execution of a protocol, being able to break the protocol after its execution (very simple examples

are learning the witness of a zero-knowledge proof or being able to extract the value of a commitment that was not unveiled). More precisely, a possible violation of the input privacy of a protocol after the completion of the protocol results in a statistical distinguishability between the environment's output in the simulation and the environment's output in the real world, which is not captured by the UC framework (in the UC framework only the *computational* indistinguishability between the environment's output in the ideal world and the environment's output in the real world is considered).

A question that arises in this context is: Can there be a UC-like framework that defines security even against such adversaries?

This question was solved by Müller-Quade and Unruh in [MU10]. They defined the *Long-term Universal Composability* which is based on the Universal Composability framework and also defines long-term security of UC secure protocols.

2.6.1 Long-Term Security

Informally, a protocol is long-term secure if no adversary is able to break its security after the computation on the protocol was completed, i. e. the adversary is not able to get the private inputs and private random coins used for the protocol of the uncorrupted parties even if the adversary gets unlimited amount of time after the computation of the protocol was completed.

A simple intuition is letting the environment and adversary be unbounded after the computation of the protocol is completed. Therefore any adversary is able to gain computational power after the computation is completed. Since the adversary does not communicate with the environment (and therefore the simulator does neither communicate with the environment) and in the UC model only the environment computes after the execution of the protocol after computation of the protocol is completed, the definition for long-term UC-security is a bit different than the intuition.

We now formally define the long-term UC security by citing Definition 3.3 in [MU10]:

Definition 2.12 (Long-term UC security, ([MU10], Definition 3.3)) *Let Π be a two-party or multi-party protocol and Φ be an ideal one-party protocol. We say that Π securely long-term UC realizes Φ if for any computationally bounded adversary $\mathcal{A}$ there exists a computationally bounded simulator $\mathcal{S}$ such that for any computationally bounded environment $\mathcal{Z}$ the family of random variables $\{EXEC_{\Pi,\mathcal{A},\mathcal{Z}}(n,z)_{z\in\{0,1\}^{poly(n)}}\}$ (the real-world execution with security parameter n and output z by $\mathcal{Z}$) and $\{EXEC_{\Phi,\mathcal{S},\mathcal{Z}}(n,z)_{z\in\{0,1\}^{poly(n)}}\}$ (the ideal world execution with security parameter n and output z by $\mathcal{Z}$) are statistically indistinguishable.*

Remark 2.13 *The output $\{EXEC_{\Pi,\mathcal{A},\mathcal{Z}}(n,z)_{z\in\{0,1\}^{poly(n)}}\}$ of the environment is defined differently than in the definition of standard-UC emulation: In the standard UC-emulation, the*

environment's output is generally defined as a single bit, whereas in the definition of long-term UC-emulation, the environment is allowed to output arbitrary strings. Especially, the environment is allowed to output its complete view.

This means for the long-term security of a UC-secure protocol that the computational power of adversary, environment and simulator remains the same as in the standard UC framework but the output of the environment (which is the environment's view) in the ideal world is *statistically* indistinguishable from the environment's output in the real world. This is obviously a stronger notion compared to standard UC security. Also, it is easy to see that long-term UC security implies standard UC security.

Since the modification of the security definition of the UC framework towards long-term UC is minimal, the composition theorem also holds for the long-term UC framework.

More formally, the Composition Theorem for long-term UC security ([MU10], Theorem 3.4) is stated as follows:

Theorem 2.14 (Universal composition theorem, see [MU10], Theorem 3.4) *Let Π, Φ and σ be polynomial-time protocols. Assume that Π long-term UC-realizes Φ. Then ρ^{Π} UC realizes σ^{Φ}*

We say a protocol is long-term revealing for some party P if its view can be entirely computed from the communication of all other parties with P without having P's view itself.

The intuition behind the definition of a long-term revealing functionality that if a functionality $\mathcal{F}$ is long-term revealing towards a party P then any secrets that P and $\mathcal{F}$ share may eventually become public.

Before we formally cite the definition of long-term revealing functionalities, we want to explain the notion of a network within the long-term UC framework: The real-world model and the ideal-world model can both be represented as a graph with the machines ($\mathcal{Z}$, $\mathcal{A}$ and the parties in the real world and $\mathcal{Z}$, $\mathcal{F}$ and the dummy parties in the ideal world) depicted as vertices and the direct communication between two machines represented as edges. We denote a network S as such a communication graph as explained above.

Definition 2.15 (Long-term revealing functionality, [MU10], Definition 4.1) *Let P be a party identifier. For a given network S, let $trans_S$ denote the transcript of all communication between a functionality $\mathcal{F}$ and all other machines (including the adversary) in an execution of $S \cup \mathcal{F}$. Let $trans_S \backslash P$ denote the transcript of all communication between $\mathcal{F}$ and all machines except P.*

We say a functionality $\mathcal{F}$ is long-term revealing (LTR) for party P if the following holds for any network S: There is a deterministic function f_S (not necessarily efficiently computable) such that with overwhelming probability we have $trans_S = f_S(n, trans_S \backslash P)$

2.6.2 Restrictions on Commitment Schemes

This Subsection is mainly based on [MU10], Section 4.1.

There are some restrictions on the long-term security of UC-secure commitment schemes. As a first result, we have that commitment schemes in general are long-term revealing for the recipients. This means that any secrets that are shared among the recipient and the functionality $\mathcal{F}_{\mathbf{com}}$ (i. e. any inputs that $\mathcal{F}_{\mathbf{com}}$ receives from R and any outputs that R receives from $\mathcal{F}_{\mathbf{com}}$) will become eventually public.

Therefore we cite [MU10], Lemma 4.2:

Lemma 2.16 (See [MU10], Lemma 4.2) *Coin toss ($\mathcal{F}_{\mathbf{ct}}$) and CRS ($\mathcal{F}_{\mathrm{CRS}}$) are long-term revealing for all parties. Commitment ($\mathcal{F}_{\mathbf{com}}$) and ZK ($\mathcal{F}_{\mathbf{zk}}$) are long-term revealing for the recipient/verifier. If G is a key generation algorithm such that the secret key depends deterministically on on the public key then PKI $\mathcal{F}_{\mathbf{PKI}}$ is long-term revealing for all parties.*

Since unconditionally binding commitment schemes are also long-term revealing for the committer, unconditionally binding commitments cannot be long-term secure.

Also commitment protocols which are based on an hybrid model whose functionality is long-term revealing to the committer, such as the functionality $\mathcal{F}_{\mathbf{crs}}$ are neither long-term secure. Since many UC-secure commitment protocols are in the $\mathcal{F}_{\mathbf{crs}}$-hybrid model, those cannot be long-term UC-secure.

For a formal statement, we cite [MU10], Theorem 4.3:

Theorem 2.17 (Impossibility of commitment with LTR functionalities ([MU10], Theorem 4.3)) *Let $\mathcal{F}$ be a functionality that is long-term revealing for party C. Then there is no non-trivial polynomial-time protocol that long-term UC-realizes commitment with sender C ($\mathcal{F}_{\mathbf{com}}$) in the $\mathcal{F}$-hybrid model.*

To be able to UC-realize a commitment we have to use a setup assumption as Canetti and Fischlin have shown in [CF01], Theorem 6 that it is impossible to UC-realize $\mathcal{F}_{\mathbf{com}}$ in the plain model (i. e. without using a setup assumption). This also holds for the realization of long-term UC secure commitment schemes.

As a direct consequence, a commitment cannot be turned around. This means the following: If we have a long-term UC secure commitment from a party A to B then we are not able to construct a long-term UC secure commitment from B to A.

For a formal statement we cite [MU10], Corollary 4.4:

Corollary 2.18 (Commitments cannot be turned around ([MU10], Corollary 4.4))
There is no non-trivial, polynomial-time protocol long-term UC realizing $\mathcal{F}_{\mathrm{com}}$ with committer A and recipient B using any number of instances of $\mathcal{F}_{\mathrm{com}}$ with committer B and recipient A.

3 Defining the Functionality For Updatable Commitments

In this chapter, we want to define the the security notion of updated protocols, called $\mathcal{F}^{\text{post}}$ security and we additionally want to define the functionalities for commitments whose CRS can be updated.

Since we consider that the adversary increases its computational power during the computation but stays computationally bounded, we cannot rely on the UC security since in this framework the computational power of the adversary is not allowed to increase during the computation of the protocol. Therefore we will define the $\mathcal{F}^{\text{post}}$ framework which is a relaxation of the long-term UC framework [MU10].

Analogously to the notion of long-term security, we allow the output of the environment to be an arbitrary output. Especially we enable the environment to output its entire view.

Instead of requiring the view of the environment in the ideal world to be statistically indistinguishable from the environment's view in the real world, we require the view of the environment in the ideal world to be computationally indistinguishable from its view in the real world iff one problem $\mathcal{L}$ is solvable (this problem resembles the old complexity assumption that might eventually not hold anymore) and one problem $\mathcal{M}$ is not solvable within polynomial time (which resembles the updated complexity assumption).

For a formal definition of $\mathcal{F}^{\text{post}}$security, we modify the definition of long-term security in a relaxed way:

Definition 3.1 ($\mathcal{F}^{\text{post}}$, compare [MU10], Definition 3.3) *Let $\mathcal{L}$ and $\mathcal{P}$ be two problems.*

A protocol π $\mathcal{F}^{\text{post}}$ securely realizes an ideal protocol ρ if for any polynomial-time adversary $\mathcal{A}$ there exists an ideal-world adversary $\mathcal{S}$ such that for any polynomial-time environment $\mathcal{Z}$, the families of random variables $\{EXEC_{\pi,\mathcal{A},\mathcal{Z}}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^{poly(n)}}$ and $\{EXEC_{\rho,\mathcal{S},\mathcal{Z}}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^{poly(n)}}$ are computationally indistinguishable iff $\mathcal{L}$ is solvable and $\mathcal{M}$ is not solvable within polynomial time.

Obviously, the definition of $\mathcal{F}^{\text{post}}$ security is stronger that standard UC security: the definition of secure realization is defined by computational indinstinguishability between the

environment's output in the real world and the environment's output in the ideal world, just as the definition of UC secure realization of protocols. We additionally require that secure emulation is given iff one problem is solvable and another is not efficiently solvable. Therefore $\mathcal{F}^{\text{post}}$ security implies UC security. On the other hand, long-term UC security implies $\mathcal{F}^{\text{post}}$ security since for $\mathcal{F}^{\text{post}}$ security the outputs of the environment in the real world vs. in the ideal world have to be computationally whereas for long-term CU security the environment's output in the real world has to be statistically indistinguishable from the environment's output in the ideal world.

This concludes that $\mathcal{F}^{\text{post}}$ lies between standard UC security and long-term UC security.

Analogously, we relax the notion of long-term revealing functionalities:

Definition 3.2 ($(\mathcal{L},\mathcal{M})$-long-term revealing functionalities, compare [MU10], Definition 4.1) *Let P be a party identifier and S be a network. Let $trans_S$ denote the transcript of all communication between a functionality $\mathcal{F}$ and all other machines (including $\mathcal{A}$) in an execution of $S \cup F$. Let $trans_S \backslash P$ denote the transcript of all communication between $\mathcal{F}$ and all machines except P.*

Let furthermore $\mathcal{L}$ and $\mathcal{M}$ be two problems.

We say a functionality $\mathcal{F}$ is $(\mathcal{L},\mathcal{M})$-long-term revealing (LTR) for party P if the following holds for any network S: There is a deterministic function f_S that is efficiently computable iff $\mathcal{L}$ is solvable and $\mathcal{M}$ is not solvable within polynomial time such that with overwhelming probability we have $trans_S = f_S(n, trans_S \backslash P)$

This weakened variant of the definition of long-term security suffices for our cases since our goal is not everlasting security but security against an adversarial machine that gains computational power during the computation. Therefore we only require *computational* indistinguishability for the emulation and define the function computing the communication of the machines in the definition of long-term revealing functionalities to to be efficiently computable.

For a better readability, we mostly omit stating the two problems $\mathcal{L}$ and $\mathcal{M}$ when those are clear from the context (i. e. we mostly will state $\mathcal{F}^{\text{post}}$ secure and therefore mean $(\mathcal{L},\mathcal{L})$-$\mathcal{F}^{\text{post}}$ secure for given problems $\mathcal{L}$ and $\mathcal{M}$).

When updating a commitment scheme, we want to switch the setup assumption of the commitment scheme in question from a weaker one to a stronger one.

Since the adversary may break the old setup assumption during the update, we want to reassure that the commitment scheme that has to be updated cannot be broken at the beginning of the update process (i. e. that the adversary does not learn the committed values and is not able to equivocate the commitment). Therefore we want the commitment scheme that has to

be updated and which produces the initial commitments in the update process to be long-term secure.

When looking at UC secure commitments in the $\mathcal{F}_{\text{CRS}}$-hybrid model, the CRS created by the simulator can only be computationally indistinguishable from the real CRS since the simulator has to have trapdoors for both equivocality and extractability.

In the case of a statistically binding commitment, the CRS is statistically indistinguishable from a real-world CRS if the committer is corrupted and in this case of corruption the environment's view in the real world is statistically indistinguishable from the environment's view in the ideal world. Thus, we can get the notion of long-term security for statistically binding commitments if we only view the case of a corrupted committer. If we look at the case of a corrupted recipient then we have that both the CRS and the environment's view are only computationally indistinguishable from the CRS and the environment's view in the real world.

The same goes with statistically hiding commitments: in the case of a corrupted recipient the CRS generated by the simulator is statistically indistinguishable from a CRS generated by $\mathcal{F}_{\text{CRS}}$ in the real-world protocol and the environment's view in the real world is statistically indistinguishable from $\mathcal{Z}$'s view in the real world.

Therefore we have to split the commitment functionality in two functionalities: A functionality that defines a statistically binding commitment, called $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and a functionality that defines a statistically hiding commitment, called $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$.

We define the new functionality $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ to behave like $\mathcal{F}_{\text{com}}$ with the following changes: In the commit phase, the adversary gets the committed message. This will enable the simulator to generate a CRS that is statistically indistinguishable from the CRS produced by $\mathcal{F}_{\text{CRS}}$ in the real world. The resulting functionality $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ can be seen in Figure 3.1.

$\mathcal{F}_{\text{com-binding}}^{\text{post}}$ for statistically binding commitments

Let C and R be two parties. Let $n \in \mathbb{N}$ be the security parameter and $l(n) \in \mathbb{N}$ be the length of the message space depending on n. The functionality $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ behaves as follows:
Upon the input $m \in \{0,1\}^{l(n)}$ from C, generate a public delayed output (committed) to R and send `(committed, m)` to $\mathcal{S}$. additionally to $\mathcal{S}$.
Upon input `(unveil)` from C generate a public delayed output `(unveil, m)` to R.

Figure 3.1: The functionality $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ for statistically binding commitments (compare [MU10], Definition 3.8).

We also want to see whether statistically hiding commitments in the $\mathcal{F}_{\text{CRS}}$-hybrid model can be long-term secure. Therefore we will again define a functionality that is based on $\mathcal{F}_{\text{com}}$

with some changes in the behavior.

Since the ideal-world adversary has to generate a CRS that is statistically indistinguishable from the real-world CRS, it is not able to generate a trapdoor for commitment extraction. Therefore we have to modify the unveil phase of $\mathcal{F}_{\text{com}}$: we let the committer send the commit message again in the unveil phase when it gets the unveil information from the committer.

The resulting functionality, denoted as $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$, can be seen in Figure 3.2.

$\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ for statistically hiding commitments

Let C and R be two parties. Let $n \in \mathbb{N}$ be the security parameter and $l(n) \in \mathbb{N}$ be the length of the message space depending on n. The functionality $\mathcal{F}_{\text{com}}^{\text{post}}$ behaves as follows:
Upon the input $m \in \{0,1\}^{l(n)}$ from C, generate the public delayed `(committed)` to R.
Upon input `(unveil, m)`, from C, generate the delayed output `(committed, m)` and send the output to R.

Figure 3.2: The functionality $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ for statistically hiding commitments (compare [MU10], Definition 3.8).

This means that statistically binding or statistically hiding commitment schemes may be able to be updated if they long-term UC realize $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ or $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$.

First, we want to investigate whether statistically hiding and resp. statistically binding UC secure commitments can be long-term secure, i. e. if there exist statistically binding and resp. statistically hiding commitment protocols that long-term securely realize $\mathcal{F}_{\text{com-binding}}^{\text{post}}$, resp. $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$. Those commitment schemes can be candidates for a CRS update.

As an example we will use the commitment scheme in [DN02]. A statistically hiding variant of this commitment scheme was used in [MU10] to show that a statistically hiding commitment scheme in the $\mathcal{F}_{\text{crs}}$-hybrid model can not be proven as long-term UC secure since the CRS generated by the simulator is only computationally indistinguishable from the CRS provided by $\mathcal{F}_{\text{crs}}$ in the real world.

In Section 3.1 we will provide an overview of the basic (computationally hiding and binding) commitment scheme. In Subsection 3.1.1 we will investigate whether the statistically binding variant of the commitment scheme in [DN02] can be proven to long-term UC-realize $\mathcal{F}_{\text{com-binding}}^{\text{post}}$. In Subsection 3.1.2 we will do the same with the statistically hiding variant of this commitment scheme.

3.1 The DN Commitment Scheme

In this section we show that there exist UC secure commitment schemes that long-term securely realize $\mathcal{F}^{\mathbf{post}}_{\mathbf{com-binding}}$, resp. $\mathcal{F}^{\mathbf{post}}_{\mathbf{com-hiding}}$.

As an example we take the commitment scheme by Damgård and Nielsen [DN02] since this commitment scheme has two variants.

The basic commitment scheme is a commitment scheme that is both computationally binding and hiding. The scheme is based on a mixed commitment scheme which was introduced in [DN02]. A mixed commitment scheme is a commitment scheme that comes with two different keys for commitment generation. One type of key, called an X-key allows generating a commitment which is extractable when having the trapdoor t_N for a system key N. When having the other type of key, called E-key together with its trapdoor t_{K_E}, one can generate a fake commitment that is equivocal, i. e. it can be unveiled to an arbitrary message from the message space. The last but most significant property of both key types is that those two type of keys are computationally indistinguishable, meaning that given such a commitment, no computationally bounded algorithm can decide whether this commitment is extractable or equivocal.

For a formal definition, we cite [DN02], Definition 1:

Definition 3.3 (Mixed commitment scheme (see [DN02], Definition 1)) *A mixed commitment scheme $commit_K$ is a commitment scheme with some global system key N which determines the key space $\mathcal{K}_N$ and message space $\mathcal{M}_N$. The key space contains two subsets (the E-keys and the X-Keys) for which the following holds:*

- *Key Generation: The system key N can be generated efficiently, together with the so-called X-trapdoor t_N (this trapdoor is the trapdoor for all X-Keys in $\mathcal{K}_N$). Given N, one can efficiently generate an X-Key and an E-Key K together with its corresponding E-trapdoor t_K, as well as commitment keys.*
- *Key Indistinguishability: Random E-Keys and random X-Keys are computationally indistinguishable from random keys as long as the X-trapdoor is not known.*
- *Equvocality: Given the E-Key K and trapdoor t_K, one can efficiently generate fake commitments c which are exactly distributed as real commitments, which can be later unveiled to an arbitrary message $m' \in \mathcal{M}_N$ out of the message space, i e. one can compute a uniformly random value r for which $c = COM_K(m', r)$ holds.*
- *Extractability: Given a commitment $c = COM_K(m, r)$ generated with an X-Key K, one can efficiently extract the message m out of c using the trapdoor t_N for system key N.*

[DN02] remark that as long as the trapdoor for the X-Keys is not known, a commitment generated with any key out of $\mathcal{K}_N$ is computationally hiding and as long as both the trapdoor for an E-Key and the trapdoor for the X-Keys are not known, a commitment generated with any key out of $\mathcal{K}_N$ is computationally binding. This is due to the fact, that E-Keys, X-Keys and random keys are computationally indistinguishable.

For computing a commitment c, we need some randomness which is drawn from a randomness space under system key N. We will denote this space as $\mathcal{R}_N$.

For the UC-commitment scheme, [DN02] require a special form of mixed commitment schemes, called *special mixed commitment schemes*. This type of commitment scheme is a mixed commitment scheme as defined in Definition 3.3 with the following additional properties:

- The sets $\mathcal{K}_N$ and $\mathcal{M}_N$ have to be finite groups.
- The amount of E-Keys has to be negligible compared to $|\mathcal{K}_N|$ and the number of X-Keys has to be overwhelming compared to $|\mathcal{K}_N|$. This means that there is a negligible fraction of keys in $\mathcal{K}_N$ which are neither E-Keys nor X-Keys.

Now we describe the actual UC commitment scheme, which we from now on denote as the DN commitment scheme.

The CRS of this basic scheme consists of two keys: the system key N along with an E-key K_1 which is generated using the system key N.

For committing, the committer and the recipient first execute a coin toss for sampling a key for the actual commitment. Therefore the committer draws a random key $K_C \leftarrow \mathcal{K}_N$ for the system key N and generates a commitment $c_1 = COM_{K_C}(K_1, r_1)$ on K_1 using the key K_1 given in the CRS and some randomness $r_1 \in \mathcal{R}_N$. The recipient then draws a random key $K_R \leftarrow \mathcal{K}_N$ for the system key N and sends K_R to C. In the last step of the coin toss, the committer sends the unveil information K_C and r_1 for commitment c_1 to R and both parties compute the new commitment key $K = K_1 \oplus K_2$. Then the committer generates the actual commitment on its input $m \in \mathcal{M}_N$: it generates a commitment $c_2 = COM_K(m, r)$ using the underlying mixed commitment scheme with randomness $r \in \mathcal{R}_N$.

Since the committer commits on a key for tossing a commitment key, we will require the key space $\mathcal{K}_N \subseteq \mathcal{M}_N$ to be a subset of the message space for system key N.

In the unveil phase, the committer simply sends the unveil information m and r for commitment c_2 to R. The recipient then checks whether the commitment c_2 is a commitment on m using r. If this is the case, R outputs `(unveil, m)`

As stated in [DN02], the DN commitment scheme has two variants: one variant which is perfectly binding and computationally hiding and the other variant is perfectly hiding and

computationally binding. In the next two sections we will describe both variants and show that both variants long-term securely realize $\mathcal{F}^{\text{post}}_{\text{com-binding}}$, resp. $\mathcal{F}^{\text{post}}_{\text{com-hiding}}$.

3.1.1 The Statistically Binding DN Commitment

For the perfectly binding DN-commitment scheme, an additional key is added to the CRS of the basic DN-commitment scheme which is an X-key K_X. This key enables the simulator to extract the committed value out of the commitment when the committer is corrupted without generating a CRS that is statistically distinguishable from a CRS generated in the real-world protocol by $\mathcal{F}_{\text{CRS}}$. Additionally, when sending the actual commitment c_2 on its input x, the committer generates an additional commitment $c_3 = COM_{K_X}(x)$ on m using the X-key and sends both c_2 and c_3 to R. In the simulation, the simulator is then able to extract m out of c_3 using the trapdoor t_N for system key N. The protocol is represented in Figure 3.3.

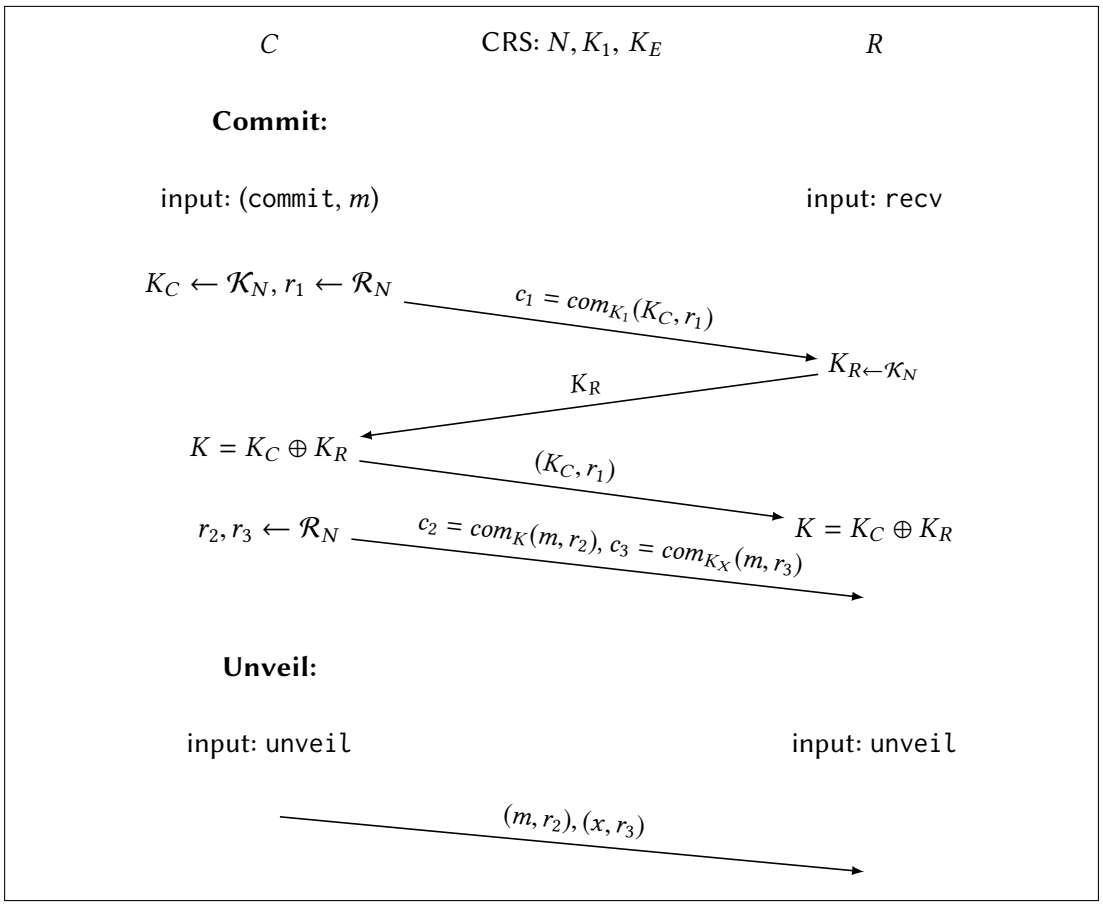

Figure 3.3: The statistically binding DN commitment scheme, cmp. [MU10], Figure 3.

Since the commitment scheme is perfectly binding and the simulator can generate a CRS in an honest manner, we can show that the perfectly binding DN-commitment scheme long-term UC-realizes $\mathcal{F}_{\text{com-binding}}^{\text{post}}$:

Lemma 3.4 *If COM is a special mixed commitment scheme, then the perfectly binding DN-commitment shown in Figure 3.3 securely long-term UC-realizes* $\mathcal{F}_{\text{com-binding}}^{\text{post}}$.

Proof. To prove our lemma we have to look at two different cases that can occur in an execution of the protocol: the case that C is corrupted and the case that R is corrupted. For each case we have to construct a simulator $\mathcal{S}$ that generates a CRS and communicates with $\mathcal{Z}$ such that $\mathcal{Z}$'s view in the ideal world is statistically indistinguishable from the environment's view in the real world.

The committer C is corrupted and R is honest In this case we assume the dummy adversary. Then the environment directly provides input to C (since the dummy adversary directly passes the messages on to C) and hence $\mathcal{Z}$ directly provides messages to R. $\mathcal{Z}$ also receives messages from the honest R (via the dummy adversary). This means that we have to construct a simulator that interacts with an environment in such a way that the view of any possible computationally bounded $\mathcal{Z}$ in the ideal world is statistically indistinguishable from the environment's real-world view. The simulator $\mathcal{S}_C$ simulating a corrupted committer is constructed as follows:

- The CRS is generated as follows: Generate a random system key N. Furthermore generate a random E-key K_1 and a random X-key K_X. In this case $\mathcal{S}_C$ knows the trapdoors for K_1 and K_X.
- $\mathcal{S}_C$ simulates an honest and unmodified instance of the recipient R.
- When receiving a message c_1 from $\mathcal{Z}$, simulate sending c_1 to R.
- Draw a random key $K_R \leftarrow \mathcal{K}_N$, as in the protocol and simulate receiving it from R.
- When receiving the reveal information (K_C, r_1) for c_1 from $\mathcal{Z}$, check whether $c_1 = COM_K(K_C, r_1)$ and abort if this is not the case. Else compute $K = K_C \oplus K_R$ and simulate sending the reveal information to the internal simulation of R.
- When receiving the message (c_2, c_3) from $\mathcal{Z}$, check that c_3 is extractable using the trapdoor for system key N. If the check fails, abort the simulation. Then extract the commit message m out of c_3 using the trapdoor for system key N and send m to $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and simulate sending (c_2, c_3) to the internal simulation of R.

- When receiving reveal information (m', r_2, r_3) from $\mathcal{Z}$, check that $m = m'$ and that $c_2 = COM_K(m,r_2)$ and $c_3 = COM_K(m, r_3)$. If this is not the case, then abort. Else send `(reveal)` to $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and simulate sending (m', r_2, r_3) to the internal simulation of R.

Obviously the CRS is generated as in the real-world protocol and hence such a CRS is statistically indistinguishable from a real-world CRS.

The coin-toss in steps two to step four is also done as a real committer and a real recipient would have done. Furthermore, if the execution is not aborted, it is guaranteed that the revealed K_C always equals the committed K_C. Also, if the simulation is aborted in the last step of the coin-toss, the reason is that the committed K_C does not equal the revealed K_C. In this case an honest R would abort the execution of the real-world protocol. In step four, the message is only sent to R, which would also happen in the real world.

In the reveal phase $\mathcal{S}_C$ sends the unveil information of c_2 and c_3 to R which corresponds to the message delivered in the reveal phase of the real-world protocol. The only difference in this step could occur if $m \neq m'$ and $c_2 = COM_K(m, r_2)$ and $c_3 = COM_{K_X}(m, r_3)$. But since c_3 is an perfectly binding commitment, this will happen only with negligible probability. To show this, we make a reduction and show that if $m \neq m'$ and $c_2 = COM_K(m, r_2)$ and $c_3 = COM_{K_X}(m, r_3)$ happens with non-negligible probability, then the commitment c_3 cannot be statistically binding.

Therefore assume for sake of contradiction that $m \neq m'$ and $c_2 = COM_K(m, r_2)$ and $c_3 = COM_{K_X}(m, r_3)$ happens with non-negligible probability.

Then we can construct an adversary that successfully equivocates the underlying commitment, breaking the perfectly binding property.

The adversary is constructed as follows:

- The input is a key N together with a trapdoor t_N and a X-Key K_X.
- Generate a random E-Key K_1 together with its trapdoor t_{K_1}.
- Run the ideal-world simulation with the following modifications:
  - Simulate the coin toss according to the simulation and get as result a key $K \in \mathcal{K}_N$.
  - When receiving a commitment (c_2, c_3) consisting of commitments c_2 generated with K and c_3 generated with K_X, use the trapdoor t_N to extract the committed message m out of c_3.
  - If in the unveil phase the environment sends unveil information m', r_2, r_3 with $m \neq m'$, $c_2 = COM_K(m', r_2)$ and $c_3 = COM_{K_X}(m', r_3)$ (i. e. the environment successfully

equivocates the commitment), find a proper $r \in \mathcal{R}_N$ such that $c_3 = COM_K(m, r)$ and output the tuple (m, r, m', r_3, c_3)

Now let us analyze the advantage of our adversary to equivocate a commitment generated with K_X: By assumption the probability that the environment can equivocate the commitment is non-negligible. This means that the environment has also to be able to successfully equivocate the underlying commitment c_3 with non-negligible probability and therefore our adversary outputs messages m, m' together with randomness $r \in \mathcal{R}_N$ for which holds $m \neq m'$ and $c_3 = COM_{K_X}(m, r) = COM_{K_X}(m', r_3)$ for some appropriate $r' \in \mathcal{R}_N$ with non-negligible probability.

This concludes that the commitment generated with K_X is not perfectly binding, which is a contradiction to the requirement that commitments computed using K_X are perfectly binding.

We conclude that in the case of a corrupted committer we can create a simulator that communicates with an environment in such a way that the environment's view in the ideal world is statistically indistinguishable from $\mathcal{Z}$'s view in the real world.

The recipient R is corrupted and C is honest Now we want to prove that simulating the corrupted recipient also results in the environment's view in the ideal world such that it is statistically indistinguishable from $\mathcal{Z}$'s view in the real world. Without loss of generality we assume the dummy adversary which only forwards the received messages from $\mathcal{Z}$ to C (via the corrupted R) and forwards all received messages from C (via R) to $\mathcal{Z}$. The simulator $\mathcal{S}_R$ is constructed as follows:

- The CRS is generated as follows: Draw a random system key N, a random E-key K_1 and a random X-Key K_X.
- $\mathcal{S}_R$ simulates an honest and unmodified instance of the recipient C.
- Upon receiving a message $(\texttt{commit}, m)$ from $\mathcal{F}^{\textbf{post}}_{\textbf{com-binding}}$, generate a random key $K_C \leftarrow \mathcal{K}_N$, compute the commitment $c_1 = COM_{K_1}(K_C, r_1)$ using randomness $r_1 \leftarrow \mathcal{R}_n$ and simulate receiving c_1 from C.
- When receiving a message K_R from $\mathcal{Z}$ where $K_R \in \mathcal{K}_N$ is a key, simulate sending it to C.
- In the last step of the coin toss simulate receiving the reveal information K_C, r_1 from the internal simulation of C and compute $K = K_C \oplus K_R$.
- As last step of the commit-phase draw $r_2, r_3 \leftarrow \mathcal{R}_n$ randomly and compute commitments $c_2 = COM_K(m, r_2)$ and $c_3 = COM_{K_X}(m, r_3)$. Simulate receiving c_2, c_3 from the internal simulation of C.

- When receiving (unveil) from $\mathcal{Z}$, simulate receiving (m, r_2, r_3) from the internal simulation of C.

Again it is obvious that the CRS generated in the simulation is statistically indistinguishable: It is generated as in the real world.

Since the coin-toss is performed in an honest manner, the first three steps of the simulation are statistically indistinguishable from the first three steps of the real-world protocol.

In the reveal phase the simulated message is a valid opening information for the generated c_2 and c_3 and thus a valid reveal message for the commitment.

Since in all steps the simulator behaves in an honest manner and the simulation is never aborted (if the environment does not abort the simulation), even an unbounded environment $\mathcal{Z}$ would have zero probability distinguishing the ideal-world simulation from a real-world execution.

Since we have shown that in each of the possible two cases of corruption we can construct a simulator that communicates in such a way that the view of the environment is statistically indistinguishable from the real-world view of $\mathcal{Z}$ we conclude that the perfectly binding DN commitment scheme long-term UC-realizes $\mathcal{F}_{\mathbf{com-binding}}^{\mathbf{post}}$. □

3.1.2 The Statistically Hiding DN Commitment

The next variation of the basic DN-commitment scheme is the perfectly hiding commitment scheme which additionally contains a commitment generated with an E-key. Therefore the CRS consists of the system key N and two E-keys: K_1 for executing the coin toss and K_E for generating the additionally equivocal commitment c_3 which is generated in the actual commit phase.

When executing the commitment scheme, both C and R together execute a coin toss as described in Section 3.1 to draw a random key K for commitment c_2. Then the committer draws a random message $\tilde{m} \leftarrow \mathcal{M}_K$ which has the same length as the input m and generates the commitment $c_2 = COM_K(m \oplus \tilde{m})$ and generates an additional perfectly hiding commitment $c_3 = COM_{K_E}(\tilde{m})$ on the randomly drawn message $\tilde{m}$. Then C sends both c_2 and c_3 to R. In the unveil phase, the committer sends the unveil information for both c_2 and c_3 to the recipient.

This commitment scheme scheme is obviously perfectly hiding: the commitment on the random message $\tilde{m}$ is perfectly hiding and therefore $m \oplus \tilde{m}$ statistically hides the message m.

For proving the security of the commitment scheme in the case of a corrupted recipient, we let the simulator act in an honest way if the committer is corrupted and in the case of a corrupted recipient, the simulator generates a fake commitment c_3 using the E-key K_E and its trapdoor t_{K_E} and generates a random commitment c_2 using the random commitment key

K produced in the coin toss phase. When getting the committed message m in the unveil phase by $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$, the simulator can then computes the unveil information $\tilde{m}$, $\tilde{r}$ such that $c_3 = COM_{K_E}(\tilde{m},\tilde{r})$. To show the indistinguishability between simulation and real protocol we will reduce the indistinguishability of the environment's view in the ideal world and in the real world to the equivocality property of the underlying mixed commitment scheme when using an E-Key: If the environment is able to distinguish between the simulation and the real execution of the commitment protocol, then we are able to construct a distinguisher that is able to distinguish between a genuine commitment $c = com_{K_E}(c,r)$ for a message $m \in \mathcal{M}_N$ and some randomness $r \in \mathcal{R}_N$ and a fake commitment c produced by using the trapdoor t_{K_E}.

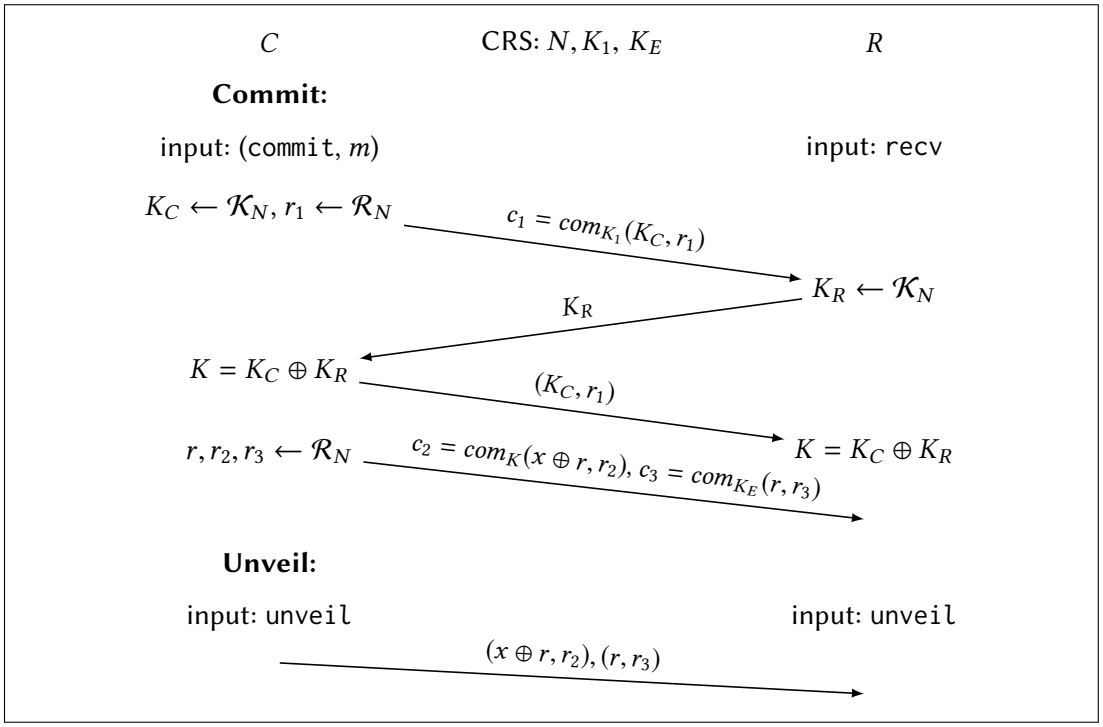

Figure 3.4: The statistically hiding DN commitment scheme, see [MU10], Figure 3.

We now formally prove that the statistically hiding DN commitment scheme long-term securely realizes $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$:

Lemma 3.5 *If COM is a special mixed commitment scheme, then the statistically hiding DN commitment scheme shown in Figure is long-term UC-realizes* $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$.

Proof. To prove our lemma we again have to show that we can construct a simulator that both generates a CRS and communicates with the environment in such a way that the environment's

view in the ideal world is statistically indistinguishable from its view in the real world. Again we have to look at two different cases of corruption: The case that only C is corrupted and the case that only R is corrupted. In each case we again have to construct a simulator $\mathcal{S}$ that communicates with the environment in such a way that environment's view in the ideal world is statistically indistinguishable from its view in the real world.

The committer C is corrupted and R is honest Without loss of generality we assume the dummy adversary which forwards all communication from $\mathcal{Z}$ to R (via the corrupted C) and forwards all communication from R to $\mathcal{Z}$. In a first step, we construct the simulator $\mathcal{S}_C$ corrupting the committer:

- The CRS is generated as follows: Draw random a system key N, and two E-keys K_1 and K_E.
- $\mathcal{S}_C$ simulates an honest and unmodified instance of the recipient R.
- When receiving a commitment c_1 from $\mathcal{Z}$, simulate sending c_1 to the internal simulation of R.
- Draw randomly a key $K_R \leftarrow \mathcal{K}_N$, as in the protocol and simulate receiving it from the internal simulation of R.
- When receiving the reveal information (K_C, r_1) for c_1 from $\mathcal{Z}$, check whether $c_1 = COM_K(K_C, r_1)$ and abort if this is not the case. Else compute the commitment key $K = K_C \oplus K_R$ and simulate sending the reveal information to the internal simulation of R.
- When receiving the message (c_2, c_3) from $\mathcal{Z}$, where the commitment c_3 was computed with K_E, set $m' = 0$, send m' to $\mathcal{F}_{\text{com}}^{\text{post}}$and simulate sending (c_2, c_3) to the internal simulation of R.
- When receiving reveal information $(m, \tilde{m}, r_2, r_3)$ from $\mathcal{Z}$, check whether $c_3 = COM_K(\tilde{m}, r_3)$ and $c_2 = COM_K(m \oplus \tilde{m}, r_2)$. If this is not the case, then abort. Else send $(\texttt{reveal}, m)$ to $\mathcal{F}_{\text{com}}^{\text{post}}$ and simulate sending $(m, \tilde{m}, r_2, r_3)$ to the internal simulation of R.

It is easy to see that the CRS is generated in an honest manner: it is drawn as $\mathcal{F}_{\text{CRS}}$ would draw the CRS in the real world.

The coin-toss which is simulated in the first three steps of the simulation is also done in an honest manner. More precisely, if the coin toss is not aborted it is guaranteed that the unveiled

K_C matches to the committed K_C. This concludes that the environment's view of the coin-toss in the ideal world and environment's view of the coin toss in the real world are identically distributed.

In the reveal phase, the simulator aborts the simulation if the commitments c_2 and c_3 cannot be opened to the received unveil information. In the real-world execution of the protocol, the recipient would also abort the execution of the protocol if the given unveil information does not match to the received commitments c_2 and c_3. This concludes that $\mathcal{S}_C$ aborts in the reveal phase if the real-world protocol is aborted in the reveal phase. If the simulation is not aborted, we have a valid opening for c_2 and c_3 which would also be sent by C to R in the real-world execution.

All in all, since there is no difference between the simulation and a real-world execution, even an unbounded $\mathcal{Z}$ cannot distinguish between the ideal-world simulation and the real world execution.

The recipient R is corrupted and the committer C is honest Now we prove that simulating a corrupted recipient results in an environment's view in the ideal world that is statistically indistinguishable from $\mathcal{Z}$'s view in the real world. Without loss of generality we can assume the dummy adversary. The constructed simulator $\mathcal{S}_R$ behaves as follows:

- The CRS is generated as follows: Draw a random system key N, and two random E-keys K_1 and K_E.
- $\mathcal{S}_R$ simulates an honest and unmodified instance of the recipient C.
- Upon receiving a message (`commit`) from $\mathcal{F}_{\text{com}}^{\text{post}}$, generate a random key K_C, compute $c_1 = COM_{K_1}(K_C, r_1)$ and simulate receiving c_1 from the internal simulation of C.
- Then draw a random key $K_R \leftarrow \mathcal{K}_N$ and simulate sending it to C.
- In the last step of the coin toss simulate receiving the reveal information (K_C, r_1) from the internal simulation of C and compute $K = K_C \oplus K_R$.
- As last step of the commit phase compute commitment $c_2 = COM_K(m', r_2)$ for a random message $m' \in \mathcal{M}_N$ using randomness $r_2, \in \mathcal{R}_N$ and produce a fake commitment c_3 together with an equivocation key ek_{c_3} using K_E and the trapdoor t_{K_E}. Simulate receiving (c_2, c_3) from the internal simulation of C
- When receiving (`unveil`, m) from $\mathcal{F}_{\text{com}}^{\text{post}}$, compute $\tilde{m} = m' \oplus m$ and use the equivocation key ek_{c_3} for the fake commitment c_3 to produce valid randomness $\tilde{r}$ for c_3 and $\tilde{m}$ and simulate receiving r, $\tilde{r}$, m and $\tilde{m}$ from the internal simulation of C.

It is easy to see that the CRS generated by $\mathcal{S}_R$ is generated in an honest manner and therefore such a generated CRS is statistically indistinguishable from a CRS generated in the real world.

Since $\mathcal{S}_R$ simulates C in an honest manner and it simulates sending the only message K_R received from $\mathcal{Z}$, the resulting simulation of the coin-toss is statistically indistinguishable from the coin toss phase of a real-world execution of the commitment scheme.

In the reveal phase, $\mathcal{S}_R$ computes the revealed $\tilde{m} = m \oplus m'$. Since $m' = m \oplus \tilde{m}$ is chosen at random, $\tilde{m}$ is also random. Since $\mathcal{S}_R$ has a trapdoor for K_E and since c_3 is perfectly hiding, it is possible for $\mathcal{S}_R$ to produce a fake commitment c_3 and later to compute reveal information $\tilde{m}, r_3'$ such that c_3 can be unveiled to $\tilde{m}$.

The commitment c_3 is perfectly hiding since fake commitments generated with an E-key together with its trapdoor are by definition identically distributed as genuine commitments and therefore no unbounded entity is able to distinguish a fake commitment from a genuine commitment produced by using a random E-key.

Since c_3 perfectly hides the random message $\tilde{m}$, c_2 also perfectly hides m and therefore even an unbounded environment is able to distinguish between the ideal world and the real world.

Hence in the case of a corrupted recipient there exists a simulator that communicates with the environment in such a way that the environment's view is statistically indistinguishable from $\mathcal{Z}$'s view in the real world.

We have shown that for each type of corruption we can construct a simulator such that the view of any environment is statistically indistinguishable from the real-world view of the environment and thus we have shown that the statistically hiding DN commitment long-term UC-realizes $\mathcal{F}_{\mathbf{com-binding}}^{\mathbf{post}}$. □

We conclude the chapter by stating that with the perfectly binding DN commitment protocol we have found a protocol that securely long-term UC realizes $\mathcal{F}_{\mathbf{com-binding}}^{\mathbf{post}}$ and with the perfectly hiding DN commitment protocol we have found a protocol that securely long-term UC realizes $\mathcal{F}_{\mathbf{com-hiding}}^{\mathbf{post}}$. Thus both protocols seem to be candidates for a CRS update.

4 The Update Process For the CRS of A Commitment

In this chapter we want show how to update a a fixed amount of CRS of an updated UC-secure commitment. By updating a CRS we mean instantiating a new CRS using a coin toss based on the new setup assumption. Therefore we want to use a coin toss to draw randomly each bit of the new CRS. For the coin toss we will use a modified commitment scheme that uses a modification of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$.

For the update process of the CRS we want to use a commitment of the commitment scheme that is based on the old setup assumption as an initial commitment. Since the adversary may increase its computational power and therefore may break the input privacy of the initial commitment protocol, we require the initial commitment to long-term UC-realize the functionality $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ or $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$. Using one of the functionalities one may assume that a $\mathcal{F}^{\text{post}}$ secure coin toss was possible.

Unfortunately, such a coin toss does not exist, as we will see below. Yet, we will find a way to instantiate a $\mathcal{F}^{\text{post}}$ secure coin toss using a $\mathcal{F}^{\text{post}}$ secure modification of a UC secure commitment scheme whose CRS is instantiated using a stand-alone secure coin toss in a modification of the $\mathcal{F}_{\text{com-binding}}^{\text{post}}$-hybrid model.

The commitment scheme using the modification of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ is a single-use commitment, meaning one instance of the protocol can only produce one secure commitment. For an arbitrary use of a coin toss we have to use a commitment scheme for which one instance can be used to produce several commitments, i. e. which is a multi-use commitment.

In Section 4.1, we will show that it is impossible to instantiate a $\mathcal{F}^{\text{post}}$ secure coin toss in the $\mathcal{F}_{\text{com-binding}}^{\text{post}}$- or in the $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$-hybrid model and define modified variant of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ such that a stand-alone secure coin toss is possible. Then we show how to modify a UC commitment scheme such that it is $\mathcal{F}^{\text{post}}$ secure using the variant of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$.

$\mathcal{F}_{\text{ct}}$ for l-bit coin toss

Let P_1 and P_2 be two parties.
The functionality $\mathcal{F}_{\text{ct}}$ behaves as follows:
Upon input `(init)` from parties P_1 and P_2, draw a random l-bit string $r \in \{0,1\}^l$ and generate a public delayed output r to P_1, P_2 and $\mathcal{S}$.

Figure 4.1: The functionality $\mathcal{F}_{\text{ct}}$ for l-bit coin toss, see [MU10], Definition 3.6.

4.1 The Coin Toss for the CRS

In this section, we show two things: First, we show that there does no coin toss exist in the $\mathcal{F}^{\text{post}}_{\text{com-binding}}$- or in the $\mathcal{F}^{\text{post}}_{\text{com-hiding}}$-hybrid model that securely $\mathcal{F}^{\text{post}}$-realizes $\mathcal{F}_{\text{ct}}$.

For an overview of the coin toss, the functionality $\mathcal{F}_{\text{ct}}$ as defined in [MU10] is shown in Figure 4.1.

In Subsection 4.1.2 we show how the functionality $\mathcal{F}^{\text{post}}_{\text{com-binding}}$ has to be modified such that a stand-alone secure bit coin toss is possible using the modified functionality. Lastly, in Subsection 4.1.3 we show that the modified version of the functionality $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$ cannot be used to instantiate a long-term UC-secure commitment and explain why this fact does not cause any problems for updating a CRS.

4.1.1 On the Impossibility of An F^{post} Secure Coin Toss

To be able to update a CRS of a commitment scheme, we would like to generate a $\mathcal{F}^{\text{post}}$ secure coin toss in the $\mathcal{F}^{\text{post}}_{\text{com-binding}}$-hybrid or in the $\mathcal{F}^{\text{post}}_{\text{com-hiding}}$-hybrid model. Unfortunately, as we will see in this subsection, there is no such coin toss protocol that is UC-secure (and hence not $\mathcal{F}^{\text{post}}$ secure). The reason for this is that if a coin toss protocol was UC-secure in the $\mathcal{F}^{\text{post}}_{\text{com-binding}}$-hybrid model or in the $\mathcal{F}^{\text{post}}_{\text{com-hiding}}$-hybrid model, then the commitment functionality $\mathcal{F}_{\text{com}}$ was UC-realizable in the plain model, which is an obvious contradiction to Theorem 6 stated in [CF01].

We will prove the impossibility of a UC secure coin toss in the $\mathcal{F}^{\text{post}}_{\text{com-binding}}$- and the impossibility of a UC secure coin toss in the $\mathcal{F}^{\text{post}}_{\text{com-hiding}}$- hybrid model. Since $\mathcal{F}^{\text{post}}$ security implies UC security, those statements also hold for the impossibility of $\mathcal{F}^{\text{post}}$ secure coin tosses in the $\mathcal{F}^{\text{post}}_{\text{com-binding}}$- or in the $\mathcal{F}^{\text{post}}_{\text{com-hiding}}$-hybrid model.

To prove that there does not exist a UC secure coin toss in the $\mathcal{F}^{\text{post}}_{\text{com-binding}}$- or in the $\mathcal{F}^{\text{post}}_{\text{com-hiding}}$-hybrid model, we will state several lemmas.

First, we will show that $\mathcal{F}_{\text{ct}}$ is a UC-complete functionality, meaning that any functionality can

be securely UC-realized in the $\mathcal{F}_{\text{ct}}$-hybrid model. Then we want to show that both functionalities $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ are UC-incomplete. The idea for this proof is the following: UC-completeness is closed under composition. This means that a protocol using a functionality $\mathcal{F}$ can only securely UC-realize a UC-complete functionality if $\mathcal{F}$ is UC-complete.

To show that both $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ are not UC-complete, we show that both functionalities can be realized in the plain model and that it is impossible to securely UC-realize $\mathcal{F}_{\text{com}}$ in the $\mathcal{F}_{\text{com-binding}}^{\text{post}}$- or in the $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$-hybrid model. We will show this via contradiction: If $\mathcal{F}_{\text{com}}$ was UC-realizable using one of the two functionalities, then (by using the composition theorem) $\mathcal{F}_{\text{com}}$ was UC-realizable in the plain model, which is an obvious contradiction to Theorem 6 in [CF01].

As the first step we formally state without proving this lemma that $\mathcal{F}_{\text{com}}$ is UC-complete:

Lemma 4.1 (UC-completeness of $\mathcal{F}_{\text{com}}$, compare [Can+02], Theorem 8.3) *Assume that enhanced trapdoor permutations exist. Then there exists for any well-formed functionality $\mathcal{F}$ a protocol in the $\mathcal{F}_{\text{com}}$-hybrid model that securely UC-realizes $\mathcal{F}$.*

An intuitive idea why $\mathcal{F}_{\text{com}}$ is UC-complete is that $\mathcal{F}_{\text{zk}}$ is UC-complete: Theorem 8.3 in [Can+02] states that if enhanced trapdoor permutations exist, then every well-formed functionality can be realized in the $\mathcal{F}_{\text{zk}}$-hybrid model, which means that $\mathcal{F}_{\text{zk}}$ is UC-complete. Since $\mathcal{F}_{\text{zk}}$ can be realized in the $\mathcal{F}_{\text{com}}$-hybrid model and due to the composition theorem, every well-formed functionality can be realized in the $\mathcal{F}_{\text{com}}$-hybrid model (assumed that enhanced trapdoor permutations exist).

Now we want to show that $\mathcal{F}_{\text{ct}}$ is UC-complete, meaning any functionality is UC-realizable in the $\mathcal{F}_{\text{ct}}$-hybrid model, especially $\mathcal{F}_{\text{com}}$. We instantiate the CRS of a UC secure commitment scheme using a coin toss.

For the proof of this lemma, we will use a commitment scheme that is a multi-use commitment scheme, hence it is proven to UC securely realize $\mathcal{F}_{\text{mcom}}$, a multi-use extension of $\mathcal{F}_{\text{com}}$. But this commitment scheme can also be used as a UC secure single-use commitment scheme with syntactical changes only. Hence we will use this commitment scheme as an example for commitment scheme in our proof.

Lemma 4.2 (UC-completeness of $\mathcal{F}_{\text{ct}}$, compare [Can+02], Theorem 8.3) *Assume that enhanced trapdoor permutations with dense public keys exist. Then there exists for any well-formed functionality $\mathcal{F}$ a protocol in the $\mathcal{F}_{\text{ct}}$-hybrid model that securely UC-realizes $\mathcal{F}$.*

Proof. Since UC-completeness is closed under composition (meaning that if a protocol realizes a UC-complete functionality $\mathcal{F}$ in a hybrid model using functionality $\mathcal{G}$, then functionality $\mathcal{G}$

is also complete), we can use Lemma 4.1 to prove our lemma. Therefore we show that there exists a protocol that realizes $\mathcal{F}_{\text{com}}$ in the $\mathcal{F}_{\text{ct}}$-hybrid model. To show this any commitment protocol in the CRS-hybrid model which assumes enhanced trapdoor permutations with dense public keys suffices.

As an example we use the single-use variant of the UAHC commitment scheme by Canetti et al. in [Can+02]. This commitment scheme assumes enhanced trapdoor permutations and we are able to further assume the enhanced trapdoor permutations to have dense public keys.

The security of this commitment scheme is based on three primitives: a pseudorandom IND-CPA secure encryption scheme (i. e. a commitment scheme that produces ciphertexts that are pseudorandom, meaning that such a ciphertext is computationally indistinguishable from a real random bit string of the same length), an IND-CCA secure encryption scheme which both are based on enhanced trapdoor permutations (for which we can assume to have dense public keys) and a commitment scheme whose security is based on the Hamiltonian cycle problem.

The CRS which we will instantiate by using $\mathcal{F}_{\text{ct}}$ consist of a random image of a one-way function (which will be used for the underlying commitment based on Hamiltonicity) and the public keys for the IND-CPA and IND-CCA secure encryption schemes.

The modified protocol works as follows:

- Both C and R send input `(init)` to $\mathcal{F}_{\text{ct}}$ and get a random string which can be split into three strings of the CRS: the random image y of a one-way function f, a random public key pk_E for the IND-CPA secure encryption scheme E and a random public key $pk_{E_{cca}}$ for the IND-CCA secure encryption scheme E_{cca}.
- commit phase on bit b
  - On input b C computes:
    - $z = aHC(b, r)$ for random string $r \in \{0,1\}^n$
    - a ciphertext $c_b \leftarrow \mathbf{E}(E_{cca}(r))$ with randomness s
    - a random string c_{1-b} of length $|c_{1-b}| = |c_b|$
  - C sends (z, c_0, c_1) to R.
  - Upon receiving, R outputs `(Received)`
- reveal phase
  - C sends `(Dec,` b, r, s`)` to R
  - Upon receiving, R checks whether $y \stackrel{?}{=} aHC(b, r)$, $c_b = E(E_{cca}(r))$ under randomness s. If the verification succeeds, R outputs `Open,` b

Since the CRS is uniformly random, the CRS of this UC secure commitment scheme can be instantiated using $\mathcal{F}_{\text{ct}}$. Due to the composition Theorem, this commitment scheme is UC-secure and the lemma follows.

□

As a next step, we want to show that both $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ are realizable in the plain model by proving that a protocol which directly sends the committed message in the commit phase to the recipient securely UC-realizes $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$. This shows that $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ are too weak to be used in a UC-secure protocol.

Lemma 4.3 ($\mathcal{F}_{\text{com-binding}}^{\text{post}}$ is UC-realizable in the plain model) *There exists a protocol that securely UC-realizes $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ in the plain model.*

Proof. The protocol that realizes $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ in the plain model is the following:

- Commit Phase: On input message m, C sends m to R
- Reveal Phase: C sends `(unveil)` to R. Upon receiving `(unveil)` from C, R outputs m.

To show that the protocol above securely UC-realizes $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ in the plain model, we have to construct a simulator that interacts with the environment such that the view of the environment in the ideal model is computationally indistinguishable from the environment's view in the real world. Again we assume the dummy adversary which forwards all communication coming from the environment to the corrupted parties and vice versa.

Let us first look at the case when the committer is corrupted: The simulator $\mathcal{S}_C$ is constructed as follows:

- Upon receiving the commitment m which is the plain text message m, send m to $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and simulate sending m to the internal simulation of R.
- Upon receiving the message `(unveil)`, send the message `(unveil)` to $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ simulate sending the received message to the internal simulation of R.

It is easy to see that $\mathcal{Z}$ has zero probability in distinguishing the real commit phase from the ideal commit phase since the simulator behaves like an honest committer.

Since the simulator also behaves like an honest committer in the unveil phase, the environment also has zero probability in distinguishing the real unveil phase from the ideal unveil phase.

Now let us look us at the case when the recipient is corrupted. Therefore construct the simulator $\mathcal{S}_R$ as follows:

- Upon receiving the message m from $\mathcal{F}_{\text{com-binding}}^{\text{post}}$, simulate receiving m from the internal simulation of C.
- Upon receiving the message `(unveil)` from $\mathcal{F}_{\text{com-binding}}^{\text{post}}$, simulate receiving `(unveil)` from the internal simulation of C.

Since the simulator behaves like an honest recipient in the commit phase, $\mathcal{Z}$ has zero probability in distinguishing the real from the ideal commit phase.

In the reveal phase, the simulator also behaves like an honest recipient and therefore the environment has zero probability in distinguishing the real unveil phase from the ideal unveil phase.

This concludes the proof. □

Lemma 4.4 ($\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ is UC-realizable in the plain model) *There exists a protocol that securely UC-realizes $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ in the plain model.*

Proof. The protocol that realizes $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ in the plain model is the following:

- Commit Phase: C sends `(committed)` to R.
- Reveal Phase: C sends `(unveil,` m`)` to R. Upon receiving `(unveil)` from C, R outputs m.

To show that the protocol above securely UC-realizes $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ in the plain model, we have to construct a simulator that interacts with the environment such that the view of the environment in the ideal world is statistically indistinguishable from the environment's view in the real world.

Let us first look at the case when the committer is corrupted: The simulator $\mathcal{S}_C$ is constructed as follows:

- Upon receiving the message `(committed)`, send $\perp$ to $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ and simulate sending `(committed)` to the internal simulation of R.
- Upon receiving the message `(unveil,` m`)`, send the message `(unveil,` m`)` to $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ simulate sending the received message to the internal simulation of R.

It is easy to see that $\mathcal{Z}$ has zero probability in distinguishing the real commit phase from the ideal commit phase since the simulator behaves like an honest committer.

Since the simulator also behaves like an honest committer in the unveil phase, the environment also has during the unveil phase zero probability to distinguish the real unveil phase from the ideal unveil phase.

Now let us look us at the case when the recipient is corrupted. Therefore construct the simulator $\mathcal{S}_R$ as follows:

- Upon receiving the message `(committed)` from $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$, simulate receiving `(committed)` from the internal simulation of C.
- Upon receiving the message `(unveil, m)` from $\mathcal{F}_{\text{com-binding}}^{\text{post}}$, simulate receiving `(unveil, m)` from the internal simulation of C.

Since the simulator behaves like an honest committer in the commit phase, $\mathcal{Z}$ has zero probability in distinguishing the real from the ideal commit phase.

In the reveal phase, the simulator also behaves like an honest committer and therefore $\mathcal{Z}$ has zero probability in distinguishing the real world from the ideal world.

This concludes the proof. □

Since we have shown that both $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ are UC-realizable in the plain model, we are able to show that both $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ are UC-incomplete functionalities. An easy way to prove this is to show that both functionalities cannot be used to instantiate a UC-complete functionality, e. g. $\mathcal{F}_{\text{com}}$. Using this, we are finally able to show that $\mathcal{F}_{\text{ct}}$ cannot be instantiated using $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ or $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$.

Lemma 4.5 (UC-incompleteness of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$) *The functionality $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ is not UC-complete.*

Proof. Assume for contradiction that $\mathcal{F}_{\text{com-binding}}^{\text{post}}$was UC-complete. Then every well-formed functionality was realizable in the $\mathcal{F}_{\text{com-binding}}^{\text{post}}$-hybrid model, especially $\mathcal{F}_{\text{com}}$. Yet since $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ is UC-realizable in the plain model and due to the composition theorem, this would mean that $\mathcal{F}_{\text{com}}$ also is UC-realizable in the plain model, which is a contradiction to Theorem 6 in [CF01]. Therefore there exists at least one well-formed functionality which is not realizable in the $\mathcal{F}_{\text{com-binding}}^{\text{post}}$-hybrid model and thus $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ cannot be UC-complete. □

Lemma 4.6 (UC-incompleteness of $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$) *The functionality $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ is not UC-complete.*

The proof is analogous to proof of Lemma 4.5.

Since we have shown that both $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ and $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ are not UC-complete and that $\mathcal{F}_{\text{ct}}$ is UC-complete, we can combine those two theorems to show that $\mathcal{F}_{\text{ct}}$ is not UC-realizable in the $\mathcal{F}_{\text{com-binding}}^{\text{post}}$- or in the $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$-hybrid model. For the next corollary, we have to informally define two terms: bilateral protocols and terminating protocols (see [CF01], Chapter 3).

A protocol is bilateral if only two parties are computing throughout the execution (i. e. such a protocol is a two-party protocol). A terminating two-party commitment protocol is a commitment protocol in which the recipient accepts the messages coming from the honest committer with overwhelming probability.

For a formal statement of the impossibility of securely UC realizing $\mathcal{F}_{\text{ct}}$ we state our theorem in a similar way as [CF01], Theorem 6.

Corollary 4.7 ($\mathcal{F}_{\text{ct}}$ is not realizable using $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ or $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$, compare [CF01], Theorem 6) *There does no bilateral and terminating protocol exist that realizes $\mathcal{F}_{\text{ct}}$ in the $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ or in the $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ hybrid model. This even holds if the simulator is allowed to depend on the environment.*

Proof. Now we want to formally prove why $\mathcal{F}_{\text{ct}}$ neither can be realized in the $\mathcal{F}_{\text{com-binding}}^{\text{post}}$-hybrid model nor in the $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$-hybrid model.

Lemma 4.2 states that $\mathcal{F}_{\text{ct}}$ is UC-complete and since UC-completeness is closed under composition each UC-complete functionality can only be realized by a protocol that uses another UC-complete functionality. Since we have shown in Lemmas 4.5 and 4.6 that both $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$ are not UC-complete, the theorem directly follows. □

Remark 4.8 *These proofs are made towards UC security. Since $\mathcal{F}^{\text{post}}$ security implies UC security those impossibbility results also hold towards $\mathcal{F}^{\text{post}}$ security.*

Although we are not able to construct a UC secure coin toss neither in the $\mathcal{F}_{\text{com-binding}}^{\text{post}}$- nor in the $\mathcal{F}_{\text{com-hiding}}^{\text{post}}$-hybrid model, we are able define a stand-alone secure coin toss to $\mathcal{F}^{\text{post}}$ realize commitments in a modification of the $\mathcal{F}_{\text{com-binding}}^{\text{post}}$-hybrid model. In this variant, we only simulate only the committing party. For P_2, we have to show that the possible output is indistinguishable from a random value.

The modification of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and the resulting coin toss protocol are shown in the next section.

4.1.2 How to Modify the Commitment Functionality

In this section we will explain how the functionality $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ is to be modified such that a stand-alone secure version of a coin toss is possible. This will enable us to build a $\mathcal{F}^{\text{post}}$ secure commitment scheme using the modified version of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$ and therefore the resulting commitment protocol will be $\mathcal{F}^{\text{post}}$ secure based on the new computational, stronger setup assumption.

As we have seen, the reason why the proof failed was because in the case of a corrupted P_2, the simulator could not create a valid commitment c such that c would open correctly to $b_1 = b \oplus b_2$. A solution for this is to only simulate a corrupted P_1 and reduce the security of the coin toss to the the hiding property when P_2 is corrupted. This means that if we assume that an adversary $\mathcal{A}$ can alter the outcome the coin toss protocol if it corrupts the party P_2, we can construct an adversary $\mathcal{B}$ that is able to break the hiding property (i. e. it is able to assign a given commitment to its committed value without having any trapdoors to equivocate the commitment) of the underlying commitment scheme by using $\mathcal{A}$). There is another problem that occurs if P_2 is corrupted: In this case the adversary gets the committed value itself and therefore the adversary can generate a value b_2 dependent on b_1. A solution to prevent this is to let the adversary corrupting P_2 receive only the computationally hiding *commitment* of the committed value, not the committed value itself. Therefore the adversary corrupting P_2 gets access to an oracle to receive on request a commitment to the committed value.

This oracle can be viewed as a non-information oracle, which was introduced in [CK02] to relax the UC security for key exchange. According to [CK02], a non-information oracle is a special type of interactive Turing machine having the property that its output is computationally independent from its communication. More specifically, the non-infomation oracle behaves as follows: When requested by the functionality, it computes (interactively) a given requested value or on a function. Thereby the functionality lets the adversary interact with the non-information oracle throughout the computation. When one of the parties is corrupted, the adversary additionally learns the internal state of the non-information oracle.

There is a significant difference between our definition of the oracle and the non-information oracle as defined in [CK02]: upon corruption, the adversary learns the internal state of the non-information oracle whereas the adversary learns nothing about the internal state of the oracle used in our scheme. Yet, viewing our oracle as a non-information oracle suffices since a commitment can be viewed as an encryption of the committed value as long as the committed value is not unveiled.

The modified functionality $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$ depicted in Figure 4.2 works as follows: First it generates a CRS for the commitment and distributes the CRS to the ideal adversary $\mathcal{S}$. If the

committer is corrupted, then $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$ additionally sends the extraction trapdoor to $\mathcal{S}$ (this enables the committer to extract the commitment in the commit phase without having to generate the CRS by itself).

When $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$ gets a commit input from C, it sends the `committed` message to R and $\mathcal{S}$. If the recipient is corrupted and $\mathcal{S}$ requests a commitment on the committed message, then the non-information oracle computes (interactively with $\mathcal{S}$) the requested commitment.

The unveil phase is the same as the unveil phase of $\mathcal{F}_{\text{com}}$: When $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$ receives an input `unveil` from R, then it sends the output (`unveiled`, m) to R and $\mathcal{A}$, where m is the committed message.

$\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$ for unconditionally binding commitments using the non-information oracle

Let $n \in \mathbb{N}$ denote the security parameter and $l(n)$ denote the length of the committed message m. Let C and R be two parties. The functionality $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$ behaves as follows:

Generation of the CRS:

Generate a random string $crs \leftarrow C$ from the distribution C of CRS for the commitment and send crs to $\mathcal{A}$. If the committer is corrupted, send additionally the extraction trapdoor t_x to $\mathcal{A}$.

Generation of the commitment:

Upon the input $m \in \{0,1\}^{l(n)}$ from C, generate a public delayed output (committed) to R. If R is corrupted and $\mathcal{A}$ requests the non-information for a commitment, then generate a commitment c on m and send c on behalf of the non-information oracle to $\mathcal{A}$. If the commitment scheme used by the non-information oracle is interactive and $\mathcal{A}$ who is corrupting R requests the non-information oracle for a commitment, then generate the commitment c interactively with $\mathcal{A}$ according to the protocol.
Upon input `(unveil)` from C generate a public delayed output `(unveil,` m`)` to R and if the recipient is corrupted and a commitment c was requested in the commit phase, send the unveil information for c to $\mathcal{A}$.

Figure 4.2: The functionality $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$ using a non-information oracle for statistically binding commitments in the $\mathcal{F}_{\text{CRS}}$ hybrid model.

We then are able to create a stand-alone secure coin toss protocol in the $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$-hybrid model.

We differentiate between two versions of the coin toss protocol in the $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$-hybrid

model: a coin toss on a single bit which can be seen in 4.3 and a coin toss on an n-bit string which can be seen in 4.4.

Both protocols work in the same way except that the bit length of the bit coin toss is limited to 1.

Therefore we will explain the protocol $\Pi^{post}_{\text{CT}-bit}$: When both parties get the input `init`, P_1 first generates a random bit $b_1 \leftarrow \{0,1\}$ and commits on this random bits by sending the input message (`commit`, b_1) to $\mathcal{F}^{\textbf{post}}_{\textbf{com}-\textbf{binding}-\textbf{od}}$. P_2 then gets a `committed` message by $\mathcal{F}^{\textbf{post}}_{\textbf{com}-\textbf{binding}-\textbf{od}}$ and draws a random bit $b_2 \leftarrow \{0,1\}$ and sends the bit b_2 to P_1. In the last step of the protocol, P_1 unveils its commitment on b_1 by sending an `unveil` message o $\mathcal{F}^{\textbf{post}}_{\textbf{com}-\textbf{binding}-\textbf{od}}$ and P_2 gets the unveiled bit b_1. Then both parties are able to compute $b = b_1 \oplus b_2$ and output b.

Protocol $\Pi^{post}_{\text{CT}-bit}$

- On input `init` $b_1 \leftarrow \{0,1\}$, P_1 sends (`commit`, b_1) to $\mathcal{F}^{\textbf{post}}_{\textbf{com}-\textbf{binding}-\textbf{od}}$ to commit to bit b_1.
- When getting (`committed`) from $\mathcal{F}^{\textbf{post}}_{\textbf{com}-\textbf{binding}-\textbf{od}}$, P_2 draws a random bit $b_2 \leftarrow \{0,1\}$ and sends b_2 to P_1.
- P_1 then sends (`unveil`) to $\mathcal{F}^{\textbf{post}}_{\textbf{com}-\textbf{binding}-\textbf{od}}$ to send the unveil information to P_2, computes the bit $b = b_1 \oplus b_2$ and outputs b.
- When getting the unveil information to the commitment, P_2 computes $b = b_1 \oplus b_2$ and outputs b.

Figure 4.3: The protocol $\Pi^{post}_{\text{CT}-bit}$ for tossing a single bit.

To show the security of $\Pi^{post}_{\text{CT}-bit}$, we will show that in the case of a corrupted party P_1, the protocol is $\mathcal{F}^{\textbf{post}}$ secure by constructing a simulator that acts in an honest way. Since we are not able to show the $\mathcal{F}^{\textbf{post}}$ security of the protocol in case of a corrupted P_2 we instead reduce the security of the coin toss protocol to the computational hiding property of the underlying commitment scheme, making the protocol stand-alone secure. Therefore we partially use the proof of Theorem 6.7.2 in [Lin17].

Now we formally want to show that $\Pi^{post}_{\text{CT}-bit}$ is a stand-alone secure coin toss in the $\mathcal{F}^{\textbf{post}}_{\textbf{com}-\textbf{binding}-\textbf{od}}$-hybrid model.

Lemma 4.9 *The protocol $\Pi^{post}_{\text{CT}-bit}$ shown in figure 4.3 is a stand-alone secure coin toss protocol that tosses a single bit in the $\mathcal{F}^{\textbf{post}}_{\textbf{com}-\textbf{binding}-\textbf{od}}$ hybrid model.*

Protocol $\Pi_{\text{CT}-string}$

- Upon input `init`, P_1 draws a random l-bit string $r_1 \leftarrow \{0,1\}^l$ and sends (`commit`, r_1) to $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$ to commit to string r_1.
- When getting (`committed`) from $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$, P_2 draws a random n-bit string $r_2 \leftarrow \{0,1\}^l$ and sends r_2 to P_1.
- P_1 then sends (`unveil`) to $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$ to send the unveil information to P_2, computes the string $r = r_1 \oplus r_2$ and outputs r.
- When getting the unveil information to the commitment, P_2 computes $r = r_1 \oplus r_2$ and outputs r.

Figure 4.4: The protocol $\Pi_{\text{CT}-string}^{post}$ for tossing a random l-bit string.

Proof. To prove that the given protocol is long-term UC-secure if P_1 is corrupted, we have to build a simulator, such that $\mathcal{Z}$'s view is statistically indistinguishable from $\mathcal{Z}$'s real-world view in the case that P_1 is corrupted and that the output of the simulation is statistically indistinguishable from a random value. Since we want to use the functionality $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$ to achieve a coin toss, we cannot use the equivocality. Therefore we have to prove that if an adversary corrupting P_2 is able to bias the outcome into a value dependent on b_1 with non-negligible probability then the hiding property of the commitment generated by the non-information is violated.

First, consider the case that P_1 is corrupted. To show that no adversary can bias the outcome of the coin toss into a value dependent on b_1 with non-negligible probability, we have to build a simulator that communicates with the environment in such a way that no environment is able to distinguish between the ideal-world simulation and the real-world execution. Again, we assume the dummy adversary that forwards all communication from the environment to the corrupted party and vice versa.

The simulator behaves as follows:

- When getting b_1 as input to the functionality $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$, simulate sending (`committed`) to the internal simulation of P_2 and send (`init`) to $\mathcal{F}_{\text{ct}}$.
- When receiving b from $\mathcal{F}_{\text{ct}}$, compute $b_2 = b \oplus b_1$ and simulate receiving b_2 from the internal simulation of P_2.
- When receiving message (`unveil`) as input to $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$, simulate sending the unveil information to the commitment to string b_1 to the internal simulation of P_2 and

output b.

It is easy to see that the view of the environment in the ideal world is statistically indistinguishable from the view of the environment in the real world-execution of the protocol: Since the environment is not able to send another value to $\mathcal{F}^{\mathbf{post}}_{\mathbf{com-binding-od}}$ after the commit phase is executed, the environment can obviously not alter the random bit b_1. Since the bit b_1 is chosen at random, the bit $b_2 = b \oplus b_1$ computed by the simulator also is a random bit and therefore no environment is able to distinguish the real world from the ideal world.

Next, we want to prove that the interaction with a corrupted P_2 results in an output whose distribution is statistically close to the distribution of outputs generated by the real-world protocol. Since the protocol cannot be proven as $\mathcal{F}^{\mathbf{post}}$ secure we have to reduce the security of the protocol in the case that the party P_2 is corrupted to the hiding property of the commitment sent by the non-information oracle.

This part of the proof is analogous to the proof of Theorem 6.7.2 in [Lin17]. Therefore we only roughly sketch the proof. For further details, see [Lin17].

The main tool in this proof is to set the outcome to the bit b received by $\mathcal{F}_{\mathbf{ct}}$ and rewinding the adversary that wants to bias the outcome if the adversary sends another value than the bit $b_2 = b_1 \oplus b$ (defined by the simulator), where b_1 is randomly chosen by the simulator.

Informally, the simulator sends the message `(init)` to $\mathcal{F}_{\mathbf{ct}}$ in order to receive a uniformly random b.

Then it generates a random bit b_1, compute the commitment c on b_1 and simulate receiving the commitment from the internal simulation of the non-information oracle. Compare the answer bit b_2 and if $b_2 = b \oplus b_1$ output bit b and complete the simulation. Else if $b \neq b_1 \oplus b_2$ and the adversary is not rewound n times, then rewind the adversary. If the adversary is rewound n times, where n is the security parameter and the adversary has still not answered with a correct bit b_2, then abort the simulation.

The simulation works since the underlying commitment sent by the simulator is computationally hiding and therefore the probability that the simulation is aborted is only negligible conditioned on the event that $\mathcal{Z}$ does not abort the simulation. Since b_1 is chosen independently of b_2 and since the commitment on b_1 is computationally hiding, the bit b_2 is chosen independently of b_1. This means that each step in the simulation is statistically indistinguishable from the corresponding step in the execution of the real-world protocol and thus the transcript of the ideal simulation is computationally indistinguishable from the transcript of the real-world execution.

□

This protocol also is usable when tossing a coin multiple times. This is due to the commitment

functionality that distributes for each instance one unique CRS.

The only part of the coin toss that could make the coin toss malleable is the commitment sent by the non-information oracle. Yet, in each instance of $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$ a separate CRS is distributed. Since in each instance of the coin toss protocol one separate instance of $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$ is called, the coin toss in the $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$-hybrid model is non-malleable. If the functionality $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$ in the coin toss protocol is replaced by a protocol that securely long-term UC-realizes $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$, then by the composition theorem the composed protocol is also non-malleable.

4.1.3 Relation to Long-Term UC Security

If we want to see whether $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$ can be used to instantiate a long-term secure commitment scheme, we investigate whether $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$ is long-term revealing.

Unfortunately, the functionality $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$ is long-term revealing for both parties: the communication of both parties with $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$ can be computed via the communication of the adversary and $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$

This means that $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$ cannot be used to instantiate another long-term secure commitment scheme. Yet, to instantiate the coin toss for sampling the new CRS we only have to use a $\mathcal{F}^{\text{post}}$ secure commitment scheme in the $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$-hybrid model. In the next section we will see how a $\mathcal{F}^{\text{post}}$ secure commitment scheme can be generated in the $\mathcal{F}_{\text{com-binding}}^{\text{post}}$-hybrid model.

4.2 The UCC-OneTime Commitment Scheme

In this section which is based on [CF01], Section 4 we want to modify a UC secure commitment scheme such that it uses the coin toss protocol $\Pi_{\text{CT}-bit}^{post}$ to build a $\mathcal{F}^{\text{post}}$ secure commitment scheme in order to instantiate a $\mathcal{F}^{\text{post}}$ secure coin toss that is based on the stronger setup assumption. This $\mathcal{F}^{\text{post}}$ secure coin toss can then be used to sample the new CRS of the updated commitment scheme.

There are some requirements to the commitment scheme to be used for the coin toss: We want the commitment scheme to have a uniformly random CRS such that the CRS is able to be drawn using a coin toss. Also, we require the CRS to be able be split into two parts to assure that the commitment scheme is usable in the $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$-hybrid model. This enables the simulator to create a CRS with either an extraction trapdoor or an equivocation trapdoor. This ensures that when instantiating the CRS by using the coin toss in the $\mathcal{F}_{\text{com-binding-od}}^{\text{post}}$-hybrid

model in a non-modular way the modified commitment scheme is $\mathcal{F}^{\text{post}}$ secure.

One might consider using the unconditionally binding or the unconditionally hiding DN commitment described in Section 3.1 to use as the commitment scheme for the $\mathcal{F}^{\text{post}}$ secure coin toss. Unfortunately, neither the unconditionally binding DN commitment scheme nor the unconditionally hiding DN commitment scheme can be used for this. The answer is simple: To be able to use a commitment scheme for the CRS update, the commitment scheme has to have a uniformly random CRS which is not guaranteed in the case of the unconditionally binding commitment scheme:

Therefore each key used in the CRS has to be from the uniform distribution, especially the E-Keys. Yet, if we recall the definition of special mixed commitments, then we can easily see that the key space for the E-Keys has to be a subspace $E \subset \mathcal{K}_N$ and that the number of E-Keys has to be negligible compared to $\mathcal{K}_N$ whereas the number of X-keys has to be overwhelming compared to the $\mathcal{K}_N$ and an E-key has to be computational indistinguishable from an X-key. This concludes that an E-Key has to have the same length as an X-key. This means that even if $\mathcal{K}_N$ was uniformly distributed, then E is not.

Since an E-Key K_1 is used both in the CRS of the unconditionally hiding DN-commitent scheme and in the unconditionally binding commitment scheme, neither the unconditionally binding nor the unconditionally hiding variant of the DN-commitment scheme can be used for updating the CRS of another commitment scheme. This also means that the CRS of the perfectly binding DN commitment scheme cannot be updated using our update procedure.

Therefore we have to use another commitment scheme whose CRS can be uniformly random and its CRS can be splits into two distinctive parts.

A good example that fulfills this requirement is the UCC-OneTime commitment scheme by Canetti and Fischlin [CF01].

The UCC-OneTime commitment scheme is an unconditionally binding UC-secure commitment scheme that was introduced by Cannetti and Fischlin in 2001. This non-interactive and unconditionally binding commitment scheme has an specific CRS structure—it consists of two parts: a random string σ which has the length of $4n$ bits and two key pairs pk_0 and pk_1 for pseudorandom generators G_{pk_0} and G_{pk_1} which map bit strings of length n to bit strings of length $4n$. This form of the CRS is important for us since one part of the CRS enables the simulator to extract the commitment in the case of a corrupted committer whereas the other part of the CRS enables the simulator to equivocate the commitment by creating the corresponding trapdoor when the recipient is corrupted. Thus we can generate the CRS using two phases of coin tosses.

The pseudorandom generator is according to [CF01] a Blum-Micali-Yao generator [BM84; Yao82] with the modification that the function used is a trapdoor permutation, not a one-way

function. The pseudorandom generator consists of a trapdoor permutation f_{pk} and a hard-core predicate B for f_{pk} and expands n random bits to $4n$ bits. The generator G_{pk} is described as follows:

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

where $f_{pk}^{n}(r)$ is the n-th fold application of f_{pk} to r.

To ensure that the CRS of the update commitment scheme can be instantiated using a coin toss, we require the CRS to be uniformly random. One way to reassure this is to assume trapdoor permutations with dense public keys. Therefore we require the trapdoor permutation used for the pseudorandom generator to have dense public keys, meaning in this case that the distribution of public keys $\mathcal{D}$ used for the pseudorandom generator G_{pk} has to be $\mathcal{D} = \{0,1\}^k$ for key length k of public keys $pk \in \mathcal{D}$. Otherwise it is not guaranteed that a random k-bit string is a valid public key for a pseudorandom generator G_{pk}.

A UCC-OneTime commitment is done as follows: The committer first draws a random $r \in \{0,1\}^n$ and computes $G_{pk_b}(r)$ for given bit b to commit to. Then C computes the commitment, depending on the input bit b: if $b = 0$, then c is set to $c = G_{pk_0}(r)$, else to $c = G_{pk_1}(r) \oplus \sigma$. The commitment c is then sent to the recipient R. In the reveal phase the committer sends b and r as the unveil information to the recipient and R can then easily check whether commitment was generated correctly or not.

To be able to use the UCC-OneTime commitment scheme for a $\mathcal{F}^{\mathbf{post}}$ secure coin toss in order to update a CRS, we have to instantiate the CRS of the UCC-OneTime commitment scheme by using coin toss protocol $\Pi_{\mathbf{ct-bit}}$ in the $\mathcal{F}^{\mathbf{post}}_{\mathbf{com-binding-od}}$-hybrid model. The idea for the realization of the CRS of the is UCC-OneTime commitment scheme using the protocol $\Pi_{\mathbf{ct-bit}}$ is to split the CRS in two parts: the random string σ and the public keys pk_0 and pk_1 for which σ is a bit string of length $4n$ and the tuple (pk_0, pk_1) can be interpreted as a bit string of length $2k$ (since each key is of length k). Then we can use the coin toss $4n$ times to create σ and again $2k$ times to create (pk_0, pk_1) separately.

For the coin tosses used in this scheme we use the functionality $\mathcal{F}^{\mathbf{post}}_{\mathbf{com-binding-od}}$ as seen in Figure 4.2 for which the non-information oracle produces a statistically binding UC-secure commitment when requested. We define the distribution of the CRS used for the underlying commitment used by the non-information oracle as C. The hiding property of this commitment scheme is based on the old complexity assumption. This means that if the adversary has not increased its computational power during the coin toss phase of the commitment scheme then the adversary is not able to bias the outcome of the coin toss phase (which is the CRS for the actual commitment scheme) into some values selected by itself.

Protocol UCC-OneTime

- The CRS
  - (pk_0, pk_1), public keys for pseudorandom generators $G_{pk_0}, G_{pk_1} : \{0,1\}^n \to \{0,1\}^{4n}$
  - $\sigma \in \{0,\ 1\}^{4n}$, random string)
- commit phase
  - If the bit to commit is $b = 0$, C computes the commitment y as follows: $y = G_{pk_0}(r)$ for a randomly drawn $r \leftarrow \{,1\}^n$, else it computes $y = G_{pk_1}(r) \oplus \sigma$. Then it sends y to R.
- reveal phase
  - C sends b and r to R. R checks whether $y = G_{pk_0}(r)$ if $b = 0$ or whether $y = G_{pk_1}(r) \oplus \sigma$ if $b = 1$. If the check succeeds, R outputs `(unveil, b)`.

Figure 4.5: The original protocol UCC-OneTime, see [CF01], Figure 4.

In the simulation we let the simulator in the case of a corrupted committer draw two random key pairs (pk_0, sk_0) and (pk_1, sk_1) using the generation algorithm *Gen* and let the output of the first coin toss phase equal (pk_0, pk_1) and simulates the second coin toss phase in an honest manner. Then the simulator is able to extract the message m out of the commitment in the commit phase. In the case of a corrupted recipient the simulator simulates the first coin toss phase in an honest manner to randomly generate a tuple of public keys (pk_0, pk_1) and generates a fake string $\sigma = G_{pk_0}(r_0) \oplus G_{pk_1}(r_1)$ for two random k-bit strings r_0, r_1 and lets the output of the second coin toss phase be the fake string σ. In the unveil phase the simulator is then able to unveil the commitment to both 0 and 1.

To show the indistinguishability between simulation and protocol execution, we reduce in the case of a corrupted committer the security of the commitment scheme to the computational hiding property of the commitment generated by the non-information oracle. In the case of a corrupted recipient, the security of the commitment scheme will be reduced to the pseudorandomness of the strings produced by the pseudorandom generator G_{pk}.

Now we can formally prove that the UCC-OneTime in combination with the two phases of coin tosses is $\mathcal{F}^{\text{post}}$ secure:

Theorem 4.10 ($(\mathcal{L},\mathcal{M})$-$\mathcal{F}^{\text{post}}$-security of UCC-OneTime-CT) *Let $\mathcal{L}$ and $\mathcal{M}$ be two problems. Assume that trapdoor permutation with dense public keys exist, $\mathcal{L}$ is solvable and $\mathcal{M}$ is not solvable within polynomial time. If the trapdoor permutation used for the commitment scheme*

UCC-OneTime-CT is based on problem $\mathcal{M}$ and the commitments sent by the non-information oracle is based on problem $\mathcal{L}$, then the commitment protocol UCC-OneTime-CT described in figure 4.6 is an active and statistically binding commitment protocol in the $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$-hybrid model that securely $(\mathcal{L}, \mathcal{M})$-$\mathcal{F}^{\text{post}}$ realizes $\mathcal{F}_{\text{com}}$.

Proof. We prove this theorem in a similar way as the proof of Theorem 7 in [CF01].

To show that this commitment scheme is $(\mathcal{L}, \mathcal{M})$-$\mathcal{F}^{\text{post}}$ secure, we have to construct a simulator such that the environment's output in the ideal world is computationally indistinguishable from the its output in the real world iff problem $\mathcal{M}$ is not solvable within polynomial time.

When constructing the simulator, we have to distinguish between two cases: The case in which the committer is corrupted and the case in which the recipient is corrupted. Additionally we have to look at the case that no party is corrupted to show that the protocol is a non-trivial protocol, meaning that even if no party is corrupted, R generates an output.

The committer C is corrupted and the recipient R is honest Without loss of generality we assume the dummy adversary which forwards all communication from and to the environment to and from the corresponding party (via the corrupted party C). As a first step, we construct the following simulator $\mathcal{S}_C$:

- Draw $2k + 4n$ random CRS $crs_1, ..., crs_{k+4n} \in C$ and simulate receiving them from the $2k + 4n$ instances of $\mathcal{F}^{\text{post}}_{\text{com-binding}}$.
- Generate two key pairs $(pk_0, td_0) \leftarrow Gen(1^n)$ and $(pk_1, td_1) \leftarrow Gen(1^n)$ and interpret (pk_0, pk_1) as a bit string $s = s^1 \cdot ... \cdot s^{2k}$ of length $2k$
- Simulate the $2k$ coin tosses for (pk_0, pk_1) sequentially in the following way (let i be the counting variable for the number of coin tosses):
  - When receiving the message $(\texttt{commit}, s_1^i)$ as the input to the i-th instance of functionality $\mathcal{F}^{\text{post}}_{\text{com-binding}}$, simulate sending the message to the internal simulation of $\mathcal{F}^{\text{post}}_{\text{com-binding}}$.
  - Compute $s_2^i = s_1^i \oplus s^i$ and simulate receiving s_2^i from the internal simulation of R.
  - When receiving the message `unveil` as input to the i-th instance of functionality $\mathcal{F}^{\text{post}}_{\text{com-binding}}$, simulate sending the message to the internal simulation of the i-th instance of $\mathcal{F}^{\text{post}}_{\text{com-binding}}$.
- Simulate the $4n$ coin tosses for σ sequentially in the following way (let i be the counting variable for the number of coin tosses):

  - Simulate receiving the subroutine output (`committed`) from the $2k + i$-th instance of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$.
  - When getting a request to the non-information oracle, generate a random commitment $c_2^i = com(b)$ for a random bit b using crs_{2k+i} and simulate receiving c_2^i from the internal simulation of the non-information oracle.
  - When receiving a random bit t_2^i, simulate sending it to the internal simulation of R
  - Simulate receiving the unveil information to c_2^i from the internal simulation of the $2k + i$-th instance of functionality $\mathcal{F}_{\text{com-binding}}^{\text{post}}$.

- Set σ to the concatenation of t^1 to t^{4n}, i. e. $\sigma = t^1 \cdot \ldots \cdot t^{4n}$
- When receiving a commitment y, check whether y lies within the image of G_{pk_0} by using the trapdoor td_0 for generator G_{pk_0}. If this is not the case, send the bit 1 to $\mathcal{F}_{\text{com}}$, else send 0 to $\mathcal{F}_{\text{com}}$ and simulate sending y to the internal simulation of R.
- When receiving the unveil information r' and b', check whether $b = b'$ and if y matches to the commitment to b' with r'. If this is not the case, abort the simulation. Else, simulate sending the unveil information to the internal simulation of R.

The coin tosses generating the public keys are done in an honest manner. This is the case since the public keys for the pseudorandom generators are both randomly generated using the key generation algorithm and then simulating the $2k$ coin tosses. Since we require the trapdoor permutation used for the pseudorandom generator to have dense public descriptions, the generation algorithm `Gen` producing a public-private key pair for a pseudorandom generator G_{pk} produces a uniformly random public key $pk \in \{0,1\}^k$. Thus the public keys produced using `Gen` and the public keys produced with the first coin toss phase are randomly drawn from the same distribution. Therefore the environment is not able to distinguish between the ideal-world coin tosses generating the public keys and the real-world coin tosses of the same type.

In the coin tosses carried out for generating σ the simulator behaves in an honest manner. Therefore the environment is not able to distinguish the coin tosses for generating σ in the ideal world from the coin tosses carried out in the real world.

In the commit phase, the simulator only extracts the committed value out of the commitment and does not deviate from the protocol. Therefore the commit phase is perfectly indistinguishable from a real commit phase.

If in the reveal phase the real-world committer does not unveil the commitment to another value than it committed to, then the ideal unveil phase is computationally indistinguishable

from a real unveil phase. Therefore consider the case that the probability that there exist commitments that can both be unveiled to 0 and 1 is non-negligible, i. e. that there exist values y that lie both in the image of G_{pk_0} and G_{pk_1}. Then with non-negligible probability the random string σ created in the second instance of coin tosses has to be pseudorandom. This would mean that the environment was able to bias the outcome of second coin toss phase with non-negligible probability to a value dependent on the values sent by the simulator with non-negligible probability. Therefore we have to show that if the probability for creating such a pseudorandom σ with the second coin toss phase is non-negligible then the commitments requested by the environment in the second coin toss phase are not computationally hiding.

More formally, we construct an algorithm that forces the environment to honest behavior in the second coin toss phase and compare the behavior of the environment used by the algorithm to the environment's behavior in the simulation.

The algorithm is constructed as follows:

- Generate two key pairs $(pk_0, td_0) \leftarrow Gen(1^n)$, $(pk_1, td_1) \leftarrow Gen(1^n)$, interpret the tuple (pk_0, pk_1) as a bit string of length $2k$ and execute the $2k$ coin tosses according to the simulation.

- Draw a random bit string σ of length $4n$ and execute $4n$ coin tosses sequentially in the following way:

  1. set a counter $l = 0$
  2. Denote σ^i the i-th bit of the bit string σ. Set in the i-th coin toss σ^i as the output of the coin toss.
  3. Draw a random bit b_1^i, compute a commitment $c_i = com(b_1^i)$ and simulate receiving c_i from the internal simulation of the $2k + i$-th instance of $\mathcal{F}_{\textbf{com-binding}}^{\textbf{post}}$.
  4. If the environment answers the commitment with a bit b_2^i for which $\sigma^i = b_1^i \oplus b_2^i$ holds, go to the coin toss of the next bit σ^{i+1} or if this is the $4n$-th coin toss, then proceed to the commit phase. If $\mathcal{Z}$ answers with a bit b_2^i that does not fulfill the condition stated above and if the counter fulfills $l < n$, then set $l = l + 1$, rewind the environment and go back to step 3. Else (this is the case if $s^i \neq b_1^i \oplus b_2^i$ and if $l \geq n$), abort the simulation.

- When getting a commitment y from $\mathcal{Z}$, check with the help of td_0, whether y lies in the image of G_{pk_0}. If the environment unveils the commitment y successfully to a bit $b' = 1$, then output 1, else output 0.

Let us now analyze the probability of the environment in succeeding to equivocate a commitment and compare the output of the environment in the algorithm to the output of the environment in the simulation.

If the environment behaves honestly during the second coin toss phase (i. e. it does not try to bias the outcome of the second coin toss phase), then the probability that the algorithm aborts the simulation is only negligible (compared to the ideal-world simulation) since the second coin toss phase is aborted with only negligible probability by the reduction algorithm and the resulting string σ is a uniformly random one. Then the probability that $\mathcal{Z}$ is able to equivocate the commitment is only negligible since σ has to be of form $\sigma = G_{pk_0}(r_0) \oplus G_{pk_1}(r_1)$, which happens only with negligible probability for a uniformly random σ.

If the environment was able to bias the outcome of second coin toss phase to a chosen bit string σ of the form $\sigma = G_{pk_0}(r_0) \oplus G_{pk_1}(r_1)$ (for which r_0 and r_1 were chosen by $\mathcal{Z}$) with non-negligible probability, then there is with non-negligible probability a difference between the output of the environment in the ideal-world simulation and $\mathcal{Z}$'s output in the reduction emulation: The ideal-world simulation is aborted in the unveil phase, whereas the reduction algorithm aborts its emulation with non-negligible probability during the second coin toss phase. This happens since at least one bit of the string σ chosen by the reduction algorithm differs from the string σ' chosen by the environment. As a conclusion the reduction algorithm rewinds at least one coin toss n times and then aborts the emulation since the bit $b_2^{i\prime}$ in each rewinding of the coin toss does not match to the bit $b_2^i = \sigma^i \oplus b_1^i$ needed by the reduction algorithm to generate its randomly chosen σ. This in turn means that the environment was able to extract the committed bit out of the commitment received by the non-information oracle to bias at least one coin toss outcome and therefore the commitment used by the non-information oracle is not hiding, which is a contradiction to the security of protocol $\Pi_{\mathbf{ct-bit}}$ shown in [Lin17], Theorem 6.2.7.

Therefore the distribution of σs generated in the ideal-world (and in the real world) is statistically close to the distribution of σs in the reduction algorithm. Conclusively, the random string σ generated in the second coin toss phase is pseudorandom only with negligible probability.

We conclude that in the case of a corrupted committer, the commitment scheme UCC-OneTime-CT securely $(\mathcal{L},\mathcal{M})$-$\mathcal{F}^{\mathbf{post}}$ realizes $\mathcal{F}_{\mathbf{com}}$.

Next, we consider the case when the recipient is corrupted.

The recipient R is corrupted and the committer C is honest Without loss of generality we again can assume the dummy adversary. The simulator $\mathcal{S}_R$ which simulates corrupting the recipient is constructed as follows:

- Generate $2k + 4n$ random CRS crs_1, ..., $crs_{2+4n} \in C$ and simulate receiving them from the first and second instance of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$.
- Simulate the $2k$ coin tosses for (pk_0, pk_1) sequentially in the following way:
  - Simulate receiving the output `(committed)` from the first instance of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$. When receiving the request to the non-information oracle, draw a random bit b_i and generate a commitment $c_1^i(b_i)$ using crs_i and simulate receiving c_1^i from the internal simulation of the non-information oracle.
  - When receiving a random bit s_2^i, simulate sending it to the internal simulation of C.
  - Simulate receiving the unveil information to c_1^i from the internal simulation of the first instance of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$.
- Set (pk_0, pk_1) to the concatenation of s^1 to s^k, i. e. $(pk_0, pk_1) = s^1 \cdot \ldots \cdot s^k$
- Draw random strings $r_1, r_2 \leftarrow \{0,1\}^n$ and compute $\sigma = G_{pk_0}(r_0) \oplus G_{pk_1}(r_2)$
- Simulate the $4n$ coin tosses for $\sigma = \sigma^1 \cdot \ldots \cdot \sigma^{4n}$ sequentially in the following way:
  - When getting the message `(commit,` t_1^i`)` as input to the second instance of functionality $\mathcal{F}_{\text{com-binding}}^{\text{post}}$, simulate sending it to the internal simulation of the second instance of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$.
  - Compute $t_2^i = t_1^i \oplus \sigma^i$ and simulate receiving t_2^i from the internal simulation of C.
  - When getting the message `(unveil)` as input to the second instance of functionality $\mathcal{F}_{\text{com-binding}}^{\text{post}}$, simulate sending it to the internal simulation of the second instance of $\mathcal{F}_{\text{com-binding}}^{\text{post}}$.
- Compute the commitment $y = G_{pk_0}(r_0)$ and simulate receiving it from the internal simulation of C.
- When getting the committed bit b from $\mathcal{F}_{\text{com}}$ in the reveal phase, simulate receiving the unveil information b, r_b from the internal simulation of C.

The coin tosses generating the public keys are done in an honest manner. Since the outcome of the first phase of coin tosses is a randomly chosen tuple of public keys for two pseudorandom generators, the coin tosses generating the public keys in the ideal world are identically distributed to the first type of coin tosses done in the real world.

The coin tosses for generating σ are also computationally indistinguishable from the real-world coin tosses generating σ: Since in the simulation the string σ is generated using pseudorandom generators, the resulting string is pseudorandom and therefore computationally indistinguishable from a truly random value.

Now we have to show that the simulated commit phase and unveil phase of the commitment scheme is computationally indistinguishable from the real-world commit and unveil phase.

Therefore we assume for the sake of contradiction that there exists an environment that is able to distinguish the real world from the ideal world. Then we can construct an algorithm that is able to distinguish between a pseudorandom value and a truly random one.

The algorithm gets as inputs a public key pk for a pseudorandom generator G_{pk}, security parameter n and a string z that is either truly random or a pseudorandom value generated using G_{pk}.

The constructed distinguishing algorithm works as follows:

- Choose a random bit $c \leftarrow \{0,1\}$ (which is our guess that the honest committer will commit to), set $pk_{1-c} = pk$, generate another key pair $(pk_c, td_c) \leftarrow Gen(1^n)$.
- Run the ideal-world simulation of the commitment scheme with the following modifications:
- Interpret the tuple (pk_0, pk_1) as bit string s of length $|(pk_0, pk_1)| = 2k$ and run $2k$ coin tosses sequentially:
  1. set a counter $l = 0$
  2. Denote s^i the i-th bit of the bit string s. Set in the i-th coin toss s^i as the output of the coin toss.
  3. Generate a random bit b_1^i, compute a commitment $c_i = com(b_1^i)$ and simulate receiving c_i from the internal simulation of C.
  4. If the environment answers the commitment with a value b_2^i for which $s^i = b_1^i \oplus b_2^i$ holds, go to the coin toss of the next bit or if this is the k-th coin toss and the check succeeds, then proceed to the first coin toss of generation of σ. If $\mathcal{Z}$ answers with a b_2^i that does not fulfill the condition stated above and if the counter $l < n$, then set $l = l + 1$, rewind the environment and go back to step 3. Else (this is the case if $s^i \neq b_1^i \oplus b_2^i$ and if $l \geq n$), abort the simulation.
- Draw a random string $r_c \in \{0,1\}^n$, set the string σ to $\sigma = G_{pk_c}(r_c) \oplus z$, interpret σ as a bit string of length $4n$ and run $4n$ coin tosses according to the ideal world simulation.

- Generate a commitment $y = G_{pk_c}(r_c)$ if $c = 0$ or $y = G_{pk_c}(r_c) \oplus \sigma$ if $c = 1$ and simulate receiving y from the internal simulation of C.
- If in the unveil phase the functionality $\mathcal{F}_{\mathbf{com}}$ sends a bit $b \neq c$ (i.e. our guess was wrong), then abort the simulation. Else simulate receiving the unveil information $b = c, r$ to y from the internal simulation of C and output whatever $\mathcal{Z}$ outputs.

First of all, since the underlying commitments for the coin toss are computationally hiding, the environment is not able to bias the outcome of the first coin toss phase and the distinguisher aborts the simulation during the coin toss phase only with a negligible probability compared to the ideal-world simulation. The proof for this can be found in the proof for theorem 6.7.2 in [Lin17]. Also, since the distinguisher draws two uniformly random keys $pk_0, pk_1 \in \mathcal{D} = \{0,1\}^k$ by using the generation algorithm Gen, the outcome of the first coin toss phase is equally distributed as the outcome of the first coin toss phase in the ideal-world simulation.

Let us now analyze the advantage of our algorithm in determining whether z is truly random or pseudorandom.

If z is a truly random $4n$ bit string (i. e. it is uniformly distributed), then σ is also truly random and therefore σ hides the committed bit c information-theoretically from $\mathcal{Z}$ at the beginning of the simulation. The probability that the distinguisher aborts the simulation during the unveil phase is $1/2$ plus some negligible proportion (conditioned on the event that the simulation is not aborted during the coin toss phase).

Conditioning on the event that the simulation is not aborted, the output of the simulation is identically distributed as $\mathcal{Z}$'s output in the real-world execution (since the random string σ is truly random and the public keys for the pseudorandom generators are valid keys).

Now let us look at the case if z is generated using G_{pk}, i. e. $z = G_{pk_1}(r_1)$ for some random $r_1 \in \{0,1\}^n$. Then $\sigma = G_{pk_0}(r_0) \oplus z = G_{pk_0}(r_0) \oplus G_{pk_1}(r_1)$ is a pseudorandom value and the fake string σ also hides the committed bit c information-theoretically from $\mathcal{Z}$ at the beginning of the simulation. We conclude that the distinguisher also aborts the simulation with probability $1/2$ plus some negligible proportion since the environment has no chance in distinguishing a commitment on 0 from a commitment on 1. Again, we conclude that the output of the distinguisher is identically distributed as $\mathcal{Z}$'s output in the ideal-world simulation, conditioned on the event that the distinguishing algorithm does not abort the simulation.

As a result, if the environment's view is computationally distinguishable form $\mathcal{Z}$'s view in the ideal world then the probability of the distinguisher in distinguishing between a pseudorandom value and a truly random value is $\varepsilon(n)/2$. Since by assumption $\varepsilon(n)$ is non-negligible, $\varepsilon(n)/2$ is also non-negligible, contradicting the requirement that the pseudorandom generator generates pseudorandom values which are computationally indistinguishable from truly random values.

Both C and R are honest When both parties are honest, we want to show that the recipient generates outputs, showing that the protocol is non-trivial.

When the honest committer gets an input `(commit, x)`, it initiates the coin tosses to produce the CRS together with R. Then it generates the commitment y on its input b and sends it to R. The recipient then outputs upon receiving y the message `(committed)`. In the unveil phase, R receives the unveil information (b, r) to the commitment y from C and then outputs the message `(unveil, b)`. Since the recipient generates an output even if both parties are honest, the protocol is non-trivial.

Since we have shown that in both cases of corruption we can generate a simulator that communicates in such a way that the environment's output in the ideal world is computationally indistinguishable from the environment's output in the real world and have additionally shown that the protocol is non-trivial, this concludes the proof.

□

We conclude that as long as the adversary was not able to increase its computational power during the coin toss phase of the commitment scheme then it is neither able to equivocate the commitment nor to extract the committed value if it was not unveiled.

Remark 4.11 *In this work we did not state a proper composition theorem since proving the composition theorem was beyond the scope of this thesis. However, having a composition theorem, proving the composition of protocols is much easier than proving the composition without having the composition theorem.*

Since this security framework is based on the UC framework, it should be possible to state a proper composition theorem for the $\mathcal{F}^{\mathbf{post}}$ framework.

Using the UCC-OneTime commitment scheme we are able to create a $\mathcal{F}^{\mathbf{post}}$ secure coin toss based on the $\mathcal{F}^{\mathbf{post}}_{\mathbf{com-binding-od}}$-hybrid model: We have shown that the commitment scheme UCC-OneTime-CT is $\mathcal{F}^{\mathbf{post}}$ secure (and hence based on the new complexity assumption) and a UC secure coin toss is possible in the $\mathcal{F}_{\mathbf{com}}$-hybrid model. If we instantiate $\mathcal{F}_{\mathbf{com}}$ in the coin toss protocol by UCC-OneTime-CT, then we have a coin toss protocol in the $\mathcal{F}^{\mathbf{post}}_{\mathbf{com-binding-od}}$-hybrid model. If there was a composition theorem was stated for the $\mathcal{F}^{\mathbf{post}}$ framework we were able use the composition theorem and the statement would follow.

If at the beginning is clear how many commitments are generated with the updated commitment scheme then instantiating the CRS by using a coin toss in combination with the commitment scheme UCC-OneTime-CT may suffice. However, if one wishes to create commitments in a flexible way, then we have to use a coin toss in the $\mathcal{F}_{\mathbf{mcom}}$-hybrid model, the multi-use extension of $\mathcal{F}_{\mathbf{com}}$. In Chapter 5 we explain how such a multi-use commitment scheme is instantiated using a $\mathcal{F}^{\mathbf{post}}$ secure coin toss.

Protocol UCC-OneTime-CT

- Generating the CRS
  - Both C and R receive $4n + 2k$ CRSs $crs_1, ..., crs_{4n+2k} \in C$ from $4n + 2k$ instances of $\mathcal{F}^{\textbf{post}}_{\textbf{com-binding-od}}$.
  - Execute the coin toss protocol $\Pi_{\text{CT}-bit}$ $2k$ times in the following way (let i be the counter variable for the number of the current coin toss):
    - C sends (commit, s_1^i) to the first instance of $\mathcal{F}^{\textbf{post}}_{\textbf{com-binding-od}}$ to commit to s_1^i.
    - When getting (committed) from the i-th instance of $\mathcal{F}^{\textbf{post}}_{\textbf{com-binding-od}}$, R sends s_2^i to C.
    - C then sends (unveil) to the i-th instance of$\mathcal{F}^{\textbf{post}}_{\textbf{com-binding-od}}$ to send the unveil information to R and computes $s^i = s_1^i \oplus s_2^i$.
    - When getting the unveil information to the commitment from the i-th instance of $\mathcal{F}^{\textbf{post}}_{\textbf{com-binding}}$, R computes $s^i = s_1^i \oplus s_2^i$.
  - Both parties concatenate the outcomes of all $2k$ coin tosses (i. e they compute $(pk_0, pk_1) = s^1 \cdot ... \cdot s^{2k}$)
  - Execute the coin toss protocol $\Pi_{\text{CT}-bit}$ $4n$ times in the following way (let i be the counter variable for the number of the current coin toss):
    - R sends (commit, t_1^i) to the $k + i$-th instance of $\mathcal{F}^{\textbf{post}}_{\textbf{com-binding-od}}$ to commit to t_1^i.
    - When getting (committed) from the $k + i$ instance of $\mathcal{F}^{\textbf{post}}_{\textbf{com-binding-od}}$, C sends t_2^i to R.
    - R then sends (unveil) to the $k + i$-th instance of$\mathcal{F}^{\textbf{post}}_{\textbf{com-binding-od}}$ to send the unveil information to C and computes $t^i = t_1^i \oplus t_2^i$.
    - When getting the unveil information to the commitment from the $k + i$-th instance of $\mathcal{F}^{\textbf{post}}_{\textbf{com-binding-od}}$, C computes $t^i = t_1^i \oplus t_2^i$.
  - Both parties concatenate the outcomes of all $4n$ coin tosses (i. e. they compute $\sigma = t^1 \cdot ... \cdot t^{4n}$)
- commit phase
  - If the bit to commit is $b = 0$, C computes the commitment y as follows: $y = G_{pk_0}(r)$ for a randomly drawn $r \leftarrow \{0,1\}^n$, else it computes $y = G_{pk_1}(r) \oplus \sigma$. Then it sends y to R.
- reveal phase
  - C sends b and r to R. R checks whether $y = G_{pk_0}(r)$ if $b = 0$ or whether $y = G_{pk_1}(r) \oplus \sigma$ if $b = 1$. If the check succeeds, R outputs (unveil, b).

Figure 4.6: The protocol UCC-OneTime with incorporated coin toss.

5 Sampling the New CRS

In this chapter, we will show how a multi-use commitment has to be modified such that it can be used for tossing several CRS of the updated commitment scheme. In Section 5.1, we will define the properties the multi-use commitment needs to be used for tossing the new CRS and give an example how it can be instantiated. In Section 5.2 we show how the multi-use commitment can be used to toss the bits for a new CRS.

5.1 Defining the Multi-Use Commitment

When tossing the new CRS for an updated commitment scheme, one may not know how many commitments one will generate with the updated scheme. Therefore using a one-time usable bit commitment scheme like the UCC-OneTime commitment scheme is not sufficing for tossing the new CRS. Therefore we additionally have to use a multi-use commitment scheme where one instance of the protocol can be used several times to commit to different values.

For this we use an already existing multi-use UC-secure commitment scheme and instantiate its CRS by making several $\mathcal{F}^{\mathbf{post}}$ secure coin tosses in the $\mathcal{F}_{\mathbf{com}}$-hybrid model, one coin toss for each bit of the CRS.

To be able to instantiate the CRS of the commitment scheme with a $\mathcal{F}^{\mathbf{post}}$ secure coin toss the commitment scheme we require the commitment scheme to be able to have a uniformly random CRS. One example that fulfills this condition is the multi-use Commitment UAHC (for UC Adaptive Hamiltonian Commitment) defined in [Can+02] which is both computationally binding and hiding.

First we describe the commitment UAHC as defined in [Can+02], Section 5 and then shortly explain how the CRS of this commitment scheme has to be modified to be usable for sampling the new CRS of the updated commitment scheme.

The commitment scheme UAHC uses an underlying commitment scheme aHC (short for adaptive Hamiltonian Commitment scheme) based on Hamiltonian cycles and additionally uses two encryption schemes: an IND-CCA secure encryption scheme E_{cca} and an IND-CPA secure encryption scheme E. The underlying commitment aHC is a variant of the Feige-Shamir commitment [FS90] which is an equivocal commitment scheme and secure against adaptive

adversaries. This modified underlying commitment scheme aHC works as follows:

- Given the CRS y which is a value that lies within the image of a one-way function f (i. e. $y = f(x)$ for some random x), use the Cook-Levin NP-reduction on the language $\{y|\exists x$ such that $y = f(x)\}$ to that of Hamiltonicity to obtain a graph G

- To commit to the bit 0, the committer commits to a random permutation π of the graph G using an underlying commitment *Com*. For committing on 1, the committer commits to a random q-cycle for q being the number of vertices in G using the underlying commitment scheme *Com*.

- To unveil the commitment, the committer unveils the underlying commitments on the entire graph G plus the permutation π to R if the committed bit was 0, otherwise the committer unveils only the q-cycle of G.

For the extractability of the commitment, Canetti et. al added an IND-CCA encryption to encrypt the randomness used for the commitment aHC. Also, they added an IND-CPA secure encryption scheme which produces pseudorandom ciphertexts. Yet, having only the encryption of the randomness destroys the equivocality. Therefore [Can+02] added a random string of the same length as the encryption which is computationally indistinguishable from a real ciphertext to restore the equivocality. This requires the encryption scheme to be pseudorandom. Since there is no known IND-CCA secure encryption scheme that produces pseudorandom ciphertexts, Canetti et al. require the additional IND-CPA secure encryption scheme.

For an overview of the classic UAHC commitment scheme, we roughly sketch the commitment:

The CRS consist of a random value y which lies in the image of a one-way function f (this value is needed for the underlying commitment scheme aHC) together with a public key pk_{cca} for an IND-CCA secure encryption scheme and a public key pk for an IND-CPA secure encryption scheme which produces pseudorandom ciphertexts.

To commit to a bit, the committer uses the underlying commitment scheme aHC to commit to the input b and uses for this commitment $z = \text{aHC}(b)$ some random string r. Then C encrypts the randomness r using the IND-CCA secure scheme and additionally encrypts the encrypted randomness with the IND-CPA secure encryption scheme using some randomness s which yields in a pseudorandom ciphertext c_b. As a last step of the commit phase, C draws a random string c_{1-b} with length $|c_{1-b}| = |c_b|$ and sends the commitment z on b together with c_b and c_{1-b} to R.

In the unveil phase, C sends the unveil information r and b for the commitment together with the randomness s used for the encryption to R. R then checks whether the unveil information for the commitment z is valid and whether c_b is a valid encryption of r.

Since the CRS of our modified commitment scheme has to be uniformly random, the one-way function f has to produce pseudorandom values. This is since a random value y in the image of f is used in the CRS.

For the use of an IND-CCA encryption scheme, [Can+02] suggested a scheme based on enhanced trapdoor permutations with the property that each encryption has at most one valid decryption. This holds for almost any encryption scheme based on enhanced trapdoor permutations, especially the encryption scheme defined in [DDN00]. For the IND-CPA encryption scheme a scheme based on trapdoor permutations with hard-core predicates is suggested. We additionally require the the trapdoor schemes used for the encryption schemes to have dense public descriptions to ensure that the public keys can be drawn using a coin toss.

To instantiate the CRS of the multi-use commitment, we will use a coin toss in the $\mathcal{F}_{\mathbf{com}}$-hybrid model. This time we are able to use a $\mathcal{F}^{\mathbf{post}}$-secure coin toss since we can instantiate the coin toss using a $\mathcal{F}^{\mathbf{post}}$ secure commitment scheme in the $\mathcal{F}^{\mathbf{post}}_{\mathbf{com-binding-od}}$-hybrid model.

In more detail, the multi-use commitment is instantiated as follows:

First of all, we have to instantiate the $\mathcal{F}^{\mathbf{post}}$ secure coin tosses used for generating the CRS of the multi-use commitment. This is done by generating as many commitments as there are bits in the CRS of the commitment scheme. Then the generated commitments are used to securely execute bit-coin tosses. As the last step, the tossed bits will be used as the CRS for the commitment scheme.

The commit and the unveil phase remain unchanged compared to the original protocol.

The modified protocol can be seen in 5.1.

The modified commitment succeeds since we require the trapdoor permutation to have dense public keys (that are based on the stronger complexity assumption $\mathcal{M}$) and therefore the CRS can be uniformly distributed.

An easy way to show the existence of a $\mathcal{F}^{\mathbf{post}}$ secure commitment scheme in the $\mathcal{F}^{\mathbf{post}}_{\mathbf{com-binding-od}}$-hybrid model was to use the composition theorem. Unfortunately, stating a proper composition theorem for the $\mathcal{F}^{\mathbf{post}}$ framework was beyond the scope of this work. But to get an idea why the commitment scheme UAHC-CT is $\mathcal{F}^{\mathbf{post}}$ secure if the CRS is updates we assume that a proper composition was stated. Then we were able to informally prove the update of the CRS in the following way: Since the commitment scheme UCC-OneTime-CT is proven to be $\mathcal{F}^{\mathbf{post}}$ secure in the $\mathcal{F}^{\mathbf{post}}_{\mathbf{com-binding-od}}$-hybrid model and a $\mathcal{F}^{\mathbf{post}}$ secure coin toss is possible in the $\mathcal{F}_{\mathbf{com}}$-hybrid model, we were able to use the composition theorem to show that the coin toss protocol using

Protocol UAHC-CT

- Generating the CRS
  - Let $l = |s|$ denote the length of the CRS s used for this commitment scheme. Then both C and R send `(init)` to l instances of $\mathcal{F}_{\text{ct}}$ and receive each a random bit s_i from each instance of $\mathcal{F}_{\text{ct}}$.
  - Both C and R concatenate all l bits $s = s_1 \cdot \ldots \cdot s_l$ to compute the CRS s.
  - Both parties split the string s into a triple (y, pk, pk_{cca}), where:
    - y is a random image of the one-way function f
    - pk is a public key for an IND-CPA secure encryption scheme E
    - pk_{cca} is a public key for an IND-CCA secure encryption scheme E_{cca}
- commit phase on bit b
  - On input b C computes:
    - $z = aHC(b, r)$ for random string $r \in \{0,1\}^n$
    - a ciphertext $c_b \leftarrow \mathbf{E}(E_{cca}(r))$ with randomness s
    - a random string c_{1-b} of length $|c_{1-b}| = |c_b|$
  - C sends (z, c_0, c_1) to R.
  - Upon receiving, R outputs `(Received)`
- reveal phase
  - C sends `(Dec,` b, r, s`)` to R
  - Upon receiving, R checks whether $y \stackrel{?}{=} aHC(b, r)$, $c_b = E(E_{cca}(r))$ under randomness s. If the verification succeeds, R outputs `Open,` b

Figure 5.1: The protocol UAHC-CT in the $\mathcal{F}_{\text{ct}}$-hybrid model, compare [Can+02], Figure 5.

UCC-OneTime-CT is a $\mathcal{F}^{\text{post}}$ secure coin toss protocol in the $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$-hybrid model. Then, by instantiating $\mathcal{F}_{\text{ct}}$ in the commitment scheme UAHC-CT by the composition of the coin toss with the commitment scheme UCC-OneTime-CT, we were able to show that there existed a protocol that $\mathcal{F}^{\text{post}}$-realizes $\mathcal{F}_{\text{mcom}}$ in the $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$-hybrid model.

Since we assume the UCC-OneTime-CT commitment scheme to be $\mathcal{F}^{\text{post}}$-secure, the resulting multi-use commitment scheme UAHC-CT is $\mathcal{F}^{\text{post}}$ secure.

5.2 The Final Step: Sampling the New CRS

Since we have now a commitment scheme we can use several times, we are finally able to sample a new CRS.

Since the new CRS is sampled using a $\mathcal{F}^{\mathbf{post}}$ secure coin toss, we require the CRS of the commitment scheme to be uniformly random.

The generation of the new CRS is done as follows:

For each bit of the new CRS execute one coin toss in the $\mathcal{F}_{\mathbf{mcom}}$-hybrid model, which is the multi-use extension of $\mathcal{F}_{\mathbf{com}}$. For instantiating the functionality $\mathcal{F}_{\mathbf{mcom}}$, one can use the suggested multi-use bit commitment protocol UAHC-CT described in Figure 5.1. Then concatenate all outcomes of the coin tosses to a string s which is the new resulting CRS for the commitment scheme to be updated.

We do not give a formal proof for the security of sampling the new CRS but rather an intuition why the sampling results in a secure CRS. Therefore we again would like to use the composition theorem. Since stating the composition theorem was beyond the scope of this work, we assume that there was such a composition theorem:

Since we require the new CRS based on the stronger complexity assumption to be uniformly random, we are able to sample it using a $\mathcal{F}^{\mathbf{post}}$ secure coin toss protocol. Since the protocol UAHC-CT is based on the stronger complexity assumption and due to the (currently not existing) composition theorem, the coin toss protocol using UAHC-CT then could also proven to be $\mathcal{F}^{\mathbf{post}}$ secure.

6 Conclusion

In this thesis we have seen how one can update a common reference string for a commitment. Therefore one can switch the setup assumption for the CRS.

We have defined two functionalities for commitments and showed that using a modification of one of the functionalities, one can construct a $\mathcal{F}^{\text{post}}$ secure commitment in the $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$-hybrid model that is usable for a coin toss for tossing the bits of the new CRS based on the stronger setup assumption.

We also have seen that a $\mathcal{F}^{\text{post}}$ secure coin toss is neither possible in the $\mathcal{F}^{\text{post}}_{\text{com-binding}}$-hybrid model nor in the $\mathcal{F}^{\text{post}}_{\text{com-hiding}}$-hybrid model. Yet, we have seen that we can construct a stand-alone secure coin toss that uses a modification of $\mathcal{F}^{\text{post}}_{\text{com-binding}}$.

We also have shown which type of commitments can be used for tossing the new CRS in the $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$-hybrid model. Those commitment schemes have a uniformly random CRS to ensure that its CRS can be generated using a coin toss protocol. We also require the commitments to have a CRS that can be split in two parts such that the generation of the CRS using the coin toss protocol is simulatable. To be able to create multiple commitments with the updated commitment using a new CRS, we also have modified a multi-use commitment such that it can be used in combination with the single-use commitment in the $\mathcal{F}^{\text{post}}_{\text{com-binding-od}}$-hybrid model.

We conclude that a CRS can be updated using our construction if it is uniformly random.

Some possible future work may include:

- This work was not focused on efficiency, therefore the update takes several steps. An idea for further work therefore can be to optimize the update process by finding more efficient protocols or by finding a multi-use commitment protocol that can even replace the modified UCC-OneTime-CT protocol such that the amount of steps in the update process can be shortened.

- The definition of a composition theorem was out of the scope of this thesis. Some future work might be concentrated on defining a proper composition theorem for the $\mathcal{F}^{\text{post}}$ framework.

- We were not able to find a combination of enhanced trapdoor permutations with dense public keys such that the probability is high enough to pick randomly a secure public key. Some further work may also concentrate on finding a solution for this.
- Also, we have focused on static adversaries. For security against adaptive adversaries, some functionalities and protocols have to be modified.
- For further work one may consider how other protocols may be updated.

Bibliography

[Ben+15] Fabrice Benhamouda et al. "Efficient Zero-Knowledge Proofs for Commitments from Learning with Errors over Rings." In: *ESORICS 2015: 20th European Symposium on Research in Computer Security, Part I*. Ed. by Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl. Vol. 9326. Lecture Notes in Computer Science. Vienna, Austria: Springer, Heidelberg, Germany, Sept. 2015, pp. 305–325.

[BHY09] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. "Possibility and Impossibility Results for Encryption and Commitment Secure under Selective Opening." In: *Advances in Cryptology – EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. Lecture Notes in Computer Science. Cologne, Germany: Springer, Heidelberg, Germany, Apr. 2009, pp. 1–35.

[Blu81] Manuel Blum. "Coin Flipping by Telephone." In: *Advances in Cryptology – CRYPTO'81*. Ed. by Allen Gersho. Vol. ECE Report 82-04. Santa Barbara, CA, USA: U.C. Santa Barbara, Dept. of Elec. and Computer Eng., 1981, pp. 11–15.

[BM84] Manuel Blum and Silvio Micali. "How to Generate Cryptographically Strong Sequences of Pseudorandom Bits." In: *SIAM Journal on Computing* 13.4 (1984), pp. 850–864.

[BM92] Mihir Bellare and Silvio Micali. "How to Sign Given Any Trapdoor Function." In: *Journal of the ACM* 39.1 (1992), pp. 214–233.

[BMM21] Brandon Broadnax, Jeremias Mechler, and Jön Müller-Quade. *Environmentally Friendly Composable Multi-Party Computation in the Plain Model from Standard (Timed) Assumptions*. 2021.

[BN00] Dan Boneh and Moni Naor. "Timed Commitments." In: *2000*. Vol. 1880. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2000, pp. 236–254.

[Can+02] Ran Canetti et al. *Universally Composable Two-Party and Multi-Party Secure Computation*. Cryptology ePrint Archive, Report 2002/140. `http://eprint.iacr.org/2002/140`. 2002.

[Can00] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. Cryptology ePrint Archive, Report 2000/067. `http://eprint.iacr.org/2000/067`. 2000.

[CF01] Ran Canetti and Marc Fischlin. "Universally Composable Commitments." In: *Advances in Cryptology – CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Düsseldorf, Germany: Springer, Heidelberg, Germany, Aug. 2001, pp. 19–40.

[CK02] Ran Canetti and Hugo Krawczyk. "Universally Composable Notions of Key Exchange and Secure Channels." In: *Advances in Cryptology – EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, Heidelberg, Germany, Apr. 2002, pp. 337–351.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. "Nonmalleable Cryptography." In: *SIAM Journal on Computing* 30.2 (2000), pp. 391–437.

[DN02] Ivan Damgård and Jesper Buus Nielsen. "Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor." In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2002, pp. 581–596.

[DP92] Alfredo De Santis and Giuseppe Persiano. "Zero-Knowledge Proofs of Knowledge Without Interaction (Extended Abstract)." In: *33rd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Oct. 1992, pp. 427–436.

[FS90] Uriel Feige and Adi Shamir. "Zero Knowledge Proofs of Knowledge in Two Rounds." In: *Advances in Cryptology – CRYPTO '89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1990, pp. 526–544.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Vol. 1. Cambridge, UK: Cambridge University Press, 2001, pp. xix + 372.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Vol. 2. Cambridge, UK: Cambridge University Press, 2004.

[GR13] Oded Goldreich and Ron D. Rothblum. "Enhancements of Trapdoor Permutations." In: *Journal of Cryptology* 26.3 (July 2013), pp. 484–512.

[Lin17] Yehuda Lindell. "How To Simulate It - A Tutorial on the Simulation Proof Technique." In: *Tutorials on the Foundations of Cryptography*. Ed. by Yehuda Lindell. Information Security and Cryptography. Springer, Heidelberg, Germany, 2017, pp. 277–346.

[MU10] Jörn Müller-Quade and Dominique Unruh. "Long-Term Security and Universal Composability." In: *Journal of Cryptology* 23.4 (Oct. 2010), pp. 594–671.

[Yao82] Andrew Chi-Chih Yao. "Theory and Applications of Trapdoor Functions (Extended Abstract)." In: *23rd*. Chicago, Illinois: IEEE Computer Society Press, Nov. 1982, pp. 80–91.

List of Tables

List of Figures

List of Theorems

Listings