# Trapdoor Commitment Schemes and Their Applications

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich Mathematik
der Johann Wolfgang Goethe-Universität
in Frankfurt am Main

—

von
Marc Fischlin
aus Offenbach am Main

—

Frankfurt am Main 2001
D F 1

Vom Fachbereich Mathematik der Johann Wolfgang Goethe-Universität
als Dissertation angenommen.

| | |
|---|---|
| Dekan: | Dr. J. Baumeister |
| Gutachter: | Dr. C. Schnorr<br>Dr. M. Sieveking |
| Datum der Disputation: | 17. Dezember 2001 |

*Look, matey,*
*I know a dead parrot when I see one,*
*and I'm looking at one right now.*

— Monty Python, The Parrot Sketch

# Preface

There are certainly one or two things about cryptography I have learned during my Ph.D. time. One thing I have noticed is that trapdoor commitments are a remarkable catalyst for the design of provably secure cryptographic protocols. The thesis reflects this. It is the result of my Ph.D. time at Professor Schnorr's group at the Johann Wolfgang Goethe-University of Frankfurt, Germany, from July 1997 till December 2001, suspended due to my community service from March 1998 till April 1999.

### Introduction

Informally, commitment schemes can be described by lockable steely boxes. In the commitment phase, the sender puts a message into the box, locks the box and hands it over to the receiver. On one hand, the receiver does not learn anything about the message. On the other hand, the sender cannot change the message in the box anymore. In the decommitment phase the sender gives the receiver the key, and the receiver then opens the box and retrieves the message. One application of such schemes are digital auctions where each participant places his secret bid into a box and submits it to the auctioneer.

In this thesis we investigate trapdoor commitment schemes. Following the abstract viewpoint of lockable boxes, a trapdoor commitment is a box with a tiny secret door. If someone knows the secret door, then this person is still able to change the committed message in the box, even after the commitment phase. Such trapdoors turn out to be very useful for the design of secure cryptographic protocols involving commitment schemes.

### Overview

In the first part of the thesis, we formally introduce trapdoor commitments and extend the notion to identity-based trapdoors, where trapdoors can only be used in connection with certain identities. We then recall the most popular constructions of ordinary trapdoor protocols and present new solutions for identity-based trapdoors.

In the second part of the thesis, we show the usefulness of trapdoors in commitment schemes. Deploying trapdoors we construct efficient non-malleable commitment schemes which basically guarantee indepency of commitments. Furthermore, applying (identity-based) trapdoor commitments we secure well-known identification protocols against a new kind of attack. And finally, by means of trapdoors, we show how to construct composable commitment schemes that can be securely executed as subprotocols within complex protocols.

### About This Thesis

The first part of the thesis mainly uses known facts to guide the reader to trapdoor commitments. Still, we also introduce the new notion of identity-based trapdoor commitments and present previously unpublished constructions of such trapdoors. The second part, exemplifying how to apply trapdoor commitments in order to devise secure protocols, is based on three papers done during my Ph.D. time: Chapter 4 about non-malleable commitment schemes is taken from a joint work with Roger Fischlin [FF00], published at Crypto 2000. The secure resettable identification part, Chapter 5, is extracted from a joint paper [BFGM01] with Mihir Bellare, Shafi Goldwasser and Silvio Micali presented at Eurocrypt 2001. The part here has been added to the earlier proposal of Bellare, Goldwasser and Micali how to achieve secure resettable identification with other techniques. Finally, Chapter 6 about universally composable commitments is almost a verbatim copy of the extended version of a paper with Ran Canetti [CF01] appearing at Crypto 2001.

The other papers published during (or before) my Ph.D. time do not resurrect in this thesis. Most of the papers deal with the design of efficient cryptographic protocols [F97a, F97b, F99, F01a, FF02] and some discuss more theoretical stuff [F97c, F01b, F00, F02]. Although [FF02] partly deals with trapdoor commitments, too, due to space reasons only the construction of a trapdoor commitment scheme in that paper is briefly presented here.

### Acknowledgments

A lot of people have contributed to this thesis, either directly or more subliminal. On the indirect side, I have to mention especially those people that kept bugging me by asking when I would finish this thesis. You finally made me do

it. Although they did not give scientific support I owe Stefanie, my parents, my brother Roger and all my friends a big thank-you.

During my Ph.D. years, I had the opportunity to attend quite a few conferences and workshops all over world. The places include countries like the United States, Finnland or Australia. For example, parts of the introduction were written at a Starbucks in Kyoto, Japan, in December 2000 (sorry guys, but the Caramel Frappuccino in Manhatten is much better). I also spent an amazing time in New York in July/August 2000, visiting the crypto group at IBM T.J. Watson; thanks to Alon, Eyal, Nick, Ran, Rosario, Shai and Tal. I am grateful to all people and organizations supporting these trips, the University of Frankfurt, Professor Claus Schnorr, IBM, IACR and all the conference chairs and workshop organizers, the Willkomm and the Leibniz foundation and anyone else supporting me.

I got in contact with a lot of researchers supporting me, sometimes it even ended up in a joint work. I would like to thank (in alphabetical order): Mihir Bellare, Stefan Brands, Ran Canetti, Cynthia Dwork, Roger Fischlin, Rosario Gennaro, Shafi Goldwasser, Shai Halevi, Yehuda Lindell, Silvio Micali, Daniele Micciancio, Tal Rabin, Omer Reingold, Alon Rosen, Dan Simon, Claus Schnorr, Berry Schoenmakers, all anonymous reviewers, and anyone else I intended to mention but forgot.

Finally, I thank Claus Schnorr and Malte Sieveking for discussions and remarks to preliminary drafts of this thesis. I am especially grateful to Professor Sieveking for suggesting the translations for the German summery, and for inspiring me how to write that summery. In addition to Professor Schnorr and Professor Sieveking, I thank Professor Kersting and Professor Wolfart for serving on the committee of my defense.

Enjoy.

Marc Fischlin December 2001

# Contents

*Chapter 1*

# Introduction

The classical issue in cryptography is encryption: how do I privately send a message to another person? At the present time we indeed know satisfactory solutions for this challenging task. But modern cryptography also supplies reliable constructions to many other areas, like digital signatures, identification, message authentication, electronic cash, electronic voting etc. Among others, commitment schemes make up an important building block for these solutions.

## 1. Commitment Schemes

Instructively, one can describe a commitment scheme with a lockable steely box. In the so-called commitment phase, one party, the sender, puts a message into a box, locks the box and gives it to the other party, the receiver. On one hand, the receiver cannot open the box to learn the message, and on the other side, the sender cannot change the message anymore. The former property is called secrecy, the latter unambiguity or binding property. In the decommitment phase, the sender gives the key to the receiver to open the box and to disclose the message.

An obvious application of commitment schemes are sealed-bid auctions. Each participant puts his bid into his box and submits the box to the auctioneer. After having collected all bids the auctioneer requests the keys from the participants, opens the boxes publicly and announces the winner. The important aspects of commitment schemes, secrecy and unambiguity, are reflected in this example: the actual bid should be kept secret until the bidding phase is over, and no bidder should be able to change his value after seeing a previously disclosed opponent's bid.

The auction case reveals another, more subtle requirement a commitment schemes must have in this setting. This requirement is not covered by secrecy and unambiguity and is not immediate if one visualizes commitment schemes as solid boxes. Namely, it should be infeasible for a bidder to "build" an appropriate box containing a bid $b + 1$ after seeing the locked box with an unknown bid $b$

of another participant. While this seems to be irrational in the context of steely boxes, it is a real threat if we implement commitment schemes digitally: the box corresponds to a bit string which —unlike hardware— can be easily copied and, for most known commitment schemes today, the encapsulated value can be incremented by external modification of the bit string. One part of this thesis presents efficient constructions of such non-malleable commitment schemes withstanding transformations. We will occassionally return to this motivating example in the course of introduction.

## 2. Trapdoor Commitment Schemes

In this thesis we investigate trapdoor commitment schemes. These are commitment schemes for which knowledge of a special information, the trapdoor, allows to overcome the binding property and to open a commitment ambiguously. At first glance this may be surprising: one of the important aspects of commitments is that they cannot be opened ambiguously, yet the trapdoor enables to bypass this. But we stress that ambiguous decommitments are only possible given this special information; without, a commitment is still solidly binding.

We explain the efficacy of trapdoors in commitment schemes on the basis of non-malleable commitments. Roughly, a commitment scheme is non-malleable if giving the adversary the original commitment of the honest party does not significantly increase his success probability of finding a commitment of a related message (e.g., a higher bid), compared to the case that the adversary does not have access to the honest party's commitment at all. Intuitively, the setting where the adversary does not get to see the other commitment describes the highest security level we can expect: the adversary's choice is made independently of the original message. Non-malleability now demands that the commitment scheme meets this high standard and thus provides indepency of commitments. In particular, it follows that a non-malleable commitment is transformation-resilient.

We next outline how to construct non-malleable commitments using trapdoors. Consider the auction case again, where an honest sender submits a bid to the auctioneer and the adversary's goal is to overbid this party by sending a higher value to the auctioneer. More precisely, the adversary first sees the commitment of the other sender and is supposed to output his commitment to a higher bid afterwards.

Assume that the honest sender's commitment contains a trapdoor but the adversary's commitment does not. Then, on one hand, the honest party's bid can still be altered by the trapdoor property in principle, even after the adversary has submitted his value. On the other hand, the adversary's commitment does not have a trapdoor and his value thenceforth is pinned down due to the binding property. Specifically, performing a gedankenexperiment, suppose that we hold the trapdoor information. Then we may give the adversary a fake commitment

to 0 on behalf of the honest party in the first place, and seclusively change this commitment to the actual value of the honest sender after the adversary has irrevocably committed to his bid. In this case, at the moment when the adversary prepares his binding commitment and unambiguously decides upon his bid, the only information available to him is the redundant commitment to 0. This means that the adversary's success probability cannot depend on the sender's initial commitment, as required for non-malleability.

Note, however, that we only perform a gedankenexperiment. But from the adversary's point of view the experiment and an actual attack are indistinguishable, because it is imperceptible for the adversary that we initially set the value to 0 and change it later via the trapdoor. Hence, the adversary's success probability in a real attack is the same as in the experiment, and so the fiction that the adversary's success probability is independent of the original commitment becomes real.

The problem with the approach above is that the honest party and the adversary usually use the same brand of box, say, the one the auctioneer announces for sending in commitments to the auction. Hence, either the box of the honest sender and the one of the adversary include a trapdoor, or neither one does. But then the argument above that the adversary irrevocably commits given only an alterable dummy commitment is no longer valid. The remedy is to run the protocol with tailor-made individual boxes. Then we may pass the sender a box containing a secret trapdoor, whilst the adversary receives a solidly binding box, and the aforementioned idea works again.

In other words, we seek a trapdoor commitment scheme where the trapdoor can only be used in connection with a certain identity, for instance, with the unique IP address of the computer of the honest bidder. Therefore, we introduce the notion of identity-based trapdoor commitments in this thesis, a refinement of ordinary trapdoor protocols, and examine how to construct and where to apply such identity-based trapdoors. For instance, our constructions of efficient non-malleable commitments apply both kinds of trapdoors simultaneously.

It would be too presumptuous to believe that we only benefit from trapdoors in commitment schemes. There are also disadvantages. Usually, trapdoor commitment schemes are less efficient than ordinary ones, or require the help of a trusted third party in the protocol, or need specific assumptions like the RSA assumption instead of more general ones like the existence of arbitrary hard-to-invert functions. Still, trapdoors facilitate the design of protocols or even make solutions possible at all. Thus, the additional requirements for trapdoor commitments sometimes pay off.

## 3. Trapdoor Commitments in Cryptography

Trapdoors in commitment protocols have already been considered and constructed in the past; they are also called equivocable commitment schemes or chameleon blobs in the literature. We recall some of the areas in which they are deployed.

One important field of applications of trapdoor commitments are zero-knowledge proofs. A zero-knowledge proof is a protocol between two parties, the prover and the verifier, in which the prover tries to convince the verifier that a certain statement is true or that he knows a secret to some public information. The verifier should be sure that the prover can only convince him if the statement is indeed true or if the prover really knows the secret. This is called soundness. On the other side, the prover's privacy should guarantee that nothing beyond the truth of the statement or the fact that he possesses a secret is revealed, e.g., the prover's secret itself should not be disclosed. We say the protocol is zero-knowledge.

Trapdoor commitment schemes have been used to construct zero-knowledge proofs [BCC88], there under the name chameleon blobs, constant-round zero-knowledge proofs in which the prover and verifier exchange only a constant number of messages [FS89, BCY91, BMO90], concurrent zero-knowledge protocols where the verifier talks to several instances of the prover in parallel [DO99, D00] and resettable zero-knowledge [CGGM00] where the verifier is even allowed to reset the prover to some previous step of the protocol. Similarly, Bellare et al. [BFGM01] realize secure resettable identification protocols via trapdoor commitments (although these identification protocols are not known to be zero-knowledge) —see Chapter 5 for details. Moreover, trapdoor commitments give rise to communication-efficient zero-knowledge protocols [CD97].

Trapdoor commitments also have an important impact on the design of non-malleable commitment protocols. As mentioned before, basically, for non-malleable commitment schemes one cannot change a commitment's content by external modifications. Trapdoors have been introduced in this context by Di Crescenzo et al. [DIO98] under the terminology of equivocable commitments. Subsequently, Fischlin and Fischlin [FF00, FF02] (see also Chapter 4) and Di Crescenzo et al. [DKOS01] applied number-theoretic constructions of trapdoor commitments to derive more efficient non-malleable systems.

Additionally, trapdoor commitments play an important role for the construction of secure signature schemes. They have been helpful in the design of secure signature schemes without relying on the strong random oracle assumption [GHR99, CS00]. Also, they turn out to be quite useful for the construction of secure undeniable signatures [KR99] where signature verification is only possible with the active help of the signer but such that the signer cannot deny a valid signature. Shamir and Tauman [ST01] deploy trapdoor commitments for signature

schemes where most of the work can be done off-line, i.e., before the message to be signed is known.

Further applications of trapdoor commitments include the design of secure multi-party computations with low communication complexity [CDN01]. Jakobsson et al. [JIS96] apply trapdoor commitments to derive designated verifier proof systems in which the prover shows to the verifier the truth of a statement, but such that the verifier cannot use this given proof to convince another person of the validity of the statement. Finally, Canetti and Fischlin [CF01] as well as Damgård and Nielsen [DN01] construct universally composable commitments, i.e., commitments which can be securely composed with other secure protocols, by means of trapdoor commitments (cf. Chapter 6).

## 4. Organization of Thesis

In Chapter 2 we introduce the basics; our approach is strongly influenced by Goldreich's book [G98]. In Section 1 of the chapter we settle some standard notations. Section 2 deals with basic cryptographic definitions. Among others, we define one-way functions and discuss standard assumptions related to RSA, factoring and the discrete logarithm problem. Further cryptographic primitives are discussed in the course of this thesis when presenting the corresponding protocols. In Section 3 we take the first step towards defining commitment protocols by introducing interactive protocols. The notation and formalization is then applied in the main section of this chapter, Section 4, defining commitment schemes rigorously. Given this, trapdoor commitment schemes are then quite straightforward to define, as done in Section 5. The concluding Section 6 of this chapter deals with the notion of identity-based trapdoor commitments for which the trapdoor is linked to a special identifier, e.g., a unique IP address of a computer. Even with knowledge of this trapdoor information, commitments related to other identifiers (e.g., to other IP addresses) are still binding.

We usually pursue a stepwise approach to formal definitions by gradually adding insight to the topic. Yet, Chapter 2 is still heavily loaded with complex technical parts and details. Readers who are primarily interested in grasping the idea of trapdoor commitment schemes may only study Sections 1 and 2 about the basics as well as the short introduction to commitment schemes in Section 4.1 and then proceed to Chapter 3 describing examples of trapdoor commitment protocols.

We present the presumably most popular constructions of trapdoor commitment schemes in Chapter 3. In Section 1 we recall number-theoretic protocols based on the discrete logarithm problem, the RSA assumption, and on the hardness of factoring numbers. Section 2 then turns to constructions based on general one-way functions and using arbitrary statistically-secret commitment schemes. We present identity-based trapdoor systems in Section 3 of this chapter.

Chapter 4 shows the application of trapdoor commitment schemes for the design of efficient non-malleable commitment schemes. The non-malleable protocols we devise, using both trapdoor and identity-based trapdoor commitments based on the discrete-logarithm or RSA problem, require only three rounds of communication between the sender and the receiver in the commitment phase and a few modular exponentiations. This holds under the reasonable condition that the network provides publicly available parameters generated by a trusted party. We also elaborate on different notions of non-malleability in the literature: one definition demands from a non-malleable commitment scheme that one cannot find a commitment of a related message, the other one says that one might be able to find such a commitment but then one is not able to open this commitment with a related message. We show that the second definition is strictly weaker under cryptographic assumptions.

In Chapter 5 we bring trapdoor commitments and identification protocols together. With an identification protocol the prover tries to convince the verifier that he is the holder of the secret key to a public key. The aim of an adversary is to pretend to the verifier to be that prover without actually knowing the secret key. So far, identification has been considered with respect to active attacks where the adversary first runs serial executions with the prover by impersonating the verifier in order to deduce something about the secret key. Then the adversary tries to intrude on behalf of the prover. As for the resettable setting, the adversary may now run concurrent executions with the prover before or while trying to impersonate. Additionally, the adversary may reset the state of the prover. By means of (identity-based) trapdoor commitments we present a general efficient transformation of well-known identification protocols into schemes withstanding reset attacks.

Chapter 6 introduces the issue of universally composable commitments. Usually, commitments are merely designed as stand-alone primitives providing secrecy and unambiguity. The non-malleability problem, for example, arises if we run several instances of the same commitment protocol. In other words, the fact that the commitment scheme is composed with other protocols may cause an unpleasant side effect. Universally composable commitments overcome this problem as they can be securely executed together with other securely composable protocols, may it be commitment, encryption or signature schemes, even if the protocol executions are arbitrarily interleaved. In particular, universally composable commitments wipe out the non-malleability problem. However, compared to our non-malleable schemes in Chapter 4, we pay with a loss in efficiency for this stronger security requirement. Once more, trapdoors in commitments enable us to build such securely composable commitments.

*Chapter 2*

# Definitions

In this chapter we introduce the cryptographic basics and define (trapdoor) commitment schemes. For a broader introduction to the foundations of cryptography we refer to [G98]. Once more, we emphasize that in order to get the basic idea of trapdoors in commitments it suffices to have a rough understanding of cryptographic basics like the discrete-log or RSA assumption and commitment schemes (all covered in Sections 1, 2 and 4.1). Then the reader may skip the more technical parts in this chapter and look at the examples of trapdoor commitment schemes in the next chapter instead.

## 1. Notations

Let $A$ be a probabilistic algorithm, or more formally, a Turing machine with a random tape. We say that $A$ is polynomial-time if there exists a polynomial $p(n)$ such that $A$ takes at most $p(n)$ steps on inputs of length $n$. Algorithm $A$ runs in expected polynomial-time if $A$ is polynomial-time on the average, the expectation taken over the internal random coins.

For a deterministic algorithm $A$ let $a = A(x)$ be the output $a$ of $A$ on input $x$. If $A$ is a probabilistic algorithm then we denote by $A(x)$ the random variable that describes the output of $A$ on input $x$. The probability space is defined by the internal coin tosses of $A$. In this case, we write $[A(x)]$ for the support of $A$ on input $x$. By $a \leftarrow A(x)$ we denote the process of sampling an output $a$ of $A$ on input $x$.

For ease of notation, we sometimes view a probabilistic polynomial-time algorithm as a deterministic one, where a sufficient number of coin tosses is provided as part of the input. Then $A(x, r)$ is the output of algorithm $A$ on input $x$ with random bits $r$; which portion of the input is considered as random will be clear from the context. Unless stated otherwise, it is understood that a polynomial-time algorithm is deterministic; else we call it a probabilistic polynomial-time algorithm.

In addition to Turing machines, we briefly introduce other models of computation. We refer to [BDG95] for more about such models. A polynomial-size circuit family is a sequence $C = (C_n)_{n\in\mathbb{N}}$ of circuits $C_n$ with the property that the total number of gates of $C_n$, including the input gates, is polynomially bounded in $n$. Unless stated differently we always refer to probabilistic circuits.

The bit length of a string $x \in \{0,1\}^*$ is given by $|x|$, and $1^n$ stands for $n$ in unary, i.e., the string that consists of $n$ bits '1'. For two strings $x, y$ of equal length we denote by $x \oplus y$ the bitwise exclusive-or. By $x \in_R \{0,1\}^n$ we refer to a uniformly chosen $n$-bit string $x$. In general, if not annotated differently, any random choice is made independently of any other sampling.

For $n \in \mathbb{N}$ we identify the set $\mathbb{Z}_n$ with the integers between 0 and $n-1$. We sometimes switch between integers and their standard binary encoding. Especially, we embed a string $x \in \{0,1\}^m$ in $\mathbb{Z}_n$ by identifying $x$ with the corresponding integer between 0 and $2^m - 1$ (where $2^m \leq n$), and more generally we write $X \subseteq \mathbb{Z}_n$ if all strings $x \in X \subseteq \{0,1\}^*$ can be embedded in this way. By $x \in_R \mathbb{Z}_n$ we denote a uniformly chosen sample $x$ from $\mathbb{Z}_n$, depending on the context viewed either as an integer or as a bit string.

A function $\delta : \mathbb{N} \to \mathbb{R}_0^+$ is called *negligible* (in $n$) if it vanishes faster than any polynomial fraction. More formally, $\delta$ is negligibe if for any polynomial $p : \mathbb{N} \to \mathbb{R}^+$ there exists an $n_0 \in \mathbb{N}$ such that $\delta(n) < 1/p(n)$ for all $n \geq n_0$. In the rest of this thesis, we abbreviate "there exists $n_0 \in \mathbb{N}$ such that ... for all $n \geq n_0$" by the expression "... for all sufficiently large $n$." The function $\delta(n)$ is *noticeable* (in $n$) if it is not negligible. We say that $\delta(n)$ is *overwhelming* if $1 - \delta(n)$ is negligible.

For example, the function $\delta(n) = 2^{-n}$ in negligible. It is easy to see that with $\delta(n)$ the product $\delta(n) \cdot p(n)$ with any positive polynomial $p(n)$ is also negligible. Additionally, if $\delta(n)$ is negligible and $f(n)$ is noticeable, then $f(n) - \delta(n)$ is noticeable, too.

For a sequence $X = (X_n)_{n\in\mathbb{N}}$ of random variables we denote by $x \leftarrow X_n$ a sample $x$ of the random variable $X_n$. The sequence $X$ is said to be *efficiently samplable* if there exists a probabilistic polynomial-time algorithm $A$ such that $X_n$ and $A(1^n)$ are identically distributed for all $n \in \mathbb{N}$. Observe that the input $1^n$ enables algorithm $A$ to run in polynomial time in the input length $|1^n| = n$.

Two sequences $X = (X_n)_{n\in\mathbb{N}}$ and $Y = (Y_n)_{n\in\mathbb{N}}$ of random variables are called *computationally indistinguishable*, $X \overset{c}{\approx} Y$, if for any probabilistic polynomial-time algorithm $\mathcal{D}$ the advantage

$$\mathsf{Adv}_{\mathcal{D}}^{X,Y}(n) = |\mathrm{Prob}[\mathcal{D}(1^n, x) = 1] - \mathrm{Prob}[\mathcal{D}(1^n, y) = 1]|$$

of $\mathcal{D}$ is negligible, where the probabilities are taken over the coin tosses of $\mathcal{D}$ and the random choice of $x \leftarrow X_n$ and $y \leftarrow Y_n$, respectively. Roughly, interpreting $\mathcal{D}$'s output 1 as a guess that the input is sampled from $X_n$, then a negligible advantage indicates that $\mathcal{D}$ almost makes the same prediction in both cases and cannot tell

the variables $X_n, Y_n$ apart. Note that giving $\mathcal{D}$ the parameter $n$ in unary on one hand tells $\mathcal{D}$ the complexity of the sample, and on the other hand allows $\mathcal{D}$ to run (at least) in polynomial time in $n$, even if the samples are much shorter.

We have chosen a uniform model for defining distinguishers. An alternative way is to adopt the non-uniform model and demand that for any probabilistic polynomial-size circuit family $C = (C_n)_{n\in\mathbb{N}}$ the advantage

$$\mathsf{Adv}_C^{X,Y}(n) = |\mathrm{Prob}[\,C_n(x) = 1] - \mathrm{Prob}[\,C_n(y) = 1]|$$

is negligible, where $x \leftarrow X_n$ and $y \leftarrow Y_n$ for the sequences $X = (X_n)_{n\in\mathbb{N}}$ and $Y = (Y_n)_{n\in\mathbb{N}}$. Here, the additional input $1^n$ is redundant as circuit $C_n$ already depends on $n$ and the circuit's size is polynomially bounded in $n$ anyway. For sake of simplicity we usually adhere the uniform notation in this thesis.

The sequences $X = (X_n)_{n\in\mathbb{N}}$ and $Y = (Y_n)_{n\in\mathbb{N}}$ of random variables are called *statistically close* or *statistically indistinguishable*, $X \overset{s}{\approx} Y$, if

$$\tfrac{1}{2} \cdot \sum_{s\in S_n} |\mathrm{Prob}[\,X_n = s] - \mathrm{Prob}[\,Y_n = s]|$$

is negligible, where $S_n$ is the union of the supports of $X_n$ and $Y_n$. If they are identically distributed, we write $X \overset{d}{=} Y$.

Obviously, identical distributions imply statistical indistinguishability, and statistically close variables are also computationally indistinguishable. The converse does not hold in general.

## 2. Cryptographic Primitives and Assumptions

We review the cryptographic assumptions related to the discussions in this thesis. We start with a very general notion of a one-way function, i.e., a function which is easy to compute but hard to invert on a random value. The existence of one-way functions is necessary for any kind of "non-trivial" cryptography [IL89, OW93]:

**Definition 2.1** (One-Way Function)**.** *A function $f : \{0,1\}^+ \to \{0,1\}^+$ is a one-way function if*

- *efficient evaluation: there exists a polynomial-time algorithm* $\mathsf{Eval}$ *such that* $\mathsf{Eval}(x) = f(x)$ *for all* $x \in \{0,1\}^+$
- *one-wayness: for any probabilistic polynomial-time algorithm $\mathcal{A}$ the inversion probability*

  $$\mathsf{Inv}_{\mathcal{A}}^{f}(n) = \mathrm{Prob}\left[\,\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))\right]$$

  *is negligible in $n$, where the probability is taken over $x \in_{\mathrm{R}} \{0,1\}^n$ and the coin tosses of $\mathcal{A}$.*

*If additionally $f(\{0,1\}^n) = \{0,1\}^n$ for all $n \in \mathbb{N}$ we say that $f$ is a one-way permutation.*

The reason for providing $\mathcal{A}$ with $n$ in unary is that otherwise a function $f$ which maps $x \in \{0,1\}^n$ to the rightmost $\log n$ bits would be one-way, simply because writing down a preimage would take exponential time. Yet, one would not consider this function to be "hard to invert" in an intuitive sense.

Why do we demand that algorithm $\mathcal{A}$ runs in strict polynomial time instead of allowing it to perform an expected polynomial number of steps? Intuitively, an algorithm inverting the function $f$ with noticeable success in reasonable time on the average would also be considered to refute the one-wayness of $f$. Fortunately, Definition 2.1 is robust with respect to such expected polynomial-time inverters. Namely, suppose an algorithm $\mathcal{A}$ inverts $f$ with expected running time $p(n)$ and success probability $1/q(n)$ for infinitely many $n$'s, where $p$ and $q$ are polynomials. Then, by Markov's inequality, $\mathcal{A}$ inverts $f$ in strict polynomial time $2q(n)p(n)$ and noticeable success probability $1/2q(n)$ for infinitely many $n$'s, and proves that $f$ is not one-way. Therefore, in the sequel we usually restrict ourself to inversion algorithms running in polynomial time in the worst case.

The definition of one-wayness can also be given with respect to non-uniform polynomial-size circuits: for any probabilistic polynomial-size circuit family $C = (C_n)_{n\in\mathbb{N}}$ the probability $\mathsf{Inv}^f_C(n) = \mathrm{Prob}\big[C_n(f(x)) \in f^{-1}(f(x))\big]$ is negligible. All the following assumptions in this section can be stated for circuits, too. Analogously to the uniform approaches these non-uniform counterparts are also widely accepted.

A putative instantiation of a one-way function is the well-known RSA function [RSA78]. Given an integer $N = pq$ of distinct $n/2$-bit primes $p, q$ and an integer $e$ relatively prime to Euler's totient function $\varphi(N) = (p-1)(q-1)$, the function value for $x \in \mathbb{Z}_N^*$ is $\mathrm{RSA}_{N,e}(x) = x^e \bmod N$. Note that this is a permutation over $\mathbb{Z}_N^*$. We also remark that without knowledge of the factorization of $N$ one is still able to efficiently compute the inverse $r^{-1} \in \mathbb{Z}_N^*$ and the power $r^e \in \mathbb{Z}_N^*$ to a given $r \in \mathbb{Z}_N^*$ and some polynomially bounded $e$.

The RSA function does not fit into Definition 2.1 since, instead of having a single function $f$, we deal with a set of functions indexed by $N$ and $e$. Also, the domain and range $\mathbb{Z}_N^*$ depend on this index. Nevertheless, we do not consider such indexed one-way functions (aka. collections of one-way functions) rigorously because the formalization is more complicated. Basically, one augments Definition 2.1 by an index generation algorithm that outputs a random index $i$ (e.g., random $N$ and $e$), and the probability of $\mathcal{A}(1^n, i, f_i(x))$ returning a preimage is taken over the choice of the index $i$, the uniform choice of $x$ in the domain, and $\mathcal{A}$'s random coins.

Let us formally state the assumption that RSA is one-way. To this end, we assume that there is some efficient algorithm $\mathsf{IndexGen}(1^n)$ that outputs $N, e$ as described above; we do not specify exactly how the algorithm does that, e.g., if the primes $p, q$ have a special form like $p = 2p' + 1, q = 2q' + 1$ for other primes

$p', q'$, how large $e$ is, etc. Some care must be taken [B99], but this is beyond our scope, and does not give useful insight in the constructions solely applying the RSA function in a "black-box" manner:

**Definition 2.2** (RSA Assumption)**.** *For any probabilistic polynomial-time algorithm $\mathcal{A}$ the inversion probability*

$$\mathsf{Inv}_{\mathcal{A}}^{RSA}(n) = \mathrm{Prob}[\,\mathcal{A}(1^n, N, e, x^e \bmod N) = x]$$

*is negligible in $n$, where the probability is taken over the choice of $(N, e) \leftarrow \mathsf{IndexGen}(1^n)$, $x \in_{\mathrm{R}} \mathbb{Z}_N^*$ and $\mathcal{A}$'s internal random coins.*

One reason for RSA being a good candidate for a one-way function is the random-self-reducibility property of RSA [AFK89]. Roughly, this means that computing the $e$-th root of any $y$ is as hard as computing it for a random $y$. More formally, assume that there exists some probabilistic polynomial-time algorithm inverting a random $y = x^e \bmod N$ for $(N, e) \leftarrow \mathsf{IndexGen}(1^n)$ with some probability $\delta(n)$ taken over the choice of $(N, e)$ and $x$ and the coin tosses. Then there is an efficient algorithm that inverts in comparable time *any* $y \in \mathbb{Z}_N^*$ for $(N, e) \leftarrow \mathsf{IndexGen}(1^n)$ with the same probability $\delta(n)$; this time, the probability space is defined by the choice of $(N, e)$ and the internal random coins. The latter algorithm chooses $r \in_{\mathrm{R}} \mathbb{Z}_N^*$, computes $y' = yr^e \bmod N$ and runs the former algorithm on input $(y', N, e)$. Now $y'$ is uniformly distributed in $\mathbb{Z}_N^*$. If the inverter returns a preimage $x'$ of $y'$, then $x = x'r^{-1} \bmod N$ is a preimage of $y$.

Apparently, the RSA assumption implies that factoring is intractable; otherwise one could simply compute $\varphi(N)$ and $d = e^{-1} \bmod \varphi(N)$ to derive $x = (x^e)^d \bmod N$. The other direction, intractability of factoring implies one-wayness of RSA, is not known to hold (and there is some indication that this might not be true [BV98]).

To present the factoring assumption we again presume that there is some algorithm $\mathsf{IndexGen}(1^n)$ that returns a random modulus $N = pq$ of $n/2$-bit primes $p, q$. Note that we consider the factoring problem with respect to RSA-like moduli:

**Definition 2.3** (Factoring Assumption)**.** *For any probabilistic polynomial-time algorithm $\mathcal{A}$ the inversion probability*

$$\mathsf{Inv}_{\mathcal{A}}^{Fact}(n) = \mathrm{Prob}[\,\mathcal{A}(1^n, N) = (p, q)]$$

*is negligible in $n$, where the probability is taken over the choice of $N = pq \leftarrow \mathsf{IndexGen}(1^n)$ and $\mathcal{A}$'s internal random coins.*

Calling the probability of $\mathcal{A}$ factoring $N$ the inversion probability refers to the fact that one can view the mapping of two random primes $p, q$ to the product $N = pq$ as a one-way function.

Though being around for more than twenty years now, RSA still is essentially unbroken. Another presumable one-way function that withstood virtually all attacks so far is based on the intractability of computing logarithms in groups like $\mathbb{Z}_p^*$ or elliptic curves [DH76]. We simplify and consider the discrete-log problem with respect to prime order subgroups of $\mathbb{Z}_p^*$ only. Unless stated otherwise all assumptions and results can be transferred to other prime order groups like elliptic curves.

Suppose once more that there is an efficient algorithm $\mathsf{IndexGen}(1^n)$ generating a random prime $p$, an $n$-bit prime $q$ with $q|p-1$, and a generator $g$ of a subgroup $G_q$ of order $q$. Again, details on this process are omitted. The one-wayness assumption says that it is hard to find the discrete logarithm $x \in \mathbb{Z}_q$ given $g$ and $g^x$:

**Definition 2.4** (Discrete Logarithm Assumption)**.** *For any probabilistic polynomial-time algorithm $\mathcal{A}$ the inversion probability*

$$\mathsf{Inv}_{\mathcal{A}}^{DL}(n) = \mathrm{Prob}[\,\mathcal{A}(1^n, p, q, g, g^x \bmod p) = x]$$

*is negligible in $n$, where the probability is taken over the choice of $(p, q, g) \leftarrow \mathsf{IndexGen}(1^n)$, $x \in_{\mathrm{R}} \mathbb{Z}_q$ and $\mathcal{A}$'s internal random coins.*

Similar to RSA, the discrete-logarithm problem is random-self-reducible: given $p, q, g$ and $y \in \mathbb{Z}_p^*$ choose $r \in_{\mathrm{R}} \mathbb{Z}_q$ and set $y' = yg^r \bmod p$ such that $y'$ is uniformly distributed; the discrete logarithm $x' = \log_g y'$ yields the discrete logarithm $\log_g y = x' - r \bmod q$ to element $y$.

In the sequel, we sometimes omit the reductions $\bmod N$ and $\bmod p$ if they are clear from the context and write for instance $g^x$ instead of $g^x \bmod p$.

## 3. Interactive Protocols

In this section we introduce the model of joint computations. Instead of dipping into the technical details of interactive Turing machines (see [G98], for example) we rather stick to a more intuitive viewpoint of algorithms that are somehow connected and can interchange messages.

In an interactive protocol between two parties, Alice and Bob, both parties are activated alternatingly. In each activation the corresponding party performs a local computation and then either stops or sends a message to the other party upon which that party is activated and the sender goes idle. If one party stops we assume that the other party is activated once more before halting. Both parties may then give a private output.

Let $w$ be a common input to Alice and Bob and $x$ and $y$ be the private inputs to the parties. Assume that Alice starts the protocol (the case that Bob begins is symmetric). Denote Alice's first message by $a_1$, Bob's reply by $b_1$, Alice's second message to Bob by $a_2$ and so on. Note that $a_1, b_1, a_2, b_2, \ldots$ are

random variables (over the random coins $\alpha$ and $\beta$ of ALICE and BOB) that depend on the inputs and the previously received messages. We therefore write $a_1 = \text{ALICE}(w, x, \alpha)$, $b_1 = \text{BOB}(w, y, \beta, a_1)$, $a_2 = \text{ALICE}(w, x, \alpha, a_1, b_1)$ etc., where we suppose a prefix-free encoding of the communication for simplicity. It is understood that $a_1 \leftarrow \text{ALICE}(w, x)$, $b_1 \leftarrow \text{BOB}(w, y, a_1)$, $a_2 \leftarrow \text{ALICE}(w, x, a_1, b_1), \ldots$ refers to an ordered sampling process, i.e., a party's message is picked with respect to the same random coins as the previous ones.

We write $\mathsf{view}_{\text{ALICE}(x),\text{BOB}(y)}(w)$ for the random variable describing a tuple of five entries: the messages communicated between both parties on inputs $w, x$ and $w, y$, the random bits of each party, and some private output of each party. We call a sample

$$v = (v_{\mathsf{msg}}, v_{\mathsf{rnd},\text{ALICE}}, v_{\mathsf{rnd},\text{BOB}}, v_{\mathsf{out},\text{ALICE}}, v_{\mathsf{out},\text{BOB}}) \leftarrow \mathsf{view}_{\text{ALICE}(x),\text{BOB}(y)}(w)$$

of this variable an *augmented view*; it consists of the view $v_{\mathsf{msg}}$ representing the communication between the parties, the random bits $v_{\mathsf{rnd},\text{ALICE}}$ and $v_{\mathsf{rnd},\text{BOB}}$ and the additional outputs $v_{\mathsf{out},\text{ALICE}}$ and $v_{\mathsf{out},\text{BOB}}$ of the parties. Notationally, we adopt a C++-like style and denote the components of a sample $u \leftarrow \mathsf{view}_{\text{ALICE}(x),\text{BOB}(y)}(w)$ for example by $u_{\mathsf{msg}}, u_{\mathsf{rnd},\text{ALICE}}$ and so on.

An adversary may, for instance, impersonate ALICE's part in a predetermined protocol between ALICE and BOB in order to fool BOB. In this case we usually mark adversarial controlled parties with an asterisk, e.g., by writing $\text{ALICE}^*$ and $v \leftarrow \mathsf{view}_{\text{ALICE}^*(x),\text{BOB}(y)}(w)$, $v_{\mathsf{rnd},\text{ALICE}^*}$, $v_{\mathsf{out},\text{ALICE}^*}$ etc.

Occassionally, we allow another party CARROL to participate in a preprocessing step and to generate another input $\sigma$ for the parties by sampling it when running on common input $w$ and private input $z$.[1] For instance, CARROL, on input $1^n$, may pick a random $n$-bit RSA-modulus $N$ and place it into $\sigma$. This value $\sigma$ is prepended to the augmented view and the entry for a sample $v \leftarrow \mathsf{view}_{\text{ALICE}(x),\text{BOB}(y),\text{CARROL}(z)}(w)$ is denoted by $v_\sigma$. The probability space is defined over the the random coins of all three parties. Typically, it is presumed that CARROL cannot be corrupted and is thus called a trusted third party and named $\mathcal{T}$; the string $\sigma$ is said to be a public string, common reference string or a public parameter, and the model is called the *common reference string model* or *public parameter model*.

[1] We grant CARROL access to the common input $w$ of ALICE and BOB. Sometimes it is preferable to let CARROL generate $\sigma$ without knowing $w$. This can always be accomplished by putting $w$ into ALICE's and BOB's private inputs $x, y$ instead; nonetheless, in all examples discussed in this thesis, we let CARROL's input $z$ be empty and $w$ be the security parameter in unary, which is accessible by all parties anyway.

## 4. Commitment Schemes

We transfer the intuition of the box setting into a formalization of commitment schemes. Our somehow "minimal" definition captures only the secrecy and binding property. That is, in contrast to the more sophisticated definition of a universally composable commitment protocol we will consider in Chapter 6, we neglect the issue of interdependency of protocol executions causing for example the non-malleability problem. There are two reasons for this. First, the basic approach here is easier to understand and gives a good intuition about commitments schemes. Second, in some settings the basic notion is sufficient, e.g., we are able to derive non-malleable commitments from certain schemes obeying only the "minimal" definition here.

### 4.1. Outline

As discussed earlier, a commitment scheme is an interactive protocol between two parties, the sender $\mathcal{S}$ holding a message, and the receiver $\mathcal{R}$. In some commitment protocols a trusted third party assists by publishing public parameters at the outset of the protocol execution; both parties, the sender and the receiver, have then access to this string.

The whole protocol is divided into the commitment phase and the decommitment stage. In the commitment phase, the sender gives some jumbled information about the message to the receiver such that, on one hand, even a malicious receiver $\mathcal{R}^*$ does not gain any information about the message of the honest $\mathcal{S}$ (secrecy), and on the other hand, a possibly dishonest sender $\mathcal{S}^*$ cannot find matching openings for different messages for a given commitment to $\mathcal{R}$ (unambiguity).

In the decommitment phase, the sender is supposed to transmit the key to "unscramble". In the algorithmic setting, this boils down to sending the original message and some evidence that the commitment really jumbles this message. Usually, the sender's random coins form this evidence, because the receiver can recompute the sender's commitment from the original message and the coins in order to check the correctness. Although there are some examples where the sender computes the evidence differently, here we adopt the simplification that the sender transmits all random coins used during the commitment phase; almost all protocols we discuss have this property. We remark that this implies that the decommitment can be done with a single transmission from the sender to the receiver, whereas the commitment phase is an interactive process in general.

In addition to secrecy and unambiguity, we also demand that a commitment scheme is complete. This means that if both parties honestly obey the protocol description then the receiver should accept the commitment and decommitment of the sender as a valid execution. Concerning secrecy and the binding property, there are two fundamental kinds of commitment schemes:

- a scheme is *statistically binding and computationally secret* if any arbitrary powerful malicious $\mathcal{S}^*$ cannot open a valid commitment ambiguously except with negligible probability, and two commitments are computationally indistinguishable for any probabilistic polynomial-time (possibly malicious) $\mathcal{R}^*$. If the binding property holds unconditionally and not only with high probability, then we call the scheme unconditionally binding or perfectly binding.
- a scheme is *computationally binding and statistically secret* if it satisfies the "dual" properties, that is, if the distribution of the commitments are statistically close for any arbitrary powerful $\mathcal{R}^*$, and yet opening a valid commitment ambiguously contradicts the hardness of some cryptographic assumption. If the distribution of the commitments of any messages are identical, then a statistically-secret scheme is called perfectly secret.

It is not hard to see that a commitment scheme cannot be statistically binding and statistically secret simultaneously.[2] It hence suffices if we say that a commitment scheme is statistically secret or statistically binding; it is then clear that the other property is achievable in a computational sense only. Both categories share the subset of commitment protocols that are merely computationally binding and computationally secret.

To limit the power of adversaries in two-party protocols it is usually assumed that both parties verify structural properties of the incoming messages (if possible at all). For example, the receiver should check that a value of the sender belongs to a certain interval, that an element $g$ really generates a group of prime order $q$ etc. Normally, it is obvious what and how the parties should check for structural correctness and we thus do not mention such verifications explicitly. We call a commitment valid if the receiver does not reject the sender's commitment due to an error in such a verification step.

### 4.2. Statistically-Binding Commitment Schemes

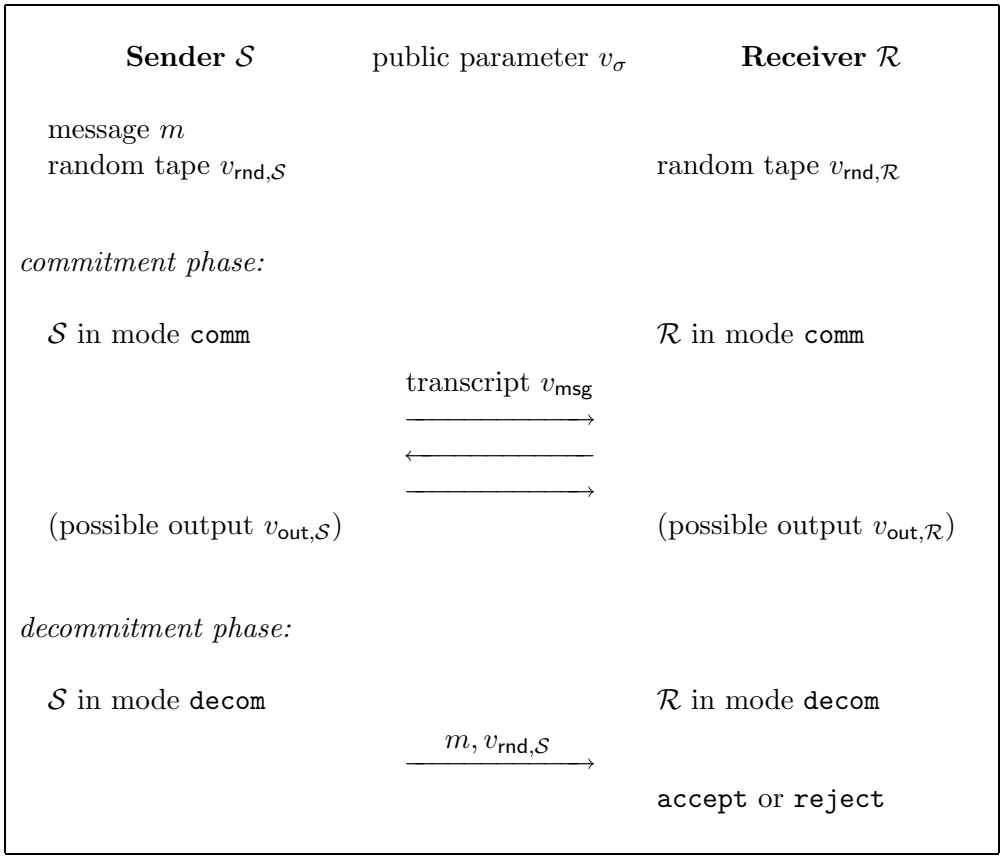
For clarity, we present the definitions of the fundamental notions for commitment schemes individually. In both cases we include a trusted third party $\mathcal{T}$ supplementing a public string $\sigma$ at the outset of the execution. If this third party is not needed then let it stay idle and let $\sigma$ be empty.

Which messages can be committed to? The protocol specifies the message space in form of a sequence $\mathsf{M} = (\mathsf{M}_n)_{n\in\mathbb{N}}$ of sets $\mathsf{M}_n \subseteq \{0,1\}^*$. We presume that the length of messages is bounded polynomially in $n$, i.e., there is a polynomial $p(n)$ such that $|m_n| \leq p(n)$ for any sequence $(m_n)_{n\in\mathbb{N}}$ of messages $m_n \in \mathsf{M}_n$. If

[2] For perfect secrecy and unambiguity this is easy to see since perfect secrecy implies that for a commitment there is a valid decommitment for any message, contradicting the perfect unambiguity demanding that no commitment can be opened with distinct messages. For statistical security the argument is only slightly more involved.

the parties run a protocol execution for security parameter $n$ then the sender is allowed to commit to any message $m \in \mathsf{M}_n$. For instance, in a bit commitment scheme we have $\mathsf{M}_n = \{0,1\}$ for all $n$ and the security paramter only determines the level of secrecy and unambiguity.

FIGURE 1. View of Execution of Commitment Protocol




There is a subtle point on defining commitment schemes: by construction they consist of two stages, while we introduced interactive protocols as single-phase processes. We use the following trick to overcome this problem: we think of $\mathcal{S}$ (and also of $\mathcal{R}$) as two algorithms combined in one. One algorithm is activated when receiving the distiniguished symbol `comm` as input, and then runs the commitment phase with the other party. The other incorporated algorithm merely processes the decommitment phase when getting `decom` (and some input from the commitment stage) as input. For ease of notation, we also adopt this notation for dishonest parties and simulators.

Figure 1 depicts the situation and notation. Further discussions succeed the definition.

**Definition 2.5** (Statistically-Binding Commitment Scheme)**.** *A tuple* $(\mathcal{S}, \mathcal{R}, \mathcal{T})$ *of probabilistic polynomial-time algorithms* $\mathcal{S}, \mathcal{R}, \mathcal{T}$ *is called a statistically-binding* $\mathsf{M}$*-commitment scheme if*

- *completeness: for all* $n \in \mathbb{N}$*, any message* $m_n \in \mathsf{M}_n$*, any augmented view* $v \in [\mathsf{view}_{\mathcal{S}(\mathtt{comm}, m_n), \mathcal{R}(\mathtt{comm}), \mathcal{T}}(1^n)]$ *we have*

$$\mathcal{R}(\mathtt{decom}, 1^n, v_\sigma, v_{\mathsf{msg}}, v_{\mathsf{rnd},\mathcal{R}}, m_n, v_{\mathsf{rnd},\mathcal{S}}) = \mathtt{accept}.$$

- *secrecy: for any sequences* $(m_n)_{n\in\mathbb{N}}$*,* $(m'_n)_{n\in\mathbb{N}}$ *of messages* $m_n, m'_n \in \mathsf{M}_n$ *and any probabilistic polynomial-time algorithm* $\mathcal{R}^*$ *the random variables*

$$v_{\mathsf{msg}} \quad \textit{defined by} \quad v \leftarrow \mathsf{view}_{\mathcal{S}(\mathtt{comm}, m_n), \mathcal{R}^*(\mathtt{comm}, m_n, m'_n), \mathcal{T}}(1^n)$$

*and*

$$v'_{\mathsf{msg}} \quad \textit{defined by} \quad v' \leftarrow \mathsf{view}_{\mathcal{S}(\mathtt{comm}, m'_n), \mathcal{R}^*(\mathtt{comm}, m_n, m'_n), \mathcal{T}}(1^n)$$

*are computationally indistinguishable.*

- *unambiguity: for any (possibly unbounded) algorithm* $\mathcal{S}^*$ *the probability that for* $v \leftarrow \mathsf{view}_{\mathcal{S}^*(\mathtt{comm}), \mathcal{R}(\mathtt{comm}), \mathcal{T}}(1^n)$ *we have*

$$(m_n, m'_n, s, s') = \mathcal{S}^*(\mathtt{decom}, 1^n, v_\sigma, v_{\mathsf{msg}}, v_{\mathsf{rnd},\mathcal{S}^*})$$

*for different messages* $m_n, m'_n \in \mathsf{M}_n$ *and counterfeits* $s, s'$ *of the random coins of the honest sender such that*

$$\mathcal{R}(\mathtt{decom}, 1^n, v_\sigma, v_{\mathsf{msg}}, v_{\mathsf{rnd},\mathcal{R}}, m_n, s) = \mathcal{R}(\mathtt{decom}, 1^n, v_\sigma, v_{\mathsf{msg}}, v_{\mathsf{rnd},\mathcal{R}}, m'_n, s') = \mathtt{accept}$$

*is negligible (over the internal random coins of all parties). If the probability is zero, then we call the scheme perfectly binding.*

Some remarks follow. In the opening step, the receiver $\mathcal{R}$ gets the sender's message and random coins as well as the view from the commitment stage, including his own random coins. This enables us to define $\mathcal{R}(\mathtt{decom}, \cdot)$ as a deterministic algorithm, since coin tosses for this algorithm can already be placed into $v_{\mathsf{rnd},\mathcal{R}}$. The receiver in the opening step returns a self-explanatory decision `accept` or `reject`. This decision can also be based on the structural verification steps in the commitment phase, and lead to rejection only later.

In our definition the sender passes all the random coins from the commitment phase to the receiver in the opening step. Sometimes the sender $\mathcal{S}$ may not want to reveal all random bits but only some evidence that the commitment really encapsulates the message. Indeed, we will touch such protocols in Chapter 3. If so, we let the sender compute with some (wlog. deterministic) function such an evidence from the message, his random coins and the communication, and let the sender hand this evidence together with the message to the receiver as decommitment. For simplicity though, we do not include this in the upcoming definitions either.

The secrecy requirement should hold for any adversarial receiver, even if the receiver knows the alternatives $m_n$ and $m'_n$ to which $\mathcal{S}$ commits, and no matter how this receiver deviates from the protocol specification —recall that the honest sender is supposed to stop the execution if he detects misbehavior, hence the adversary's possibility to deviate is limited to the case that his messages are somewhat indistinguishable from the ones of $\mathcal{R}$.

Secrecy as we have defined it comes in a non-uniform flavor. That is, the secrecy requirement demands that there are no sequences of messages for which the receiver can distinguish the commitments. In other words, a protocol providing secrecy disallows the pure existence of such messages (although these sequences may not be efficiently computable). A uniform approach would be to let the receiver $\mathcal{R}^*$ pick the messages $m_n$ and $m'_n$ after learning the public parameters for security parameter $1^n$. This implicitly defines two efficiently samplable sequences of random variables describing the messages. Similar to the case of uniform and non-uniform inverters for one-way functions, all results concerning computationally-secret commitments can be put in the uniform setting as well, although we stick to this non-uniform version for simplicity.

An attack on the binding property is described by letting a malicious sender $\mathcal{S}^*$ first execute the commitment stage with the honest receiver, and then decide how to fool the receiver with distinct, valid openings. Observe that, although $\mathcal{S}^*$ combines two algorithms, if we run $\mathcal{S}^*$ in mode $\texttt{decom}$ and give it $v_\sigma, v_{\mathsf{msg}}, v_{\mathsf{rnd},\mathcal{S}^*}$ from the commitment sample, then all information gathered by $\mathcal{S}^*$ in mode $\texttt{comm}$ is available to $\mathcal{S}^*(\texttt{decom}, \cdot)$, too. Analogously to $\mathcal{R}^*(\texttt{decom}, \cdot)$, we presume wlog. that $\mathcal{S}^*(\texttt{decom}, \cdot)$ works deterministically.

### 4.3. Statistically-Secret Commitment Schemes

As for statistically-secret commitments recall that, concerning information-theoretical security, the roles of unambiguity and secrecy are swapped. Besides this, the definition is very similar to the one of a statistically-binding scheme:

**Definition 2.6** (Statistically-Secret Commitment Scheme)**.** *A tuple $(\mathcal{S}, \mathcal{R}, \mathcal{T})$ of probabilistic polynomial-time algorithms $\mathcal{S}, \mathcal{R}, \mathcal{T}$ is called a statistically-secret $\mathsf{M}$-commitment scheme if*

- *completeness: for all $n \in \mathbb{N}$, any message $m_n \in \mathsf{M}_n$, any augmented view $v \in [\mathsf{view}_{\mathcal{S}(\texttt{comm}, m_n), \mathcal{R}(\texttt{comm}), \mathcal{T}}(1^n)]$ we have*

$$\mathcal{R}(\texttt{decom}, 1^n, v_\sigma, v_{\mathsf{msg}}, v_{\mathsf{rnd},\mathcal{R}}, m_n, v_{\mathsf{rnd},\mathcal{S}}) = \texttt{accept}.$$

- *secrecy: for any sequences $(m_n)_{n\in\mathbb{N}}$, $(m'_n)_{n\in\mathbb{N}}$ of messages $m_n, m'_n \in \mathsf{M}_n$ and any (possibly unbounded) algorithm $\mathcal{R}^*$ the random variables*

$$v_{\mathsf{msg}} \quad \textit{defined by} \quad v \leftarrow \mathsf{view}_{\mathcal{S}(\texttt{comm}, m_n), \mathcal{R}^*(\texttt{comm}, m_n, m'_n), \mathcal{T}}(1^n)$$

*and*

$$v'_{\mathsf{msg}} \quad \textit{defined by} \quad v' \leftarrow \mathsf{view}_{\mathcal{S}(\mathtt{comm},m'_n),\mathcal{R}^*(\mathtt{comm},m_n,m'_n),\mathcal{T}}(1^n)$$

*are statistically close; if they are identically distributed we say that the scheme provides perfect secrecy.*

- *unambiguity: for any probabilistic polynomial-time algorithm $\mathcal{S}^*$ the probability that for $v \leftarrow \mathsf{view}_{\mathcal{S}^*(\mathtt{comm}),\mathcal{R}(\mathtt{comm}),\mathcal{T}}(1^n)$ we have*

$$(m_n, m'_n, s, s') = \mathcal{S}^*(\mathtt{decom}, 1^n, v_\sigma, v_{\mathsf{msg}}, v_{\mathsf{rnd},\mathcal{S}^*})$$

*for different messages $m_n, m'_n \in \mathsf{M}_n$ and counterfeits $s, s'$ of the random coins of the honest sender such that*

$$\mathcal{R}(\mathtt{decom}, 1^n, v_\sigma, v_{\mathsf{msg}}, v_{\mathsf{rnd},\mathcal{R}}, m_n, s) = \mathcal{R}(\mathtt{decom}, 1^n, v_\sigma, v_{\mathsf{msg}}, v_{\mathsf{rnd},\mathcal{R}}, m'_n, s') = \mathtt{accept}$$

*is negligible (over the internal random coins of all parties).*

### 4.4. Expanding the Message Space

We remark that given a commitment scheme for message space $\mathsf{M} = (\mathsf{M}_n)_{n\in\mathbb{N}}$ it is easy to devise a commitment scheme for messages $\mathsf{M}_n^{p(n)}$ for any polynomial $p(n)$. Namely, for parameter $n$ repeat $p(n)$ independent executions of the original protocol in parallel. Obviously, the derived scheme also provides secrecy and unambiguity, inheriting the corresponding statistical property. In particular, one can extend a bit commitment scheme to a protocol that allows to commit to polynomially bounded messages.

Another solution to derive commitment protocols for large messages are so-called collision-intractable hash function. Loosely speaking, these are functions $H$ compressing large inputs to small outputs, but such that it is infeasible to find collisions $x \neq x'$ with $H(x) = H(x')$. Given such a hash function mapping long messages to the original space $\mathsf{M}_n$, the sender first applies the hash function to his message and then runs the commitment protocol on this smaller hash value. For statistically-binding commitment schemes, though, this reduces the unambiguity to computational unambiguity, whereas for statistically-secret schemes the combined protocol also hides the message information-theoretically. In fact, collision intractable hash function suffice to construct statistically-secret commitment schemes [NY89, DPP93, HM96].

### 4.5. Discrete-Logarithm-Based Example

Let us consider an example of a perfectly-secret and computationally-binding commitment scheme based on the discrete logarithm problem; further examples

follow when presenting constructions of trapdoor commitment schemes. This example here will also serve as a base for explaining trapdoor and identity-based trapdoor commitments in the following sections.

Let $p$ and $q$ be a large primes such that $q|(p-1)$. Let $g$ be a generator of the group $G_q \subseteq \mathbb{Z}_p^*$ of order $q$. The discrete logarithm assumption states that, given $p, q, g$ and $h = g^x \bmod p$ for random $x \in_{\mathrm{R}} \mathbb{Z}_q$, it is infeasible to compute the discrete logarithm $x$ of $h$ to base $g$. Under this assumption we now construct a secure commitment protocol.

We assume that a trusted third party chooses and publishes $p, q, g$ and $h$ at the outset of the protocol (we also assume that $h \neq 1$ such that $h$ is also a generator of $G_q$). For the commitment to a message $m \in \mathbb{Z}_q$ the sender $\mathcal{S}$ selects a random $r \in_{\mathrm{R}} \mathbb{Z}_q$ and computes $M = g^m h^r \bmod p$ and transmits this value $M$ to the receicer $\mathcal{R}$, who simply checks that $M \in G_q$ by verifying $M \in \mathbb{Z}_p^*$ and $M^q = 1 \bmod p$. In the decommitment step, $\mathcal{S}$ hands $m, r$ to the receiver. The receiver checks that $m, r \in \mathbb{Z}_q$ and that $M = g^m h^r \bmod p$ for the commitment $M$.

The scheme is perfectly secret: if the sender follows the prescribed program then $M$ is just a random group element. This is so because the random element $h^r$ hides the factor $g^m$ information-theoretically.

The scheme is computationally binding: if a malicious sender finds valid openings $m, r \in \mathbb{Z}_q$ and $m', r' \in \mathbb{Z}_q$ with $m \neq m'$ to the previously given commitment $M$, then $g^m h^r = M = g^{m'} h^{r'} \bmod p$ and therefore $g^{m-m'} = h^{r'-r} \bmod p$. Since $m \neq m'$ we have $r \neq r'$ and the inverse $(r'-r)^{-1}$ to $r'-r$ in $\mathbb{Z}_q$ exists; but then the discrete logarithm of $h$ to $g$ equals $x = (m-m')(r'-r)^{-1} \bmod q$. Put differently, ambiguous decommitments imply that the sender is able to solve the discrete logarithm problem, which, by assumption, is infeasible.

Note that the discussion above hides the asymptotic character of a commitment scheme. Namely, we have defined the scheme with respect to a single instantiation of $p, q, g, h$. The asymptotic parameter $n$ is implicit, and formally we demand for parameter $n$ that $q = q(n)$ is an $n$-bit prime, such that the message space $\mathsf{M}_n \subseteq \mathbb{Z}_{q(n)}$ grows with $n$, etc. Transferring a description as above to asymptotic notion is usually straightforward, and we keep on disregarding this technical nuisance.

## 5. Trapdoor Commitment Schemes

Our notion of the trapdoor property in commitment schemes follows the zero-knowledge approach (cf. [G98]): there is an efficient simulator whose description of the commitment phase (i.e., the public string, the communication and the coin tosses) is indistinguishable from executions with the honest parties, yet this simulator is also able to output some additional trapdoor information which enables to adapt the openings to the commitment for any given messages.

The indistinguishability implies that the whole protocol execution with the simulator could have taken place involving the honest parties. Thus, no adversary impersonating the receiver will be able to detect whether it is run "in the real world" with the third party and the sender, or in an emulation with the simulator. But this is what we seek in order to come up with a security reduction: if an adversary breaks a cryptographic protocol involving commitments, then the adversary's behavior does not change noticeably if we replace the actual commitment execution with a simulated one; otherwise the cases would be distinguishable. Yet, in contrast to the honest parties being tied to their commitment in a true execution, in such a simulated execution we can now open commitments ambiguously, lending us more power and possibly enabling us to prove security of the complex protocol.

Recall the example of the perfectly-secret commitment scheme based on the discrete logarithm problem. There, the trusted party publishes primes $p, q|(p-1)$ and two generators $g, h$ of the group $G_q \subseteq \mathbb{Z}_p^*$ of prime order $q$. To commit to message $m \in \mathbb{Z}_q$ the sender hands $M = g^m h^r \bmod p$ for random $r \in_{\mathrm{R}} \mathbb{Z}_q$ to the receiver, and reveals $m, r$ in the opening phase. We have seen that this scheme is perfectly secret and computationally binding.

The discrete-logarithm scheme also includes a trapdoor. Let the simulator pick $p, q$ and $g$ as the trusted party, and let it generate $h = g^x \bmod p$ for random $x \in_{\mathrm{R}} \mathbb{Z}_q^*$. The simulator publishes these values. Basically, the value $x$, or more precisely, the inverse $x^{-1}$ in $\mathbb{Z}_q^*$, is the trapdoor because if the simulator commits on behalf of the sender to some message $m_0$ by sending $M = g^{m_0} h^{r_0} \bmod p$ for random $r_0 \in_{\mathrm{R}} \mathbb{Z}_q$, then the simulator can open this commitment with any message $m \in \mathbb{Z}_q$ by computing $r = r_0 - (m - m_0)x^{-1} \bmod q$. In this case,

$$M = g^{m_0} h^{r_0} = g^{m_0} h^{r+(m-m_0)x^{-1}} = g^{m_0} h^r g^{m-m_0} = g^m h^r \bmod p.$$

Formally, we define all values necessary to adapt the decommitment as the trapdoor, i.e., here $(x, m_0, r_0)$ form the trapdoor. In the definition below, we do not specify that the simulator first generates a commitment in the prescribed way and knows the message $m_0$ explicitly. In general, the simulator may rather pick an appropriate string $M$ counterfeiting a commitment, but such that $M$ can be opened with any message later on.

Observe that, even for a malicious receiver, the simulator's behavior in this example is identical to the one of the honest parties: the public parameters are identically distributed, and so is the commitment $M$ as well as the adapted decommitments $m, r$ (note that $r$ is uniformly distributed because $r_0$ is). Hence, this is an example of a so-called perfectly-simulative trapdoor commitment.

In the following definition, we consider arbitrary commitment schemes, either statistically secret, statistically binding or computationally with respect to both properties.

**Definition 2.7** (Trapdoor Commitment Scheme)**.** *Let* $(\mathcal{S}, \mathcal{R}, \mathcal{T})$ *be an* $\mathsf{M}$*-commitment scheme. Then the scheme is called a trapdoor* $\mathsf{M}$*-commitment scheme if for any probabilistic polynomial-time algorithm* $\mathcal{R}^*$ *there exists an expected polynomial-time algorithm* $\mathsf{Sim}$ *such that for any sequence* $(m_n)_{n\in\mathbb{N}}$ *of messages* $m_n \in \mathsf{M}_n$ *the following holds:*

> *on input* $(\mathtt{comm}, 1^n)$ *the simulator* $\mathsf{Sim}$ *outputs a tuple*
>
> $$(w_\sigma, w_{\mathsf{msg}}, w_{\mathsf{rnd},\mathcal{R}^*}, w_{\mathsf{out},\mathsf{Sim}}) \leftarrow \mathsf{Sim}(\mathtt{comm}, 1^n)$$
>
> *such that given* $w_{\mathsf{out},\mathsf{Sim}}$ *and the message* $m_n$ *the simulator returns*
>
> $$(w_{\mathsf{rnd},\mathcal{S}}, w_{\mathsf{out},\mathcal{S}}, w_{\mathsf{out},\mathcal{R}^*}) = \mathsf{Sim}(\mathtt{decom}, 1^n, m_n, w_{\mathsf{out},\mathsf{Sim}})$$
>
> *with the property that* $(w_\sigma, w_{\mathsf{msg}}, w_{\mathsf{rnd},\mathcal{S}}, w_{\mathsf{rnd},\mathcal{R}^*}, w_{\mathsf{out},\mathcal{S}}, w_{\mathsf{out},\mathcal{R}^*})$ *is indistinguishable from* $\mathsf{view}_{\mathcal{S}(\mathtt{comm},m_n),\mathcal{R}^*(\mathtt{comm}),\mathcal{T}}(1^n)$.

*We say that the trapdoor scheme is*

- *perfectly simulative if the distributions are identical,*
- *statistically simulative if the random variables are statistically close,*
- *computationally simulative if the random variables are computationally indistinguishable.*

*We call the simulator's output* $w_{\mathsf{out},\mathsf{Sim}}$ *a trapdoor.*

Our definition is kept on a rather simple level. More generally, one could concede the simulator a small error for not finding appropriate values. We do not include this here as the simulations we deal with are errorless. Also note that everything in the opening step is determined by the augmented view, hence it suffices to demand indistinguishability with respect to these outputs, including the random bits and the message.

Also, we remark that our simulator has to prepare an ambiguously openable commitment for a single message only. Alternatively, the simulator could be obliged to output several dummy commitments and to open them later properly after seeing the messages. This can in principle be accomplished with the single-commitment case by letting both parties run independent executions for each message. However, most examples of trapdoor commitment protocols in the next chapter consist of a set-up phase in which the sender and the receiver install parameters that can be applied for several commitments. In this case it usually suffices that the simulator generates the parameters such that it gets to know some secret value (sometimes also called trapdoor). Then the simulator can generate a sequence of dummy commitments and adapt the openings with this secret value.

Another point is that we restrict the receivers to polynomial-time, although they are allowed to have unlimited power in statistically-secret commitment protocols. We explain the motivation for this. The simulator should have comparable

complexity as the honest parties, and is thus supposed to run in expected polynomial time. This implies that if the malicious receiver is unbounded, then there is no possibility for the simulator to use this receiver as an efficient subroutine. However, emulating $\mathcal{R}^*$ in subprocedure calls is the common technique to "fake" the augmented view. Nonetheless, there are cases where the receiver is all-powerful and yet there is an efficient simulator, e.g., if the receiver passively obtains a single messages during the commitment phase, like in the discrete-logarithm example.

Our notion of a trapdoor commitment scheme neglects side information available to $\mathcal{R}^*$ about the message $m_n$, for example if this message has been used in another subprotocol before. This side information is captured by a probabilistic polynomial-time computable function $\mathrm{Hist}(\cdot)$. In comparison to Definition 2.7, the $\mathtt{comm}$-part of the simulator gets as additional input a sample $h \leftarrow \mathrm{Hist}(1^n, m_n)$; this sample is also given to $\mathcal{R}^*$. Basically, providing $\mathsf{Sim}$ with the same side information as $\mathcal{R}^*$ is necessary because the receiver and the simulator should have equal possibilities of basing their messages on this side information. The $\mathtt{decom}$-part of $\mathsf{Sim}$ already gets $m_n$ as input and the actual sample $h$ may be repeated as part of the simulator's output $w_{\mathsf{out},\mathsf{Sim}}$ of the commitment stage. The augmented view generated by this simulator should be indistinguishable from $\mathsf{view}_{\mathcal{S}(\mathtt{comm},m_n),\mathcal{R}^*(\mathtt{comm},h),\mathcal{T}}(1^n)$, with respect to the parties' coin tosses and $h \leftarrow \mathrm{Hist}(1^n, m_n)$. Formally, we demand that for any efficient $\mathcal{R}^*$ there is some simulator $\mathsf{Sim}$ such that for any probabilistic polynomial-time computable function $\mathrm{Hist}(\cdot)$ the above holds.

Finally, it is worth mentioning that Definition 2.7 is robust with respect to parallel repetitions of the basic commitment protocol or concerning the hash-and-commit paradigm. Both approaches have been discussed in the previous section in order to enlarge the message space.

## 6. Identity-Based Trapdoor Commitments

For ease of presentation and since the examples we discuss in the next chapter achieve this, we restrict ourself to non-interactive commitment schemes in the public parameter model for defining identity-based trapdoor commitments. In such a non-interactive commitment protocol, either trapdoor or not, public parameters are published by a trusted party $\mathcal{T}$ and the sender sends a single commitment message to the receiver. That is, the commitment function, parameterized by the public data, maps a message and random coins to a commitment. In particular, the role of the receiver in the commitment phase is limited to the one of a passive observer. We may therefore assume that the receiver does not need coin tosses at all and does not output anything except for the decision.

We first extend the notion of an ordinary commitment scheme to one of an identity-based commitment in the non-interactive case. Such an identity-based commitment takes an aditional identifier as input besides the message, typically this is a random bit string. Specifically, we assume that there is an efficiently

samplable sequence $\mathsf{ID} = (\mathsf{ID}_n)_{n\in\mathbb{N}}$ of random variables $\mathsf{ID}_n$ over $s(n)$-bit strings (where $s(n)$ is some polynomial specified by the commitment protocol). For parameter $n$ we let the sender use some of the random bits for the commitment to sample an identifier $\mathrm{id}_n \leftarrow \mathsf{ID}_n$ and let the sender append this sample $\mathrm{id}_n$ to the commitment in clear. We remark that the commitment itself may also depend on $\mathrm{id}_n$. Then the definitions of statistically-binding and statistically-secret commitment schemes carry over to such *identity-based* $(\mathsf{ID},\mathsf{M})$*-commitment schemes*. To underline the role of the identifiers we itemize them explicitly in the commitment message in the following definition, and write for example $v_{\mathsf{msg}}, \mathrm{id}_n$ instead of $v_{\mathsf{msg}}$ for the view.

For a trapdoor in an identity-based commitment the simulator gets as input a random $\mathrm{id}_0 \leftarrow \mathsf{ID}_n$ and then generates the public parameters on behalf of $\mathcal{T}$. The simulator also outputs a trapdoor information that allows to open commitments involving the identifier $\mathrm{id}_0$ ambiguously. However, it is still infeasible —or even impossible— to find commitments and ambiguous openings under the simulator's public parameters for some id different than $\mathrm{id}_0$. This holds even if one is given the trapdoor information of the simulator. Put differently, the trapdoor is tied to this specific identifier $\mathrm{id}_0$ and does not help to overcome the binding property for other identifiers.

As an example of an identity-based protocol we return to the commitment scheme based on the discrete-logarithm problem. Instead of publishing only two generators $g, h$ of a group $G_q$, this time the trusted party announces three generators $g_1, g_2$ and $h$. A sender with identity $\mathrm{id} \in \{0,1\}^s \subseteq \mathbb{Z}_q$ computes his commitment to $m \in \mathbb{Z}_q$ by $M = (g_1^{\mathrm{id}} g_2)^m h^r \bmod p$ and sends $(\mathrm{id}, M)$ to the receiver. Instructively, the identity determines the generator $g := g_1^{\mathrm{id}} g_2$ and the parties run the well-known protocol on the generators $g$ and $h$. We omit the details that this is indeed an identity-based trapdoor protocol and refer the reader to Chapter 3.

Note that if there are only a few users and the number of identities is small, then there is a trivial solution to derive identity-based trapdoor schemes from ordinary trapdoor systems. Specifically, for each identity id place an independent instance of the basic trapdoor commitment scheme like $g_{\mathrm{id}}, h_{\mathrm{id}}$ into the public string and let the sender with identity id use the corresponding instance when committing. The trapdoor simulator also picks an instance for each identity but such that the simulator knows the trapdoor for $\mathrm{id}_0$ only (e.g., $\log_{g_{\mathrm{id}_0}} h_{\mathrm{id}_0}$). Clearly, this is an identity-based trapdoor commitment scheme. Nonetheless, this solutions becomes impractical if the number of identities is too large: a public string of $2^s$ instances is inacceptable for large $s$. Hence, we are looking for more sophisticated solutions, like the aforementioned one based on the discrete-logarithm problem.

If we adopt an abstract viewpoint and regard commitments as lockable steely boxes, then in identity-based trapdoor commitments only a certain box has got a

trapdoor while the other boxes provide unambiguity. In particular, the trivial solution sketched above can be viewed as giving each party an individual "box" $g_{\mathrm{id}}, h_{\mathrm{id}}$, and only the box of the party with identity $\mathrm{id}_0$ contains a trapdoor. However, as pointed out, individual boxes are too cumbersome. Rather than customizing each box, we envision a general construction kit which is completely assembled on the sender's side by assimilating the sender's identity. Then we can incorporate a "personal" trapdoor to the kit that works only for identity $\mathrm{id}_0$ but not for any other id. In the example above with commitment $M = (g_1^{\mathrm{id}} g_2)^m h^r \bmod p$ the construction kit consists of the components $g_1, g_2, h$ and a sender with identity id first assembles $g_{\mathrm{id}} := g_1^{\mathrm{id}} g_2$ and then uses the "box" $g_{\mathrm{id}}, h$ to commit.

**Definition 2.8** (Non-Interactive Identity-Based Trapdoor Commitment Scheme)**.** *Let $(\mathcal{S}, \mathcal{R}, \mathcal{T})$ be a non-interactive identity-based $(\mathsf{ID}, \mathsf{M})$-commitment scheme. The scheme is called an identity-based trapdoor $(\mathsf{ID}, \mathsf{M})$-commitment scheme if there exists an expected polynomial-time algorithm $\mathsf{Sim}$ such that for any sequence $(m_n)_{n\in\mathbb{N}}$ of messages $m_n \in \mathsf{M}_n$ the following holds:*

> *on input $(\mathtt{comm}, 1^n, \mathrm{id}_0)$ where $\mathrm{id}_0 \leftarrow \mathsf{ID}_n$ the simulator $\mathsf{Sim}$ outputs a tuple*
>
> $$(w_\sigma, w_{\mathsf{msg}}, \mathrm{id}_0, w_{\mathsf{out},\mathsf{Sim}}) \leftarrow \mathsf{Sim}(\mathtt{comm}, 1^n, \mathrm{id}_0)$$
>
> *such that given $w_{\mathsf{out},\mathsf{Sim}}$, $\mathrm{id}_0$ and the message $m_n$ the simulator returns*
>
> $$(w_{\mathsf{rnd},\mathcal{S}}, w_{\mathsf{out},\mathcal{S}}) = \mathsf{Sim}(\mathtt{decom}, 1^n, \mathrm{id}_0, m_n, w_{\mathsf{out},\mathsf{Sim}})$$
>
> *with the property that $(w_\sigma, w_{\mathsf{msg}}, \mathrm{id}_0, w_{\mathsf{rnd},\mathcal{S}}, w_{\mathsf{out},\mathcal{S}})$ is indistinguishable from $\mathsf{view}_{\mathcal{S}(\mathtt{comm},m_n),\mathcal{R}^*(\mathtt{comm}),\mathcal{T}}(1^n)$.*

*We say that the trapdoor scheme is*

- *perfectly simulative if the distributions are identical,*
- *statistically simulative if the random variables are statistically close,*
- *computationally simulative if the random variables are computationally indistinguishable.*

*We call the simulator's output $w_{\mathsf{out},\mathsf{Sim}}$ together with $\mathrm{id}_0$ a trapdoor.*

*Furthermore, if the scheme $(\mathcal{S}, \mathcal{R}, \mathcal{T})$ is computationally binding then the following holds:*

> *for any probabilistic polynomial-time algorithm $\mathcal{S}^*$ the probability that for $(w_\sigma, w_{\mathsf{msg}}, \mathrm{id}_0, w_{\mathsf{out},\mathsf{Sim}})$ output by $\mathsf{Sim}(\mathtt{comm}, 1^n, \mathrm{id}_0)$ for random $\mathrm{id}_0 \leftarrow \mathsf{ID}_n$ we have*
>
> $$(m_n, m_n', s, s', v_{\mathsf{msg}}, \mathrm{id}_n) \leftarrow \mathcal{S}^*(\mathtt{decom}, 1^n, w_\sigma, w_{\mathsf{msg}}, \mathrm{id}_0, w_{\mathsf{out},\mathsf{Sim}})$$

*for different messages* $m_n, m'_n \in \mathsf{M}_n$ *and* $\mathrm{id}_n \in [\mathsf{ID}_n]$ *different than* $\mathrm{id}_0$ *and strings* $s, s'$ *such that*

$$\mathcal{R}(\mathtt{decom}, 1^n, w_\sigma, v_{\mathsf{msg}}, \mathrm{id}_n, m_n, s) = \mathcal{R}(\mathtt{decom}, 1^n, w_\sigma, v_{\mathsf{msg}}, \mathrm{id}_n, m'_n, s') = \mathtt{accept}$$

*is negligible (over the internal random coins of all parties),*

*If the scheme* $(\mathcal{S}, \mathcal{R}, \mathcal{T})$ *is statistically or perfectly binding then the following hods:*

*for any (possibly unbounded) algorithm* $\mathcal{S}^*$ *the probability that for* $(w_\sigma, w_{\mathsf{msg}}, \mathrm{id}_0, w_{\mathsf{out},\mathsf{Sim}})$ *output by* $\mathsf{Sim}(\mathtt{comm}, 1^n, \mathrm{id}_0)$ *for random* $\mathrm{id}_0 \leftarrow \mathsf{ID}_n$ *we have*

$$(m_n, m'_n, s, s', v_{\mathsf{msg}}, \mathrm{id}_n) \leftarrow \mathcal{S}^*(\mathtt{decom}, 1^n, w_\sigma, w_{\mathsf{msg}}, \mathrm{id}_0, w_{\mathsf{out},\mathsf{Sim}})$$

*for different messages* $m_n, m'_n \in \mathsf{M}_n$ *and* $\mathrm{id}_n \in [\mathsf{ID}_n]$ *different than* $\mathrm{id}_0$ *and strings* $s, s'$ *such that*

$$\mathcal{R}(\mathtt{decom}, 1^n, w_\sigma, v_{\mathsf{msg}}, \mathrm{id}_n, m_n, s) = \mathcal{R}(\mathtt{decom}, 1^n, w_\sigma, v_{\mathsf{msg}}, \mathrm{id}_n, m'_n, s') = \mathtt{accept}$$

*is negligible for statistically-binding* $(\mathcal{S}, \mathcal{R}, \mathcal{T})$*, and zero for perfectly-binding* $(\mathcal{S}, \mathcal{R}, \mathcal{T})$ *(over the internal random coins of all parties).*

*Chapter 3*

# Constructions of Trapdoor Commitment Schemes

This chapter introduces several constructions of trapdoor commitment schemes. We distinguish between number-theoretic constructions applying the discrete-logarithm or RSA problem for instance, and complexity-based solutions using general cryptographic assumptions like the existence of one-way functions. We also present constructions of identity-based trapdoor commitments in the concluding section. We remark that we do not discuss the recently announced trapdoor commitment schemes by Barak [B01] which, unlike our solutions, neither rely on the public parameter model nor proofs of knowledge.

## 1. Number-Theoretic Constructions

All constructions of trapdoor commitments in this section rely on the chameleon blobs presented in [BCC88]. They are all perfectly simulative (which means that the output produced with knowledge of the trapdoor looks exactly like a correctly generated commitment and opening) and they satisfy the definition of a trapdoor scheme with side information (i.e., the receiver will not be able to notice the difference to an honest commitment and opening even if he already knows something about the message).

### 1.1. Discrete-Logarithm-Based Construction

We start by presenting the basic non-trapdoor commitment due to Pedersen [P91] and discuss afterwards how to transform it into a trapdoor protocol.

The receiver samples a random group (description) by running $\mathsf{IndexGen}(1^n)$ and obtains a subgroup $G_q \subseteq \mathbb{Z}_p^*$ of prime order $q$ generated by some $g$. Group operations in $\mathbb{Z}_p^*$ are efficiently computable and it is easily verifiable that $g$ indeed generates the subgroup $G_q$. Any other groups with these properties and for which the discrete-logarithm problem is presumably hard work as well.

The receiver picks a secret $x \in_R \mathbb{Z}_q^*$, computes $h = g^x$ and sends $g, h$ as well as the group description $(p, q, g)$ to the sender who checks the correctness of the parameters (i.e., that $p, q$ are prime, that $q|(p-1)$, that $g, h \in \mathbb{Z}_p^* - \{1\}$ and that $g^q = h^q = 1$). The sender now chooses $r \in_R \mathbb{Z}_q$ at random, and for $m$ in the message space $\mathsf{M}_n \subseteq \mathbb{Z}_q$ he computes and sends $M = g^m h^r$. This concludes the commitment phase.

In the decommitment phase, the sender transmits $m, r$ and the receiver checks that this is a proper representation of $M$, i.e., that $m, r \in \mathbb{Z}_q$ and that $M = g^m h^r$. If so, the receiver accepts, and rejects otherwise.

Obviously, this commitment scheme is perfectly secret since $M$ is a random group element. On the other side, if the sender finds two openings $(m, r), (m', r')$ for $m \neq m'$ (and thus $r \neq r'$) of the same $M$, then

$$\log_g h = (m - m')(r' - r)^{-1} \bmod q$$

where $(r - r')^{-1}$ is the inverse of $r - r' \neq 0$ in $\mathbb{Z}_q$. Hence, under the discrete-log assumption it is infeasible to find different openings.

Ambiguous decommitments imply that one knows the discrete logarithm of $h$ to $g$. Vice versa, and this is the essential trapdoor information, knowledge of this discrete logarithm enables to find distinct valid openings: if one has committed to some $M = g^{m_0} h^{r_0}$ then in order to decommit to any $m \in \mathbb{Z}_q$ the holder of the trapdoor $\log_g h$ computes

$$r = r_0 + (m_0 - m)(\log_g h)^{-1} \bmod q$$

It is readily verified that $(m, r)$ is a correct opening to $M$, too.

Alternatively to letting the receiver select the group and the generators, these values may be chosen and given to both the sender and the receiver by some trusted third party before the actual commitment starts. In this case the sender does not need to verify the correctness of the parameters, because the trusted third party follows the sampling procedure honestly.

All that remains is to guarantee that a simulator is able to get to know the discrete logarithm $x$ of $h$ to base $g$ (remember that the receiver chooses $x$ secretly). For example, this can be achieved by letting some trusted party publish the group, $g$ and $h$ at the beginning. The simulator, faking the public string too, generates a group by sampling $\mathsf{IndexGen}(1^n)$ and $h$ as $g^x$ for random, but known $x \in_R \mathbb{Z}_q^*$. The simulator also outputs $M = h^{r_0}$ for random $r_0 \in_R \mathbb{Z}_q$ as a commitment for $m_0 = 0$ and returns the trapdoor $(x, r_0)$. Given some message $m$ the simulator can open $M$ with $m, r = r_0 - mx^{-1} \bmod q$; the distribution of the public data, the commitment $M$ and the opening are identically distributed to the case of an execution with a trusted party and the honest sender, even if $\mathcal{R}^*$ has some side information about the message.

Another possibility to let the simulator learn the trapdoor $\log_g h$ and forgoing a trusted party is to let the receiver give a zero-knowledge proof of knowledge for $x$. Such a proof of knowledge guarantees that the simulator can extract $x$ in expected polynomial time, usually this is accomplished by repeating several executions with the receiver. Although the simulator may be able to extract $x$ from such a proof of knowledge, it follows from the zero-knowledge property that a dishonest sender in a real execution, on the other side, does not learn anything useful about $x$ (e.g., because this sender is not allowed to repeat protocol runs). Thus, for the malicious sender finding ambiguous decommitments is still as hard as without such a proof.

Once the group and generators have been established, either via the public parameter model or by an interactive process, the same parameters can be used to commit to several values. In this case, the trapdoor simulator is also able to open a sequence of given commitments $M_1 = h^{r_{0,1}}, M_2 = h^{r_{0,2}}, \ldots$ ambiguously if it knows the discrete logarithm of the generators by opening each commitment accordingly. The resulting output is still perfectly simulative.

### 1.2. RSA-Based Construction

The RSA-based trapdoor commitment scheme is based on Okamoto's ordinary RSA commitment [O92] and is similar to the discrete-log one. Namely, running $\mathsf{IndexGen}(1^n)$ the receiver chooses an $n$-bit RSA-modulus $N = pq$ and a prime exponent $e \geq 2^{n+1}$; by this choice, the exponent $e$ is relatively prime to $\varphi(N) < 2^{n+1}$ and this fact is publicly verifiable without knowledge of the factorization of $N$. The receiver also picks a random $x \in_{\mathrm{R}} \mathbb{Z}_N^*$, computes $g = x^e \bmod N$ and hands $N, e, g$ to the sender who checks that $e$ is a prime larger than $2^{n+1}$ and that $g \in \mathbb{Z}_N^*$. The sender now selects $r \in_{\mathrm{R}} \mathbb{Z}_N^*$ at random, computes $M = g^m r^e \bmod N$ and commits to $m \in \mathsf{M}_n \subseteq \mathbb{Z}_e$ by sending $M$. In order to decommit, the sender transmits $m, r$ and the receiver checks the correctnes of these values.

Since $e$ is relatively prime to $\varphi(N)$ taking elements to the $e$-th power is a permutation on $\mathbb{Z}_N^*$ and thus the commitment $M$ is a uniformly distributed element in $\mathbb{Z}_N^*$ and reveals nothing about the message. Finding $(m, r), (m', r')$ with $m \neq m'$ (and thus $|m - m'| \in \mathbb{Z}_e - \{0\}$) to the same $M$ yields the equation

$$g^{m-m'} = (r'r^{-1})^e$$

from which the $e$-th root $x$ of $g$ can be easily computed by the formula

$$x = g^a (r'r^{-1})^b \qquad \text{where } a, b \text{ satisfy } ae + b(m - m') = 1$$

The values $a, b$ are easily computable via the extended Euclidean algorithm for relatively prime $e$ and $m - m'$. It follows that coming up with ambiguous decommitments is infeasible under the RSA assumption.

The trapdoor information is the $e$-th root $x$ of $g$ together with the random string $r_0$ used to compute a commitment $M = r_0^e \bmod N$ for message $m_0 = 0$. It

is worth noticing that the trapdoor does not necessarily include the factorization of $N$. With the help of $(x, r_0)$ one can easily transform the commitment $M = r_0^e$ into one of $m$ by letting $r = x^{-m} r_0 \bmod N$. Then, $g^m r^e = g^m (x^{-m})^e r_0^e = r_0^e = M \bmod N$.

Again, in order to let a simulator know $x$ one can either put $N, e, g$ in the public string, in which case the simulator selects a random $x \in \mathbb{Z}_N^*$ and sets $g = x^e \bmod N$ and thus simulates the honest sender perfectly, or we let the receiver give a zero-knowledge proof of knowledge. If the RSA parameters are placed into the public string then we may choose smaller prime exponents $e$ relatively prime to $\varphi(N)$; after all, it is guaranteed that the trusted party chooses $e$ correctly. The choice of a smaller exponent is preferable since it speeds up the exponentiation in the commitment phase.

Analogously to the discrete-logarithm case, the parameters can be applied to commit more than once. If the simulator knows the $e$-th root of $g$ then it can modify the openings to a set of given commitments.

### 1.3. Factoring-Based Construction

The construction in this section resembles the RSA solution and a slightly more restrictive version without reference to trapdoors has appeared in [D95]. For a thorough discussion see [FF02].

We briefly explain the underlying number theory. Let $N = pq$ be an $n$-bit RSA modulus and let $\eta \in \{1, 2, \ldots, n\}$ denote the smallest integer such that $2^{\eta+1}$ neither divides $p-1$ nor $q-1$. Squaring is a permutation over the $2^\eta$-th powers of $\mathbb{Z}_N^*$, that is, squaring is a one-to-one mapping on the group $\{z^{2^\eta} | z \in \mathbb{Z}_N^*\}$. More generally, squaring permutes the $2^n$-th powers for any odd $n$-bit integer $N$, not necessarily being an RSA modulus.

Let the integer $t \geq 1$ bound the length of the messages that can be committed to. The receiver generates a random $n$-bit RSA-modulus $N = pq$ and computes and integer $\tau$ that upper bounds $\eta - 1$. He also calculates $g = x^{2^{\tau+t}} \bmod N$ for random $x \in_{\mathrm{R}} \mathbb{Z}_N^*$ and hands $(N, \tau, t, g)$ to the sender. To commit to a message $m \in \mathsf{M}_n \subseteq \mathbb{Z}_{2^t}$, the sender picks $r \in_{\mathrm{R}} \mathbb{Z}_N^*$ at random, computes $M = g^m r^{2^{\tau+t}} \bmod N$ and sends $M$ to the receiver.

Since $\tau + t \geq \eta$ and squaring is a permutation over the $2^\eta$-th powers the value $M = g^m r^{2^{\tau+t}}$ is a uniformly distributed $2^\eta$-th power if $r$ is selected at random. The message $m$ is therefore perfectly hidden (as long as $N, \tau$ and $g$ are properly chosen; we will discuss this issue at the end of this section). Finding openings $(m, r)$ and $(m', r')$ for the same commitment implies that

$$\left(r' r^{-1}\right)^{2^{\tau+t}} = g^{m-m'} = \left(g^{2^i}\right)^{(m-m')/2^i}$$

where $i < t$ is the maximum value such that $m, m'$ coincide on the $i$ least significant bits. Since the exponents $2^{\tau+t}$ and $(m-m')/2^i$ are relatively prime, one can compute a $2^{\tau+t}$-th root of $g^{2^i}$ with the same technique as in the RSA case. Taking into account $\tau + t - i \geq \tau + 1 \geq \eta$ we thus derive a $2^\eta$-th root $y$ of $g$. If instead of giving the sender $g = x^{2^{\tau+t}}$ we send the identically distributed $g = x^{2^\eta}$, then the sender's success probability of finding ambiguous openings remains unchanged. But this time the root $y$ together with $x$ yields the factorization of $N$ with probability at least $1/2$.

Secrecy relies on the fact that $g$ is indeed a $2^\eta$-th power in $\mathbb{Z}_N^*$. This can either be achieved by letting a trusted party place these values into the public string, or by letting the sender commit to

$$M = \left(g^{2^n}\right)^m r^{2^{\tau+t+n}} \bmod N$$

for the $n$-bit modulus $N$ (and by checking that $N$ is odd). For any odd integer $N$ of $n$ bits squaring is a permutation on the $2^n$-th powers. Therefore, by this choice, $M$ is distributed independently of $m$ and perfect secrecy is guaranteed, no matter how $N, \tau$ and $g$ are chosen. This trick appears in [H99]. Unambiguity follows analogously to the previous case.

A simulator for the trapdoor property (in the public parameter model) chooses $g$ as $g = x^{2^{\tau+t}} \bmod N$ for random $x \in_R \mathbb{Z}_N^*$ such that it knows a $2^{\tau+t}$-th root of $g$. The trapdoor consists of the $2^{\tau+t}$-th root $x$ of $g$ and the random string $r_0$ to compute a commitment $M = r_0^{2^{\tau+t}} \bmod N$ of message $m_0 = 0$. To derive a commitment of $m$ for value $M$, set $r = x^{-m} r_0 \bmod N$ such that $g^m r^{2^{\tau+t}} = g^m (x^{-m})^{2^{\tau+t}} r_0^{2^{\tau+t}} = r_0^{2^{\tau+t}} = M \bmod N$. If there is no public string and the receiver chooses $(N, \tau, t, g)$ —and we use the trick of raising the elements to their $2^n$-th power first— then we add a zero-knowledge proof of knowledge of a $2^{\tau+t}$-th root $x$ of $g$. The simulator is able to extract this root and to proceed as in the public parameter model. In both cases, the parameters can also be used for more than a single commitment, both for honest parties and the trapdoor simulator.

## 2. Complexity-Based Constructions

Next we address trapdoor protocols based on more general assumptions.

### 2.1. Constructions Based on One-Way Functions

We present two fundamental approaches to trapdoor commitment schemes based on one-way functions. One approach works in the public parameter model where a trusted third party generates a public string at the outset of the execution, whereas the other one in the plain model does not require such a set-up mechanism. Both approaches are computationally simulative, i.e., the behavior of the trapdoor simulator is computationally indistinguishable from the one of the honest parties.

**Public Parameter Model.** The ingeniuos approach we present originates in [DIO98] and is based on Naor's commitment scheme [N91]. In Naor's bit commitment protocol it is assumed that a pseudorandom generator is available. Pseudorandom generators are efficient deterministic algorithms that stretch a short random seed into a longer output; this output is computationally indistinguishable from a truly random string of the same length. Pseudorandom generators exist if and only if one-way functions exist [HILL99].

Let $G$ be pseudorandom generator stretching $n$ bits inputs to $3n$ bits output. The receiver first chooses a random $3n$-bit string $\sigma$ and sends it to the sender. To commit to bit $b$ the sender selects a seed $r \in_R \{0,1\}^n$ at random and returns $G(r)$ if $b = 0$ or $G(r) \oplus \sigma$ for $b = 1$. Decommitting is done by sending $(b, r)$, and the receiver verifies that these values match the previously given commitment.

From the pseudorandomness of the generator it follows that the receiver cannot distinguish both cases $b = 0$ and $b = 1$ significantly, i.e., the scheme is computationally secret. As for unambiguity, valid openings $(0, r)$ and $(1, r')$ require that $G(r) = G(r') \oplus \sigma$. But since $\{G(r) \oplus G(r') \mid r, r' \in \{0,1\}^n\}$ has at most $2^{2n}$ elements, the probability that a random $\sigma \in \{0,1\}^{3n}$ hits this set is at most $2^{-n}$. Except for such "bad" $\sigma$ the commitment is perfectly binding. Overall, the protocol provides statistical unambiguity.

Now assume that the string $\sigma$ is put into the public random string. Then the simulator "cheats" by selecting $\sigma$ as $G(r_0) \oplus G(r_1)$ for random $r_0, r_1 \in \{0,1\}^n$ and committing to $G(r_0)$. The trapdoor is the pair $(r_0, r_1)$ because in order to open $G(r_0)$ as 0 simply send $(0, r_0)$, and to open as 1 transmit $(1, r_1)$. The former opening is obviously correct. In the latter case the receiver learns that indeed $G(r_1) \oplus \sigma = G(r_0)$ equals the commitment of the first stage. The output of the simulator however is computationally indistinguishable from the honest case as it is infeasible to tell apart a random string $\sigma$ and a pseudorandom one $G(r_0) \oplus G(r_1)$. Unlike in the case of the number-theoretic construction in the previous section the parameter $\sigma$ here can only be used for a single trapdoor-changeable commitment.

Another possibility of accomplishing trapdoor commitments in the public parameter model is discussed by Feige and Shamir in [FS89].[1] As opposed to the previous example, this approach allows more flexibility concerning secrecy as it supports both computational and statistical secrecy (we address the statistically secret version in the next section). Yet, the commitment only provides computational unambiguity, and the protocol is less efficient.

The public string in the scheme by [FS89] consists of a graph containing a directed Hamiltonian cycle. It should be infeasible to find a cycle in this graph and, therefore, we present a method to generate such a graph such that finding a cycle is as hard as inverting a one-way function. Namely, the graph is generated

[1] In fact, [FS89] do not present their protocol in the public parameter model but rather in the plain model. However, this adaption here is straightforward.

as follows: first, pick a random $x \in_R \{0,1\}^n$ and apply a one-way function $f$ to $x$. Then reduce the function value $f(x)$ via standard Karp reductions to an instance $H$ of the $\mathcal{NP}$-hard problem Directed Hamiltonian Cycle. Although Karp reductions are only designated to solve language problems, if we apply standard reductions then going the reduction steps backwards allows us also to recover a preimage of $f(x)$ given a cycle in $H$. In other words, a witness for the graph yields an efficiently computable witness for $f(x)$, and is therefore hard to find.

The sender commits to bit 0 by permuting the graph $H$ randomly and committing to each bit of the adjacency matrix of this permuted graph individually. This is done with a standard (not necessarily trapdoor) commitment scheme, say, with Naor's non-interactive scheme based on any one-way function in the public parameter model. To commit to bit 1 the sender individually commits to the bits of an adjacency matrix that describes a graph with a random Hamiltonian cycle only and without further edges. Again, this is done with some standard commitment protocol.

To decommit to 0 the sender reveals the permutation and decommits to each standard commitment and the receiver verifies the correctness. That is, the receiver checks that the decommitments are valid and that they match the permuted graph. For an opening to 1 the sender only discloses the decommitments to bits describing the random directed Hamiltonian cycle in the otherwise empty graph.

An ambiguous decommitment to 0 and 1 yields a directed Hamiltonian cycle in $H$, because the 1-decommitment reveals a cycle in the committed graph and the permutation of the 0-opening discloses the positions in the original graph $H$. Ambiguous openings are therefore infeasible to find. On the other hand, secrecy of the standard commitment scheme implies secrecy of the trapdoor commitment. The (reusable) trapdoor is a cycle in $H$, since an honestly generated 0-commitment can later be opened with 1, too, by revealing the cycle in the permutation of $H$ only.

**Plain Model.** The ideas of the previous section work in the plain model, too. Di Crescenzo and Ostrovsky [DO99] present an interactive version of the trapdoor scheme in [DIO98] which does not need public parameters. There, the receiver first commits to a random $3n$-bit string $\alpha$. Then the sender announces a random string $\beta \in_R \{0,1\}^{3n}$, and the receiver opens the commitment to $\alpha$. The string $\sigma$ is defined as $\sigma := \alpha \oplus \beta$ and the sender commits to a bit by using $\sigma$ as in Naor's scheme.

The trapdoor simulator biases the outcome of the first phase by announcing a random string $\beta$ after having learned the commitment to $\alpha$, and completing this phase including the step where the receiver reveals $\alpha$. The simulator rewinds the execution to the step after the receiver has committed to $\alpha$ and, this time, sends $\beta := \alpha \oplus G(r_0) \oplus G(r_1)$. Since this $\beta$ is indistinguishable from random and because

the receiver's commitment is binding, the receiver is likely to send $\alpha$ again. But then $\sigma = G(r_0) \oplus G(r_1)$ and a commitment $G(r_0)$ can be opened both as 0 and 1.

For the Hamiltonian cycle scheme in the plain model the receiver generates the graph $H$ as described in the public parameter case and sends it to the sender at the outset, and also gives a zero-knowledge proof of knowledge that he knows a directed Hamiltonian cycle (in fact, the weaker notion of a witness-hiding proof of knowledge is sufficient, see [FS89] for details). The binding property follows as above since the proof is zero knowledge and does not help to find the cycle. But a simulator is able to extract the cycle from the proof of knowledge efficiently and to get to know the trapdoor which is applicable to several commitments.

## 2.2. Statistically-Secret Trapdoor Commitments

The solutions here work with any statistically-secret commitment protocol, for instance, with the non-interactive one using collision-intractable hash functions [NY89, DPP93, HM96].

The first solution is a modification of the Hamiltonian cycle example above. There, the sender commits to each bit of the adjacency matrix of either a permutation of the graph $H$ or of the matrix describing a trivial graph with a random cycle. This is done with a basic commitment scheme. Now, if we take a statistically-secret commitment system for this, then we obtain a statistically-secret trapdoor scheme, either in the public parameter or plain model. Also, this trapdoor scheme is statistically simulative (and even perfectly simulative in the public string model if the underlying commitment protocol is perfectly secret). Also, the trapdoor can be reused in several commitments.

An alternative approach which does not require the detour of the reduction to the Directed Hamiltonian Cycle problem is to start with any statistically-secret bit commitment protocol, not necessarily a trapdoor one. The existence of statistically-secret commitment schemes implies that one-way functions exist [IL89], and according to the previous section, we therefore have a non-interactive statistically-binding trapdoor bit commitment scheme in the public parameter model. Alternatively, we may take the interactive equivocable scheme due to [DO99] and obtain an interactive trapdoor commitment scheme. For the presentation here we stick to the non-interactive version in the public parameter model.

Our statistically-secret trapdoor bit scheme is the combination of the computationally-hiding trapdoor scheme and the ordinary statistically-hiding one. That is, in order to commit to a bit $b$ the sender first computes a commitment $B$ of this bit under the trapdoor protocol. This can be done without interaction. Instead of sending this value $B$, the sender and the receiver run the statistically-secret commitment protocol for each bit of $B$ in parallel. In the decommitment phase the sender gives all the random coins and the bit $b$ to the receiver.

Apparently, the combined protocol is statistically secret because the outer protocol hides $B$ statistically. Since the inner scheme is statistically binding, finding ambiguous openings still requires to break the unambiguity of the statistically-secret protocol. However, the assembled protocol inherits the trapdoor property of the inner scheme: if we can output some $B$ which can be opened with any value for $b$, then the transformation of $B$ under the statistically-secret commitment is also openable with any bit $b$. This also implies that the scheme is computationally simulative. Unfortunately, the set up can be exploited only once by the trapdoor simulator to produce a universally openable commitment.

## 3. Identity-Based Trapdoor Commitments

We extend the number-theoretic and complexity-based schemes of the previous sections to identity-based ones. Recall that such identity-based trapdoor commitments link the possibility of finding ambiguous openings with the trapdoor to a special given identifier, say, a user's name. Even if someone else knows this trapdoor information and the special identifier, faking the opening of a commitment for some other identifier is still infeasible or impossible (e.g., for a different login name).

While the constructions of such identity-based trapdoor commitments under specific assumptions are perfectly simulative, the solutions using one-way functions are merely computationally simulative.

### 3.1. Number-Theoretic Constructions

Our solutions rely once more on the discrete-logarithm, RSA and factoring assumption. We remark that, besides the message, the commitment now also depends on an identifier id which is sent in clear together with the actual commitment of the message.

**Discrete-Logarithm.** For the discrete-logarithm setting the public parameters consist of a group $G_q \subseteq \mathbb{Z}_p^*$ of prime order $q$ generated by some $g_1$ and of two further generators $g_2, g_3$ of $G_q$. To commit to $m \in \mathbb{Z}_q$ with $\mathrm{id} \in \{0,1\}^s \subseteq \mathbb{Z}_q$ the sender picks $r \in_{\mathrm{R}} \mathbb{Z}_q$ and computes

$$M := \left(g_1^{\mathrm{id}} g_2\right)^m g_3^r$$

and sends $M$ together with id to the receiver.

To set up the identity-based trapdoor the simulator picks $G_q \subseteq \mathbb{Z}_p^*$ and $g_1, g_3$ at random. Given the special identifier $\mathrm{id}_0 \in \mathbb{Z}_q$ the simulator selects $x \in_{\mathrm{R}} \mathbb{Z}_q$ and computes $g_2$ as $g_2 := g_1^{-\mathrm{id}_0} g_3^x$. With this choice the public parameters are distributed independently of $\mathrm{id}_0$, and because $g_1^{\mathrm{id}_0} g_2 = g_3^x$ and $x = \log_{g_1^{\mathrm{id}_0} g_2} g_3$ it is easy to adapt a decommitment to $M := g_3^{r_0}$ for message $m_0 = 0$ to any other message $m$ for this value $\mathrm{id}_0$. Altogether, the scheme is perfectly simulative.

The trapdoor property is linked to $\mathrm{id}_0$. That is, given the trapdoor $(\mathrm{id}_0, x)$ it is still infeasible to find a commimtent $M = (g_1^{\mathrm{id}} g_2)^m g_3^r$ and ambiguous decommitments $(m, r), (m', r')$ for the same id different than $\mathrm{id}_0$. Because this would imply that

$$\left(g_1^{\mathrm{id}} g_2\right)^m g_3^r = M = \left(g_1^{\mathrm{id}} g_2\right)^{m'} g_3^{r'}$$

or equivalently,

$$g_1^{(\mathrm{id} - \mathrm{id}_0)(m-m')} = g_3^{(r'+xm')-(r+xm)}.$$

Since $\mathrm{id} - \mathrm{id}_0, m - m' \neq 0 \bmod q$ one can easily compute $\log_{g_1} g_3$, contradicting the discrete-log assumption.

**RSA.** For the RSA version the public parameters include a modulus $N$, a prime exponent $e$ as in the basic RSA case and two random elements $g, h \in \mathbb{Z}_N^*$. A commitment to message $m \in \mathbb{Z}_e$ with $\mathrm{id} \in \mathbb{Z}_e$ is given by

$$M := (g^{\mathrm{id}} h)^m r^e \bmod N.$$

The trapdoor simulator selects $N, e$ and $g$ as before and to some $\mathrm{id}_0 \in \{0,1\}^s \subseteq \mathbb{Z}_e$ it computes $h := g^{-\mathrm{id}_0} x^e$ for random $x \in_{\mathrm{R}} \mathbb{Z}_N^*$. Then the simulator knows the $e$-th root $x$ of $g^{\mathrm{id}_0} h = x^e$ and is able to decommit accordingly for this value $\mathrm{id}_0$. The scheme is perfectly simulative.

Given $(\mathrm{id}_0, x)$, distinct valid openings $(m, r), (m', r')$ for the same commitment $M$ and some $\mathrm{id} \neq \mathrm{id}_0$ yield the $e$-th root of $g$: we have

$$\left(g^{\mathrm{id}} h\right)^m r^e = M = \left(g^{\mathrm{id}} h\right)^{m'} (r')^e$$

and therefore

$$g^{(\mathrm{id} - \mathrm{id}_0)(m-m')} = (x^{m'-m} r^{-1} r')^e.$$

Since $\mathrm{id} - \mathrm{id}_0, m - m' \neq 0 \bmod e$ one can easily compute an $e$-th root of $g$.

**Factoring.** The factoring-based system requires $(N, \tau, t, g)$ as in the basic scheme and another random $2^{\tau+t}$-th power $h \in \mathbb{Z}_N^*$. The commitment to $\mathrm{id} \in \mathbb{Z}_{2^t}$ and message $m \in \mathbb{Z}_{2^t}$ is given by

$$M := (g^{\mathrm{id}} h)^m r^{2^{\tau+2t}} \bmod N$$

for random $r \in_{\mathrm{R}} \mathbb{Z}_N^*$. Note that we raise $r$ to the $2^{\tau+2t}$-th power, not the $2^{\tau+t}$-th power.

To build in a trapdoor the simulator chooses $(N, \tau, t, g)$ as before and produces $h$ as $h := g^{-\mathrm{id}_0} x^{2^{\tau+2t}} \bmod N$ for the given $\mathrm{id}_0$. By this, the simulator obtains a $2^{\tau+2t}$-th root $x$ of $g^{\mathrm{id}_0} h = x^{2^{\tau+2t}}$. On the other hand, for $\mathrm{id} \neq \mathrm{id}_0$ a commitment

$$(g^{\mathrm{id}} h)^m r^{2^{\tau+2t}} = M = (g^{\mathrm{id}} h)^{m'} (r')^{2^{\tau+2t}}$$

for different $m \neq m'$ reveals a root of $g$:

$$g^{(\mathrm{id} - \mathrm{id}_0)(m-m')} = (x^{m'-m} r^{-1} r')^{2^{\tau+2t}}.$$

As $\mathrm{id} - \mathrm{id}_0, m - m' \neq 0 \bmod 2^t$ and the absolute value of the product is less than $2^{2t}$ the fact that this gives a root of $g$ and the factorization of $N$ with probability at least $1/2$ follows as in the basic case.

### 3.2. Complexity-Based Construction

Recall the trapdoor commitment scheme in Section 2 where the public string contains a uniformly distributed string $\sigma \in \{0,1\}^{3n}$ and where the sender transmits $G(r)$ or $G(r) \oplus \sigma$, and the simulator sets $\sigma$ as $G(r_0) \oplus G(r_1)$ and commits by $G(r_0)$.

The public parameters in our identity-based trapdoor commitment scheme are $2s$ uniformly chosen bit strings $\sigma_{i,a} \in_{\mathrm{R}} \{0,1\}^{3n+s}$ for $i = 1, 2, \ldots, s$ and $a \in \{0,1\}$. Additionally, suppose that there is pseudorandom generator stretching $n$ to $3n+s$ bits. To commit to a bit $b$ the sender first picks $\mathrm{id} \in \{0,1\}^s$. This determines a $(3n+s)$-bit string

$$\sigma := \bigoplus_{i=1}^{s} \sigma_{i,\mathrm{id}_i}$$

where $\mathrm{id}_i$ denotes the $i$-th bit of id. To commit to $b$ now select a random $r \in_{\mathrm{R}} \{0,1\}^n$, compute $y := G(r)$ for $b = 0$ and $y := G(r) \oplus \sigma$ for $b = 1$. Transmit the pair $(\mathrm{id}, y)$.

This new schemes inherits the basic properties of Naor's protocol. Namely, it is computationally hiding and, since the probability that there exist $(\mathrm{id}, y)$ with ambiguous decommitments is at most $2^{2n+s}/2^{3n+s} = 2^{-n}$ over the choice of the $\sigma_{i,a}$'s, the scheme is statistically binding.

The trapdoor installation is similar to the basic case. Given $\mathrm{id}_0 \in \{0,1\}^s$ in advance, choose all $\sigma_{i,a}$'s at random except for $\sigma_{s,\mathrm{id}_{0,s}}$. This value is chosen as

$$\sigma_{s,\mathrm{id}_{0,s}} := G(r_0) \oplus G(r_1) \oplus \bigoplus_{i=1}^{s-1} \sigma_{i,\mathrm{id}_{0,i}}$$

for random $r_0, r_1 \in_{\mathrm{R}} \{0,1\}^n$. The simulater commits to a dummy bit by sending $(\mathrm{id}_0, G(r_0))$ and is able to open this commitment both for $b = 0$ and $b = 1$. For any $\mathrm{id} \neq \mathrm{id}_0$, however, the same statistical binding property as before holds. Since the way how the simulator prepares the $\sigma_{i,a}$'s is indistinguishable from a truly random choice, the scheme is computationally simulative.

A drawback of the previous solution is that the trapdoor can only be used once. A more complex solution using the Hamiltonian cycle approach establishes a reusable trapdoor. We next outline this construction.

Let $G$ be a pseudorandom generator expanding $n$ bits to, say, $3n$ bits; the output length must be large enough to ensure that a random string of the same length is in the range of the generator with negligible probability only.

Similar to the reduction of the image $f(x)$ of a one-way function $f$ to a graph $H$ in Section 2, we can map a $3n$-bit string (either truly random or an image of the generator $G$) to a graph $H$. The corresponding language problem is to decide if a graph has a directed Hamiltonian cycle. In particular, if the string is random then, unless this string accidentally falls into the range of the pseudorandom generator $G$, the derived graph does not contain a cycle. Conversely, if the string is pseudorandom then the graph has a cycle, and a cycle can be computed from the reduction if a preimage of the string under $G$ is known. Furthermore, if one is given a graph that resulted either from a pseudorandom or a truly random string, it is infeasible to decide with significant advantage if there is a cycle. Else one could distinguish the generator's output from random.

The identity-based trapdoor bit commitment scheme works as follows. Generate $2s$ graphs $H_{i,a}$ for $i = 1, \ldots, s$ and $a \in \{0,1\}$ by picking $2s$ random $3n$-bit strings and reducing each string to a graph problem. Put these graphs into the public string.

To commit to a bit $b$ under identity $\mathrm{id} \in \{0,1\}^s$ the sender takes $H_{i,\mathrm{id}_i}$ for $i = 1, \ldots, s$ and uses each graph to commit to $b$ with the aforementioned Feige-Shamir protocol [FS89]. Namely, for each $i$ the sender commits to a random permutation of $H_{i,\mathrm{id}_i}$ if $b = 0$ and to a trivial graph containing a random Hamiltonian cycle if $b = 1$. The sender transmits all these $s$ commitments to the receiver, together with the identity. Decommitting is done accordingly, in particular, the receiver checks that the sender has taken the right graph $H_{i,\mathrm{id}_i}$ of the pair $(H_{i,0}, H_{i,1})$ and that each of the $s$ decommitments is valid and for the same bit $b$. Depending on the kind of commitment used for committing to the adjacency matrices, the overall scheme is either statistically binding and computationally secret, or it is computationally binding and statistically (or even perfectly) secret.

How do we incorporate a trapdoor? For given $\mathrm{id}_0 \in \{0,1\}^s$ we generate the graph $H_{i,\mathrm{id}_{0,i}}$ by running the pseudorandom generator $G$ on a random string $r_i$, and reducing the output to derive $H_{i,\mathrm{id}_{0,i}}$. Note that $r_i$ yields a Hamiltonian cycle in $H_{i,\mathrm{id}_{0,i}}$. We generate the other graphs from randomly chosen strings.

For identity $\mathrm{id}_0$ knowledge of a cycle in each $H_{i,\mathrm{id}_{0,i}}$ allows to adapt commitments, whereas for $\mathrm{id} \neq \mathrm{id}_0$ at least one graph $H_{i,\mathrm{id}_i}$ contains no cycle with overwhelming probability and thus provides unambiguity of the whole commitment. On the other side, generating the graphs $H_{i,\mathrm{id}_{0,i}}$ "pseudorandomly" is indistinguishable from the real initialization. Therefore, the scheme is computationally simulative, but the trapdoor for identity $\mathrm{id}_0$ is reusable.

We remark that both approaches in this section can be turned into protocols in the plain model, requiring no public parameters. This is accomplished as in the case of basic trapdoors by using interactive coin-flipping protocols to generate the strings $\sigma_{i,a}$ or the strings from which the graphs $H_{i,a}$ are derived, respectively. That is, the receiver first commits to a random string $\alpha_{i,a}$, the sender transmits

random strings $\beta_{i,a}$ and the receiver opens $\alpha_{i,a}$. The outcome is $\sigma_{i,a} = \alpha_{i,a} \oplus \beta_{i,a}$ (or the string which determines the graph $H_{i,a}$). By rewinding the execution, the trapdoor simulator is able to bias the outcome as desired.

*Chapter 4*

# Efficient Non-Malleable Commitment Schemes

In this chapter we discuss non-malleable commitment schemes. An extended abstract of some of the results has been published in [FF00]. This version here does not include the part about the proof of knowledge using the Chinese Remainder Theorem, and, in particular, the improvement from RSA to factoring mentioned in [FF02]. The chapter should be intelligible without the other parts of the thesis, provided that the reader is familiar with the concept of commitments.

We are indebted to Cynthia Dwork for discussions about non-malleability. We also thank the participants of the Luminy 1999 crypto workshop for stimulating discussions, as well as the Crypto 2000 reviewers and program committee, especially Shai Halevi. We are also grateful to Yehuda Lindell and to Jonathan Katz for informing us about their works. Finally, we thank Rosario Gennaro, Tal Rabin and Alon Rosen for discussions and pointing out a gap in the main proof, and Claus Schnorr for drawing our attention to problems with the previously given definitions of non-malleability in the proceedings version of our paper.

## 1. Introduction

Loosely speaking, a commitment scheme is non-malleable if one cannot transform the commitment of another person's secret into one of a related secret. Such non-malleable schemes are for example important for auctions over the Internet: it is necessary that one cannot generate a valid commitment of a bid $b + 1$ after seeing the commitment of an unknown bid $b$ of another participant. Unfortunately, this property is not achieved by commitment schemes in general, because ordinary schemes are only designated to hide the secret. Even worse, most known commitment schemes are in fact provably malleable.

The concept of non-malleability has been introduced by Dolev et al. [DDN00]. They present a non-malleable public-key encryption scheme (based on any trapdoor permutation) and a non-malleable commitment scheme with logarithmically many rounds based on any one-way function. Yet, their solutions involve cumbersome non-interactive and interactive zero-knowledge proofs, respectively. Further non-malleable encryption schemes with improved efficiency under various assumptions have appeared since then [BR93**,** BR94**,** CS98]. As for commitment protocols, Di Crescenzo et al. [DIO98] present a non-interactive and non-malleable commitment scheme based on any one-way function in the common random string model. Though being non-interactive, their system is rather theoretical as it excessively applies an ordinary commitment scheme to non-malleably commit to a single bit. Other non-malleable commitment protocols have been suggested after the proceedings version of our paper [FF00] had been published; we review these schemes at the end of this introduction.

Here, we present efficient perfectly-secret non-malleable commitment schemes based on standard assumptions, such as the RSA assumption or the hardness of computing discrete logarithms. Our schemes are designed in the public parameter model (aka. auxilary string model). That is, public parameters like a random prime $p$ and generators of some subgroup of $\mathbb{Z}_p^*$ are generated and published by a trusted party. We stress that, in contrast to public-key infrastructure, this model does not require the participants to put any trapdoor information into the parameters. The public parameter model relies on a slightly stronger assumption than the common random string model. Yet, the difference is minor as modern networks are likely to provide public parameters for standard crypto systems. Moreover, as for the example of the discrete logarithm, the public parameter model can be formally reduced to the common random string model if we let the participants map the random string via standard procedures to a prime and appropriate generators.

In our schemes the sender commits to his message using an ordinary, possibly malleable discrete-log- or RSA-based commitment scheme and performs an efficient three-round witness-independent proof of knowledge, both times using the public parameters. While the straightforward solution of a standard proof of knowledge fails (because the adversary may in addition to the commitment also transform the proof of knowledge), we force the adversary to give his "own" proof of knowledge without being able to adapt the one of the original sender. Similar ideas have also been used in [DDN00]. In our case, the proof of knowledge guarantees that the adversary already knows the message he has committed to. This means that he is aware of some information about the related message of the original sender, contradicting the secrecy property of the ordinary commitment scheme.

We also address definitional issues. According to the definition of Di Crescenzo et al. [DIO98], a scheme is non-malleable if the adversary cannot construct a commitment from a given one, such that after having seen the opening of the original commitment, the adversary is able to correctly open his commitment with a related message. But the definition of Dolev et al. [DDN00] demands more: if there is a one-to-one correspondence between the commitment and the message (say, if the commitment binds unconditionally), then they define that such a scheme is non-malleable if one cannot even generate a commitment of a related message. We call schemes having the latter property *non-malleable with respect to commitment*. For these schemes to contradict non-malleability it suffices to come up with a commitment such that there exists a related opening. Schemes satisfying the former definition are called *non-malleable with respect to decommitment* or, for sake of distinctiveness, *with respect to opening*. In this case, the adversary must also be able to open the modified commitment correctly given the decommitment of the original commitment. The scheme in [DDN00] achieves the stronger notion, whereas we do not know if the scheme in [DIO98] is also non-malleable with respect to commitment.

A commitment scheme which is non-malleable in the strong sense is non-malleable with respect to opening, too.[1] We stress that the other direction does not hold in general. That is, given a statistically-secret commitment scheme which is secure with respect to opening, we can devise a commitment scheme satisfying the weak notion, but not the strong definition. Since our statistically-secret schemes based on standard assumptions like RSA or discrete-log achieve non-malleability with respect to opening, both notions are *not* equivalent under any of these standard assumptions.

We believe that non-malleability with respect to opening is the appropriate notion for perfectly- and statistically-secret schemes. The reason is that for such schemes virtually any commitment can be opened with any message in principle. Hence, finding a commitment of a related message to a given commitment is easy: any valid commitment works with very high probability. Although there is at least one application of non-malleable commitment schemes in the context of authenticated key-exchange where non-malleability with respect to commitment is necessary [GL01], non-malleability with respect to opening still seems to be adequate for most applications. For instance, recall the example of Internet auctions. The commitments of the bids are collected and then, after a deadline has passed, are requested to be opened. Any secret which is not correctly revealed is banned. Therefore, security with respect to opening suffices in this setting.

[1] Although this seems to follow directly from the requirements, it heavily depends on the subtleties of the definitions. Indeed, compared to [DDN00], we strengthen the requirements for non-malleablity with respect to commitment in order to imply the notion of non-malleability with respect to opening. The scheme in [DDN00] also satisfies our more stringent definition.

Following the publication of the proceedings version of our work [FF00], several other non-malleable commitment schemes have been proposed. Di Crescenzo et al. [DKOS01] present more practical variants of the system in [DIO98] relying on the RSA or discrete-log assumption and the public parameter model; see also [FF02] for improved versions of these protocols resulting in more efficient schemes than the ones here. These schemes achieve perfect secrecy and non-malleability with respect to opening. Yet, in contrast to our solution here, all these protocols are not known to provide non-malleability if the adversary is additionally given some useful side information about the message for which it tries to find a related commitment, e.g., if the message is used in other subprotocol executions.

In [DKOS01] it is also pointed out that secure public-key encryption is sufficient for non-malleable commitments. Basically, the public parameters contain a public key of a secure encryption scheme and in order to commit the sender encrypts the message and hands it to the receiver. Hence, using potentially stronger assumptions like the decisional Diffie-Hellman assumption and the encryption scheme in [CS98], or non-standard assumptions like the random oracle methodology, one derives alternatives to the solutions here and in [DKOS01, FF02]. Yet, the encryption-based approach provides only computational secrecy and the latter may be insufficient in some settings, especially since knowledge of the secret key to the public key from the public parameters enables to decrypt the message. Also, using random oracles there is a simpler approach to accomplish non-malleable commitments. We sketch this solution in Section 6. More non-malleable (but less efficient) commitment schemes in the broader context of universally composable commitments have been constructed by Canetti and Fischlin [CF01] and subsequently by Damgård and Nielsen [DN01]. We discuss the protocols of Canetti and Fischlin in Chapter 6 of this thesis.

The chapter is organized as follows. In Section 2 we define non-malleable commitment schemes. Section 3 separates the notions of non-malleability with respect to commitment and opening. In Section 4 we present efficient schemes in the public parameter model based on the discrete-log assumption, and in Section 5 we turn to the RSA case. Finally, in Section 6 we show how to construct non-malleable schemes in the random oracle model.

## 2. Non-Malleability

As mentioned in the introduction, different notions of non-malleability have been used implicitly in the literature. To highlight the difference we give a formal definition of non-malleable commitment schemes, following the approach of [DDN00].

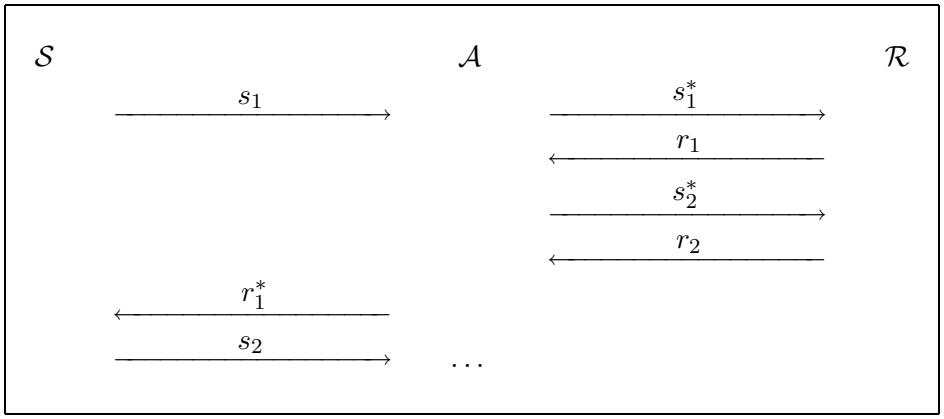
### 2.1. Scenario

For non-interactive commitment schemes, all the adversary can do is modify a given commitment. In the interactive case, though, the adversary might gain

advantage from the interaction. We adopt this worst-case scenario and assume that the adversary interacts with the original sender, while at the same time he is trying to commit to a related message to the original receiver.

A pictorial description of a so-called *person-in-the-middle attack* (PIM attack) on an interactive protocol is given in Figure 1. The adversary $\mathcal{A}$ intercepts the messages of the sender $\mathcal{S}$. Then $\mathcal{A}$ may modify the messages before passing them to the receiver $\mathcal{R}$ and proceeds accordingly with the answers. In particular, $\mathcal{A}$ decides to whom he sends the next message, i.e., to the sender or to the receiver. This is the setting where $\mathcal{A}$ has full control over the parties $\mathcal{R}_1$ and $\mathcal{S}_2$ in two supposedly independent executions $\langle \mathcal{S}_1, \mathcal{R}_1 \rangle(m)$, $\langle \mathcal{S}_2, \mathcal{R}_2 \rangle(m^*)$ of the same interactive protocol. Here and in the sequel, we usually mark values sent by the adversary with an asterisk.

FIGURE 1. Person-In-The-Middle Attack on Interactive Protocols




Apparently, the adversary can always commit to the same message by forwarding the communication. In many applications, this can be prevented by letting the sender append his identity to the committed message. The messages of the sender and the adversary are taken from a space $\mathsf{M}$. Abusing notations, we view $\mathsf{M}$ also as an efficiently computable distribution, and write $m \in_{\mathrm{R}} \mathsf{M}$ for a randomly drawn message according to $\mathsf{M}$.

The adversary is deemed to be successful if he commits to a related message, where related messages are identified by so-called interesting relations: a probabilistic polynomial-time algorithm $\mathsf{R}$ taking inputs from $\mathsf{M} \times \mathsf{M}$ and returning a bit is called an *interesting relation* if $\mathsf{R}(m, m) = 0$ with probability 1 for all $m \in \mathsf{M}$ (to exclude copying). Moreover, we let the interesting relation on the second argument accept the undefined symbol $\perp$, capturing the case that the adversary does not produce a valid commitment or decommitment; in this case we set $m^* = \perp$ and we demand $\mathsf{R}(m, \perp) = 0$ with probability 1.

We assume that $\mathsf{M}$ generates the sender's message $m$ and also a value $\mathrm{Hist}(m)$ representing the a-priori information that the adversary has about $m$. For instance, $\mathrm{Hist}(m)$ could represent an additional hash value of the sender's message $m$, or information gathered from other protocol executions where the sender uses $m$. In comparison to [DDN00] where Hist is a separate function, attributing Hist to $\mathsf{M}$ admits an easy way to include information about the sampling process of $m$ into $\mathrm{Hist}(m)$. For ease of notation we write both $m \in_{\mathrm{R}} \mathsf{M}$ and $(m, \mathrm{Hist}(m)) \in_{\mathrm{R}} \mathsf{M}$.

Since we work in the public parameter model, we extend the input of $\mathsf{M}$ and $\mathsf{R}$ by adversarial parameters AdvPar that the adversary produces after having learned the the public parameters PubPar. The value AdvPar may for example include the public parameters PubPar. The motivation for this is that it should be infeasible for the adversary to find a suitable relation or distribution on the messages even if the publicly available parameters are given. For the same reason, we base the relation $\mathsf{R}$ also on the side information $\mathrm{Hist}(m)$. In summery, we denote the message space and distribution as $\mathsf{M}(\mathrm{AdvPar})$ and the relation by $\mathsf{R}(\mathrm{AdvPar}, \mathrm{Hist}(m), \cdot, \cdot)$.

### 2.2. Definition

The definition on non-malleable commitments follows the well-known idea of defining secure encryption, namely, we will demand that for any adversary $\mathcal{A}$ transforming the sender's commitment successfully, there should be an adversary $\mathcal{A}'$ that finds a commitment to a related message with almost the same probability as $\mathcal{A}$ but without the sender's help.

We describe the attack in detail. First, the public parameters PubPar are generated by a trusted party according to a publicly known, efficiently samplable distribution (if a protocol does not need public information then this step is skipped). On input PubPar the adversary $\mathcal{A}$ then picks the adversarial parameters AdvPar for $\mathsf{M}$ and $\mathsf{R}$. The sender $\mathcal{S}$ is initialized with $m \in_{\mathrm{R}} \mathsf{M}(\mathrm{AdvPar})$. Now $\mathcal{A}$, given $\mathrm{Hist}(m)$, mounts a PIM attack with $\mathcal{S}(m)$ and $\mathcal{R}$. Let $\pi_{\mathsf{com}}(\mathcal{A}, \mathsf{M}, \mathsf{R})$ denote the probability that, at the end of the commitment phase, the protocol execution between $\mathcal{A}$ and $\mathcal{R}$ constitutes a valid commitment for some message $m^*$ satisfying $\mathsf{R}(\mathrm{AdvPar}, \mathrm{Hist}(m), m, m^*)$. Let $\pi_{\mathsf{open}}(\mathcal{A}, \mathsf{M}, \mathsf{R})$ denote the probability that $\mathcal{A}$ is also able to successfully open the commitment after $\mathcal{S}$ has decommitted.

In a second experiment, a simulator $\mathcal{A}'$ tries to commit to a related message without the help of the sender. That is, $\mathcal{A}'$ gets as input random public parameters PubPar, generates adversarial parameters $\mathrm{AdvPar}'$ and then, given $\mathrm{Hist}(m)$ for some $(m, \mathrm{Hist}(m)) \in_{\mathrm{R}} \mathsf{M}(\mathrm{AdvPar}')$, it commits to $\mathcal{R}$ without interacting with $\mathcal{S}(m)$. Let $\pi'_{\mathsf{com}}(\mathcal{A}', \mathsf{M}, \mathsf{R})$ denote the probability that this is a valid commitment to some related message $m'$ under public parameters PubPar with respect to relation $\mathsf{R}(\mathrm{AdvPar}', \mathrm{Hist}(m), \cdot, \cdot)$. By $\pi'_{\mathsf{open}}(\mathcal{A}', \mathsf{M}, \mathsf{R})$ we denote the probability

that $\mathcal{A}'$ additionally reveals a correct decommitment. Equivalently, we may define $\pi'_{\mathsf{open}}(\mathcal{A}', \mathsf{M}, \mathsf{R})$ as the probability that $\mathcal{A}'$ simply outputs a related message (without reference to public parameters, commitment and decommitment).

Note that all probabilities are implicit functions of a security parameter. The probability space in each case is taken over the randomness of all algorithms.

It is now tempting to define non-malleability with respect to commitment and with respect to opening by comparing $\pi_{\mathsf{com}}(\mathcal{A}, \mathsf{M}, \mathsf{R})$, $\pi'_{\mathsf{com}}(\mathcal{A}', \mathsf{M}, \mathsf{R})$ as well as $\pi_{\mathsf{open}}(\mathcal{A}, \mathsf{M}, \mathsf{R})$, $\pi'_{\mathsf{open}}(\mathcal{A}', \mathsf{M}, \mathsf{R})$ and asking for small differences. In the former case this would agree with the definition in [DDN00] and in the other case this would extend it straightforwardly to non-malleability with respect to opening. But, surprisingly at first, for non-malleability with respect to commitment we even oblige the simulator to open his commitment and contrast $\pi_{\mathsf{com}}(\mathcal{A}, \mathsf{M}, \mathsf{R})$ with $\pi'_{\mathsf{open}}(\mathcal{A}', \mathsf{M}, \mathsf{R})$. The are two reasons for this. First, otherwise any statistically-secret commitment protocol would be non-malleable with respect to commitment, because if the simulator merely outputs a commitment of some fixed message this is also a commitment of a related message with high probability. However, this would certainly contradict the intuition of non-malleable systems, in particular, since we know provably malleable statistically-secret protocols. The other reason is that, even in the case of statistically-binding schemes, we were unable to show that the presumably stronger non-malleability notion a la [DDN00] implies the weaker one. With our approach here this trivially follows from the definition, because the requirements for the simulator in both cases are identical while the adversary trying to refute non-malleability with respect to commitment even faces a simpler task.

For sake of completeness we include the original definition of Dolev et al. [DDN00] and call this non-malleability with respect to DDN. We remark that the commitment scheme in [DDN00] also satisfies "our" notion of non-malleability with respect to commitment.

**Definition 4.1.** *A commitment scheme is called*

a) *non-malleable with respect to commitment if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{A}'$ such that for any message space $\mathsf{M}$ and any interesting relation $\mathsf{R}$ the difference $\pi_{\mathsf{com}}(\mathcal{A}, \mathsf{M}, \mathsf{R}) - \pi'_{\mathsf{open}}(\mathcal{A}', \mathsf{M}, \mathsf{R})$ is negligible.*[2]

b) *non-malleable with respect to opening if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{A}'$ such that for any message space $\mathsf{M}$ and any interesting relation $\mathsf{R}$ the difference $\pi_{\mathsf{open}}(\mathcal{A}, \mathsf{M}, \mathsf{R}) - \pi'_{\mathsf{open}}(\mathcal{A}', \mathsf{M}, \mathsf{R})$ is negligible.*

[2]Here we allow a very liberal definition of negligible functions: the function may also be negative at some value $n$, in which case it is certainly less than $p(n)$ for any strictly positive polynomial $p(\cdot)$. In our case this means that the simulator does even better than the adversary and, thus, this still reflects our intuition of non-malleability.

c) *non-malleable with respect to DDN if for every adversary* $\mathcal{A}$ *there exists a simulator* $\mathcal{A}'$ *such that for any message space* $\mathsf{M}$ *and any interesting relation* $\mathsf{R}$ *the difference* $\pi_{\mathsf{com}}(\mathcal{A}, \mathsf{M}, \mathsf{R}) - \pi'_{\mathsf{com}}(\mathcal{A}', \mathsf{M}, \mathsf{R})$ *is negligible.*

If $\mathsf{M}$ and $\mathsf{R}$ are clear from the context we usually abbreviate the success probabilities by $\pi_{\mathsf{com}}(\mathcal{A})$, $\pi'_{\mathsf{com}}(\mathcal{A}')$, $\pi_{\mathsf{open}}(\mathcal{A})$ and $\pi'_{\mathsf{open}}(\mathcal{A}')$, respectively.

Some remarks about the experiment of $\mathcal{A}'$ follow. The simulator $\mathcal{A}'$ does not have the power to choose the public parameters PubPar for the commitment to $\mathcal{R}$. This is so because the simulator is obliged to produce a correct commitment to $\mathcal{R}$ under the same honestly chosen public data PubPar as the sender and the adversary. This rules out counterintuitive solutions proving obviously transformable commitments non-malleable. Nonetheless, $\mathcal{A}'$ picks his own AdvPar$'$, not necessarily related to $\mathcal{A}$'s selection AdvPar. But since the relation $\mathsf{R}$ depends on these adversarial parameters AdvPar and AdvPar$'$, it is clear that the relation can rule out significantly diverging choices of $\mathcal{A}'$, and hence AdvPar$'$ is likely to be indistinguishable from AdvPar.

Slightly relaxing the definition, we admit an *expected* polynomial-time simulator $\mathcal{A}'$. In fact, we are only able to prove our schemes non-malleable with this deviation. The reason for this is that we apply proofs of knowledge, so in order to make the success probability of $\mathcal{A}'$ negligibly close to the adversary's success probability, we run a knowledge extractor taking expected polynomial-time.[3] Following the terminology in [DDN00], we call such schemes *liberal non-malleable* with respect to commitment, opening and DDN, respectively.

Consider a computationally-binding and perfectly-secret commitment scheme. There, every valid commitment is correctly openable with every message (it is, however, infeasible to find different messages that work). Thus, we believe that non-malleability with respect to opening is the interesting property in this case. On the other hand, non-malleability with respect to commitment is also a concern for statistically-binding commitment schemes: with overwhelming probability there do not exist distinct messages that allow to decommit correctly. This holds for *any dishonest* sender and, in particular, for the person-in-the-middle adversary. We can therefore admit this negligible error and still demand non-malleability with respect to commitment.

### 2.3. The Multi-Party Setting

Our definition captures the simple setting of three parties. In the auction case, for instance, usually more parties participate and the adversary's intention may be

[3]The same problem occurs in [DDN00]. Alternatively, the authors of [DDN00] also propose a definition of $\epsilon$-malleability, which basically says that for given $\epsilon$ there is a strict polynomial-time simulator (polynomial in the security parameter $n$ and $\epsilon^{-1}(n)$) whose success probability is only $\epsilon$-far from the adversary's probability.

to overbid only a certain opponent to ensure that this person does not win. Hence, we may let $\mathcal{A}$ talk to several senders $\mathcal{S}_1, \ldots, \mathcal{S}_{\text{poly}}$ with (probably dependent) messages $m_1, \ldots, m_{\text{poly}}$ generated by $\mathsf{M}(\text{AdvPar})$ together with side information $\text{Hist}(m_1, \ldots, m_{\text{poly}})$. The relation now takes AdvPar, $\text{Hist}(m_1, \ldots, m_{\text{poly}})$ and poly + 1 messages as input and it is required that the (poly + 1)-st message $m^*$ is different from any other message $m_i$, and that the relation is never satisfied if $m^* = \perp$. We remark that all our protocols remain secure in this multiple-sender setting.

A problem occurs if we let the adversary commit in several executions with $\mathcal{R}$ to messages $m_1^*, \ldots, m_{\text{poly}}^*$ and extend the relation accordingly, both in the single- or multiple-sender case. Dolev et al. [DDN00] show that this scenario is not reducible to the single-adversary case in general and suggest an alternative definition where the adversary is supposed to announce a subset $i_1, \ldots, i_k$ of the executions with the receiver in the commitment phase, inducing a set of messages $m_{i_1}^*, \ldots, m_{i_k}^*$ for which he tries to be successful. We return to the multi-party case at the end of Section 4.3 when discussing this issue for our schemes.

## 3. On the Relationship of Non-Malleability Notions

Clearly, non-malleability with respect to commitment implies non-malleability with respect to opening and with respect to DDN. On the other hand, we show that (under standard cryptographic assumptions) the converse does not hold in the public parameter model. To this end, we construct a bit commitment scheme that does not even achieve the DDN notion, but is non-malleable with respect to opening.

To separate the notions we consider Naor's bit commitment scheme [N91] in the public parameter model. Let $G$ be a pseudorandom generator expanding $n$ bits random input to $3n$ bits pseudorandom output. That is, the variables $(X_n)_{n\in\mathbb{N}}$ and $(Y_n)_{n\in\mathbb{N}}$ are computationally indistinguishable, where $X_n$ equals $G(r)$ for a random $r \in \{0,1\}^n$ and $Y_n$ is the uniform distribution on $\{0,1\}^{3n}$.

Let $\sigma$ be a random $3n$-bit string put into the public parameters. In order to commit to a bit $b$ in Naor's protocol the sender chooses a random $r \in \{0,1\}^n$ and transmits $y = G(r)$ for $b = 0$ or $y = G(r) \oplus \sigma$ if $b = 1$. The decommitment consists of $(b, r)$. Not only is this scheme computationally secret and statistically binding, it is also *strongly malleable*, i.e., given a commitment $y$ of a bit $b$ one can always derive a commitment of $b \oplus 1$ by sending $y \oplus \sigma$.

Next, we construct an assembled commitment scheme (in the public parameter model) which consists of a combination of Naor's scheme and an arbitrary statistically-secret system $\text{Com}_{\text{secret}}$ which is non-malleable with respect to opening. To commit to bit $b$, independently execute the statistically-secret protocol

and Naor's scheme on $b$, either in parallel or sequentially. Opening is done by decommitting for both schemes in parallel.

Obviously, this assembled scheme is computationally secret and statistically binding. We show that this scheme only achieves the weaker non-malleability property. The intuition is that the assembled scheme inherits non-malleability with respect to opening from the statistically-secret protocol, and the strong malleability of Naor's scheme (together with the fact that virtually any statistically-secret commitment is in principle openable with any value) inhibits non-malleability with respect to commitment.

**Theorem 4.2.** *If there is a statistically-secret bit commitment scheme that is non-malleable with respect to opening, then there exists a statistically-binding commitment scheme in the public parameter model that is non-malleable with respect to opening, but not with respect to commitment and not with respect to DDN.*

Theorem 4.2 also holds for liberal non-malleable statistically-secret protocols in the public parameter model.

**Proof.** Since one-way functions exist if commitment schemes exists [IL89], and one-way functions imply pseudorandom generators [HILL99], Naor's scheme and therefore the assembled system above is realizable given the statistically-secret bit commitment scheme.

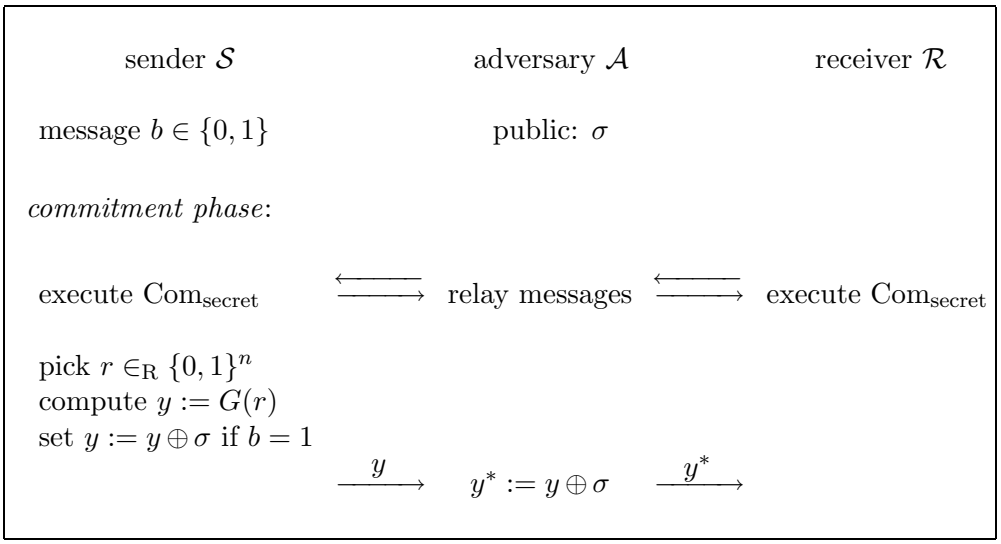
We first show that the assembled scheme is *not* non-malleable with respect to DDN (and therefore not with respect to commitment). Define the relation $\mathsf{R}$ to consist of the pairs $(b, b \oplus 1)$ and the message space to be the uniform distribution on $\{0,1\}$, i.e., both $\mathsf{M}$ and $\mathsf{R}$ are independent of the adversarial parameters. Let $\mathrm{Hist}(b)$ be empty.

Given access to a sender committing to an unknown random bit $b \in_{\mathrm{R}} \{0,1\}$ we run a PIM attack and relay all messages between the receiver and the sender for $\mathrm{Com}_{\mathrm{secret}}$. Additionally, we alter Naor's part of the sender's commitment to a commitment of $b^* = b \oplus 1$ by the strong malleability property and forward it to the receiver (Figure 2).

Since $\mathrm{Com}_{\mathrm{secret}}$ is statistically secret, with overwhelming probability that part of the sender's commitment can be opened as 0 *and* 1. Hence, with probability negligibly close to 1 we are able to construct a valid commitment of $b^* = b \oplus 1$ for the assembled scheme and to satisfy the relation $\mathsf{R}$. On the other hand, any simulator not seeing the commitment of the random bit $b$ cannot output a commitment of $b' = b \oplus 1$ with a probability exceeding $1/2$ by more than a negligible amount (this negligible amount is due to the binding error of Naor's protocol). Thus, the assembled scheme is *not* non-malleable with respect to DDN.

The fact that the combined scheme is non-malleable with respect to opening follows from the non-malleability of the statistically-secret system. Specifically, let

FIGURE 2. Malleability With Respect to Commitment



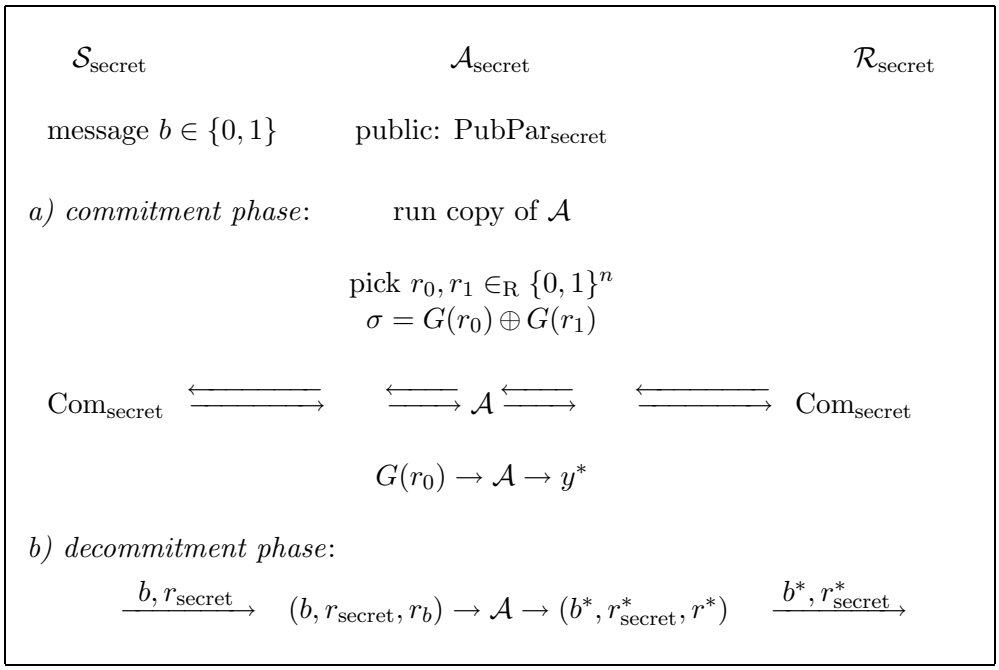

$\mathcal{A}$ be an adversary attacking the assembled system. We have to present a simulator that —"out of the blue"— outputs a related message with essentially the same probability $\pi_{\mathsf{open}}(\mathcal{A})$ as $\mathcal{A}$ for all $\mathsf{M}, \mathsf{R}$. In an intermediate step we construct an adversary $\mathcal{A}_{\mathrm{secret}}$ from $\mathcal{A}$ such that $\mathcal{A}_{\mathrm{secret}}$ attacks the non-malleability property of $\mathrm{Com}_{\mathrm{secret}}$.

Consider the adversary $\mathcal{A}_{\mathrm{secret}}$ that commits and decommits to a related message for the protocol $\mathrm{Com}_{\mathrm{secret}}$. $\mathcal{A}_{\mathrm{secret}}$ mounts a PIM attack interacting with the sender $\mathcal{S}_{\mathrm{secret}}$ and receiver $\mathcal{R}_{\mathrm{secret}}$ of $\mathrm{Com}_{\mathrm{secret}}$ on (possibly empty) public parameters $\mathrm{PubPar}_{\mathrm{secret}}$. $\mathcal{A}_{\mathrm{secret}}$ also runs a virtual copy of $\mathcal{A}$ attacking the assembled scheme. Basically, $\mathcal{A}_{\mathrm{secret}}$ uses $\mathcal{A}$ to generate a related commitment and opening for $\mathrm{Com}_{\mathrm{secret}}$ by adding the steps of Naor's scheme. For this, $\mathcal{A}_{\mathrm{secret}}$ exploits the equivocable version of Naor's scheme presented in [DIO98]. Informally, such an equivocable commitment enables the sender to prepare a dummy commitment which can be later opened with any value, yet this process is indistinguishable from a true execution. This means, instead of letting $\sigma$ be a random string, we choose $\sigma$ as $G(r_0) \oplus G(r_1)$ for random $r_0, r_1 \in \{0,1\}^n$. Then, to commit to a dummy value, send $y = G(r_0)$; to open it with 0 reveal $r_0$ or transmit $r_1$ for a decommitment to 1.

$\mathcal{A}_{\mathrm{secret}}$ emulates $\mathcal{A}$ by choosing $\sigma = G(r_0) \oplus G(r_1)$ and passing $(\mathrm{PubPar}_{\mathrm{secret}}, \sigma)$ to $\mathcal{A}$. Adversary $\mathcal{A}$ returns parameters AdvPar which $\mathcal{A}_{\mathrm{secret}}$ uses in his attack on $\mathrm{Com}_{\mathrm{secret}}$, too. This defines a distribution $\mathsf{M}(\mathrm{AdvPar})$ on $\{0,1\}$ as well as a relation $\mathsf{R}(\mathrm{AdvPar}, \cdot, \cdot, \cdot)$ for both $\mathcal{A}$'s and $\mathcal{A}_{\mathrm{secret}}$'s attack. $\mathcal{A}_{\mathrm{secret}}$ next feeds all messages of $\mathcal{S}_{\mathrm{secret}}$ and $\mathcal{R}_{\mathrm{secret}}$ of the execution of $\mathrm{Com}_{\mathrm{secret}}$ into $\mathcal{A}$

and also forwards all replies of $\mathcal{A}$. Additionally, $\mathcal{A}_{\text{secret}}$ submits a dummy commitment $y = G(r_0)$ on behalf of the sender to $\mathcal{A}$ in the simulation. Later, when $\mathcal{A}_{\text{secret}}$ learns $\mathcal{S}_{\text{secret}}$'s decommitment of bit $b$ it forwards this decommitment to $\mathcal{A}$ and opens the dummy commitment $y$ in $\mathcal{A}$'s simulation accordingly. Output the part of $\mathcal{A}$'s opening for $\text{Com}_{\text{secret}}$ and stop. See Figure 3.

FIGURE 3. Non-Malleability With Respect to Opening




As for the analysis, first note that $\mathcal{A}_{\text{secret}}$'s success probability poducing a valid commitment and decommitment of a related messages is negligibly close to $\pi_{\mathsf{open}}(\mathcal{A})$. This follows from the fact that a fake $\sigma$ is indistinguishable from a honestly chosen one, i.e., otherwise it would be easy to derive a successful distinguisher contradicting the pseudorandomness of $G$'s output.

More formally, assume that $\mathcal{A}$'s success probability drops noticeably when run on a fake string in the simulation. Then we construct a distinguisher for the pseudorandom generator $G$ as follows. We are given $1^n$ and $z \in \{0,1\}^{3n}$ and are supposed to tell whether $z$ is truly random or has been derived by running $G$. Pick random $r \in \{0,1\}^n$ and set $\sigma = G(r) \oplus z$. Next, start $\mathcal{A}$'s attack on the assembled scheme by presenting $(\text{PubPar}_{\text{secret}}, \sigma)$. Sample $(b, \text{Hist}(b))$ according to the distribution $\mathsf{M}(\text{AdvPar})$ and continue $\mathcal{A}$'s attack by impersonating the honest parties in the execution of $\text{Com}_{\text{secret}}$. Also, let the sender commit in Naor's protocol execution by sending $y = G(r)$ if $b = 0$ and $z$ if $b = 1$. In the opening

phase, decommit to this part by revealing $(b, r)$. Output 1 exactly if $\mathcal{A}$ succeeds, that is, if $\mathsf{R}(\text{AdvPar}, \text{Hist}(b), b, b^*) = 1$ for a valid opening of $\mathcal{A}$ to $b^*$.

Observe that if $z$ is really random we output 1 with probability $\pi_{\mathsf{open}}(\mathcal{A})$, because the distribution of the data in the simulation is the same as in an actual attack on the assembled scheme. If $z$ is pseudorandom then we output 1 with the probability that $\mathcal{A}_{\text{secret}}$ is victorious. By assumption, this is noticeably smaller than $\pi_{\mathsf{open}}(\mathcal{A})$, and therefore we distinguish random and pseudorandom inputs with noticeable advantage. This, however, refutes the pseudorandomness of $G$.

Altogether, we have constructed an adversary $\mathcal{A}_{\text{secret}}$ that succeeds in attacking $\text{Com}_{\text{secret}}$ for public parameters $\text{PubPar}_{\text{secret}}$ virtually with the same probability that $\mathcal{A}$ succeeds in attacking the assembled scheme on $\text{PubPar}_{\text{secret}}$ and truly random $\sigma$. By assumption, for $\mathcal{A}_{\text{secret}}$ there is a simulator $\mathcal{A}'_{\text{secret}}$ succeeding in outputting a related message essentially with the same probability as $\mathcal{A}_{\text{secret}}$. But then this algorithm $\mathcal{A}'_{\text{secret}}$ is also an appropriate simulator for adversary $\mathcal{A}$ attacking the assembled scheme. □

Applying our constructions we conclude:

**Corollary 4.3.** *Under the discrete-log or RSA assumption, there is an interactive commitment scheme in the public parameter model that is liberal non-malleable with respect to opening, but not with respect to commitment and not with respect to DDN.*

## 4. Discrete-Logarithm-Based Non-Malleable Commitments

In this section we introduce our discrete-log based commitment schemes which are non-malleable with respect to opening; the RSA case is discussed in Section 5.

In Section 4.1 we start with an instructive attempt to achieve non-malleability by standard proof-of-knowledge techniques. We show that this approach yields a scheme which is only non-malleable with respect to opening against static adversaries, i.e., adversaries that try to find a commitment after passively observing a commitment between the original sender and receiver. In Section 4.2 we develop out of this our scheme which is non-malleable against the stronger PIM adversaries. The formal proof of non-malleability appears in Section 4.3.

### 4.1. Non-Malleability with Respect to Static Adversaries

Consider Pedersen's well-known discrete-log-based perfectly-secret scheme [P91]. Let $G_q$ be a cyclic group of prime order $q$ and $g_0, h_0$ two random generators of $G_q$. Assume that computing the discrete logarithm $\log_{g_0} h_0$ is intractable (e.g., if $G_q$ is an appropriate elliptic curve or subgroup of $\mathbb{Z}_p^*$). To commit to a message $m \in \mathbb{Z}_q$, choose $r \in_{\text{R}} \mathbb{Z}_q$ and set $M := g_0^m h_0^r$. To open this commitment, reveal $m$ and $r$. Obviously, the scheme is perfectly secret as $M$ is uniformly distributed in

$G_q$, independently of the message. It is computationally binding because opening a commitment with distinct messages requires computing $\log_{g_0} h_0$.

Unfortunately, Pedersen's scheme is malleable: given a commitment $M$ of some message $m$ an adversary obtains a commitment for $m+1 \bmod q$ by multiplying $M$ with $g$. Later, the adversary reveals $m+1 \bmod q$ and $r$ after learning the original decommitment $m, r$. This holds even for static adversaries. Such adversaries do not try to inject messages in executions, but rather learn a protocol execution of $\mathcal{S}$ and $\mathcal{R}$ —which they cannot influence— and afterwards try to commit to a related message to $\mathcal{R}$. As for non-malleability with respect to opening, the adversary must also be able to open the commitment after the sender has decommitted.

A possible fix that might come to one's mind are proofs of knowledge showing that the sender actually knows the message encapsulated in the commitment. For the discrete-log case such a proof of knowledge consists of the following steps [O92]: the sender transmits a commitment $S := g_0^s h_0^t$ of a random value $s \in_R \mathbb{Z}_q$, the receiver replies with a random challenge $c \in_R \mathbb{Z}_q$ and the sender answers with $y := s + cm \bmod q$ and $z := t + cr \bmod q$. The receiver finally checks that $SM^c = g_0^y h_0^z$.

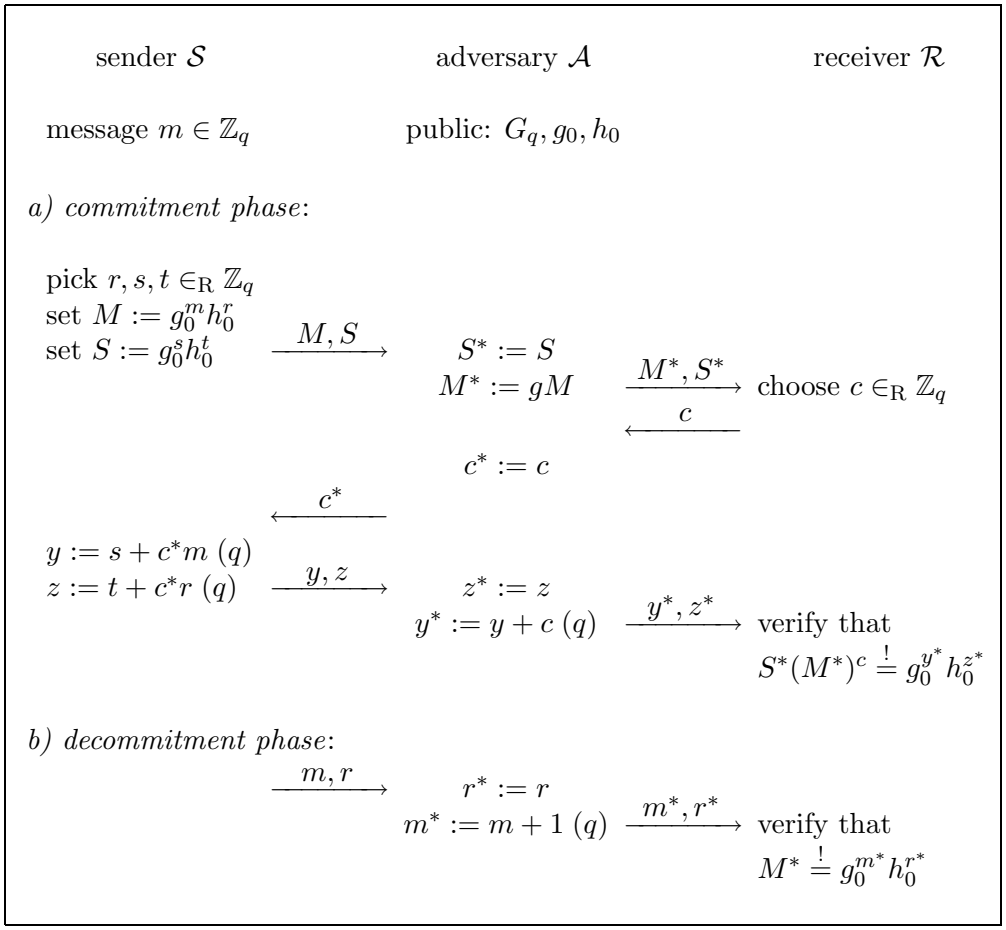
If we add a proof of knowledge to Pedersen's scheme we obtain a protocol which is non-malleable with respect to opening against static adversaries. This follows from the fact that any static adversary merely sees a commitment of an unknown message before trying to find an appropriate commitment of a related message. Since the proof of knowledge between $\mathcal{S}$ and $\mathcal{R}$ is already finished at this point, the static adversary cannot rely on the help of $\mathcal{S}$ and transfer the proof of knowledge. We leave further details to the reader and focus instead on the non-malleable protocol against PIM adversaries in the next section.

### 4.2. Non-Malleability with Respect to PIM Adversaries

The technique of assimilating a proof of knowledge as in the previous section does not thwart PIM attacks. Consider again the PIM adversary committing to $m+1 \bmod q$ by multiplying $M$ with $g$. First, this adversary forwards the sender's commitment $S$ for the proof of knowledge to the receiver and hands the challenge $c$ of the receiver to the sender. Conclusively, he modifies the answer $y, z$ of the sender to $y^* := y + c \bmod q$ and $z^* := z$. See Figure 4. Clearly, this is a valid proof of knowledge for $m+1 \bmod q$ and this PIM adversary successfully commits and later decommits to a related message.

Coin-flipping comes to rescue. In a coin flipping protocol one party commits to a random value $a$, then the other party publishes a random value $b$, and finally the first party decommits to $a$. The result of this coin flipping protocol is set to $c := a \oplus b$ or, in our case, to $c := a + b \bmod q$ for $a, b \in \mathbb{Z}_q$. If at least one party is honest, then the outcome $c$ is uniformly distributed (if the commitment scheme is binding and secret).

FIGURE 4. PIM Attack on Pedersen's Commitment Scheme with Proof of Knowledge




The idea is now to let the challenge in our proof of knowledge be determined by such a coin-flipping protocol. But if we also use Pedersen's commitment scheme with the public generators $g_0, h_0$ to commit to value $a$ in this coin-flipping protocol, we do not achieve any progress: the adversary might be able to commit to a related $a^*$ and thus bias the outcome of the coin-flipping to a suitable challenge $c^*$.

The solution is to apply Pedersen's scheme in this subprotocol with the commitment $M$ as one of the generators, together with an independent generator $h_1$ instead of $g_0, h_0$; for technical reasons we rather use $(g_1 M)$ and $h_1$ for another generator $g_1$. As we will show, since the coin-flipping in the proof of knowledge between $\mathcal{A}$ and $\mathcal{R}$ is based on generators $g_1 M^*$ and $h_1$ instead of $g_1 M, h_1$ as in the sender's proof of knowledge, this prevents the adversary from adapting the sender's and receiver's values and therefore to transfer the proof of knowledge.

The reader may think of this as identity-based trapdoor commitments with "identities" $M$ and $M^*$, respecively, and compare this to the construction in Section 3 of Chapter 3 (page 35). There, we applied the similar idea using $g_1 g_2^{\mathrm{id}}$ and $h_1$ for the generators instead of $g_1 M$ and $h_1$. Here, $M$ and $M^*$ replace the identities $g_2^{\mathrm{id}}$ of the sender and $g_2^{\mathrm{id}^*}$ of the adversary.

We describe the protocol given in Figure 5 which combines the aforementioned ideas. The public parameters are (a description of) a cyclic group $G_q$ of prime order $q$ and four random generators $g_0, g_1, h_0, h_1$ of $G_q$. Basically, the sender $\mathcal{S}$ commits to his message $m \in \mathbb{Z}_q^*$ with Pedersen's scheme[4] by computing $M = g_0^m h_0^r$ and proves by a proof of knowledge (values $S, c, y, z$ in Figure 5) that he is aware of a valid opening of the commitment. The challenge $c$ in this proof of knowledge is determined by a coin-flipping protocol with values $A, a, u, b$.

FIGURE 5. Discrete-Log-Based Non-Malleable Commitment Scheme

| **Sender $\mathcal{S}$** | $G_q, g_0, g_1, h_0, h_1$ | **Receiver $\mathcal{R}$** |
|---|---|---|
| message $m \in \mathbb{Z}_q^*$ | | |
| *a) commitment phase*: | | |
| choose $a, r, s, t, u \in_{\mathrm{R}} \mathbb{Z}_q$ | | |
| set $M := g_0^m h_0^r$ | | |
| set $A := (g_1 M)^a h_1^u$ | | |
| set $S := g_0^s h_0^t$ | $\xrightarrow{M, A, S}$ | choose $b \in_{\mathrm{R}} \mathbb{Z}_q$ |
| | $\xleftarrow{b}$ | |
| set $c := a + b \bmod q$ | | |
| set $y := s + cm \bmod q$ | | |
| set $z := t + cr \bmod q$ | $\xrightarrow{a, u, y, z}$ | set $c := a + b \bmod q$ |
| | | check $A \stackrel{!}{=} (g_1 M)^a h_1^u$ |
| | | check $SM^c \stackrel{!}{=} g_0^y h_0^z$ |
| *b) decommitment phase*: | | |
| | $\xrightarrow{m, r}$ | check $M \stackrel{!}{=} g_0^m h_0^r$ |

It is clear that our protocol is computationally binding under the discrete-log assumption, and perfectly secret as the additional proof of knowledge for $m$

[4]Note that as opposed to Pedersen's scheme we require that $m \neq 0$; the technical reason is that in the security proof we need to invert the message modulo $q$.

is *witness independent* (aka. perfectly witness indistinguishable) [FS90], i.e., for any challenge $c$ the transmitted values $S, y, z$ are distributed independently of the actual message [O92].

**Proposition 4.4.** *The commitment scheme in Figure 5 is perfectly secret and, under the discrete-log assumption, computationally binding.*

In the next section we stringently prove that our scheme is indeed non-malleable. By now, we already remark that the non-malleability property of our scheme also relies on the hardness of computing discrete logarithms. This dependency is not surprising: after all, any adversary being able to compute discrete logarithms with noticeable probability also refutes the binding property of Pedersen's scheme and can thus decommit for any related message with this probability.

A rough idea why our protocol is non-malleable can be described as follows. Given a commitment $M$ of some unknown message $m$ (together with a witness-independent proof of knowledge described by $S, c, y, z$) with respect to parameters $p, q, g_0, h_0$ we show how to employ the PIM adversary $\mathcal{A}$ to derive some information about $m$. Namely, if we are able to learn the related message $m^*$ of the adversary by extracting it via his "self-employed" proof of knowledge, then we know that $m$ is related to $m^*$ for the relation $\mathsf{R}$. This, of course, contradicts the perfect secrecy of the commitment $M$. We remark that the formal proof of non-malleability requires to come up with a simulator generating a related message without the help of the sender. However, as we will show, the essential part of the simulator is made out of such an extraction procedure. For details and further discussion we refer to the next section.

**Theorem 4.5.** *Under the discrete-logarithm assumption, the scheme in Figure 5 is a perfectly-secret commitment scheme which is liberal non-malleable with respect to opening.*

It is worthwhile to point out that we cannot hash longer messages to $\mathbb{Z}_q^*$ before applying our non-malleable commitment scheme. Because then we extract the hash value and not the message $m^*$ itself. But this could be insufficient, since it might be impossible to deduce anything about $m$ via $\mathsf{R}(\mathrm{AdvPar}, \mathrm{Hist}(m), m, m^*)$ given solely the hash value of $m^*$. The same disadvantage occurs in the RSA case. A solution for this using so-called a-posteriori verifiable proofs of knowledge relying on the Chinese Remainder Theorem appears in [FF00]. There, one can first hash the message as the proof of knowledge operates on the original message instead of the hash value.

### 4.3. Formal Proof of Non-Malleability

We present the proof of non-malleability of the protocol in the previous section first from a bird's eye view and progressively fill in more details. The main part of

the proof consists of the construction of an extraction procedure that enables us to extract the adversary's message related to the original message. We start with an outline of this procedure, then analyze it with respect to restricted attacks and, subsequently, supplement the remaining steps for full-fledged attacks. Finally, we discuss that the required non-malleability simulator can be derived from the extraction procedure. At the end of this section we turn to the multi-party setting.

**Outline of Extraction Procedure.** In this outline here, we make some simplifications concerning the adversary: first, we assume that the PIM adversary always catches up concerning the order of the transmissions, i.e., sends his first message after learning the first message of $\mathcal{S}$ and answers to $\mathcal{S}$ after having seen $\mathcal{R}$'s response etc. Second, let the adversary *always* successfully commit and decommit to a related message, rather than with, say, small probability. Third, we presume that $\mathsf{M}$ is independent of the adversarial parameters. All restrictions will be removed in the following passages.

To learn the adversary's message $m^*$ we use the proof of knowledge in our commitment protocol. Intuitively, a proof of knowledge guarantees that the prover knows the message, i.e., one can extract the message by running experiments with the prover. Specifically, we inject values $p, q, g_0, h_0, M, S, c, y, z$ into a simulated PIM attack with $\mathcal{A}$ and impersonate $\mathcal{S}$ and $\mathcal{R}$. Additionally, we choose $g_1$ at random and set $h_1 := (g_1 M)^w$ for a random $w \in_{\mathrm{R}} \mathbb{Z}_q$. We also compute random $a_0, u_0 \in_{\mathrm{R}} \mathbb{Z}_q$ and insert $g_1, h_1$ and $A := (g_1 M)^{a_0} h_1^{u_0}$ into the experiment with $\mathcal{A}$. We start with the extraction procedure by committing to $m, s, a_0$ via $M, S, A$ on behalf of the sender. Then, by the predetermination about the order of the transmissions, the adversary sends $M^*, S^*, A^*$ (possibly by changing $M, S, A$ and without knowing explicitly the corresponding values $m^*, r^*$ etc.). See Figure 6 on page 61 for a pictorial description.

We play the rest of the commitment phase twice by rewinding it to the step where the receiver chooses $b$ and sends it to the adversary $\mathcal{A}$. To distinguish the values in both repetitions we add the number of the loop as subscript and write $a_1, a_1^*, a_2, a_2^*$ etc.

The first time, the adversary upon receiving $b_1$ passes some $b_1^*$ to the (simulated) sender $\mathcal{S}$, and expects $\mathcal{S}$ to open the commitment for $a$ and supplement the proof of knowledge for $M$ with respect to the challenge $a_1 + b_1^* \bmod q$. By the trapdoor property of Pedersen's commitment scheme [BCC88] we are able to open $A$ with any value for $a_1$ since we know $\log_{(g_1 M)} h_1$. That is, to decommit $A$ with some $a_1$ reveal $a_1$ and $u_1 = u_0 + (a_0 - a_1)/\log_{(g_1 M)} h_1 \bmod q$; it is easy to verify that indeed $A = (g_1 M)^{a_1} h_1^{u_1}$. In particular, we choose $a_1$ such that $a_1 + b_1^* \bmod q$ equals the given value $c$. Hence, $y$ and $z$ are proper values to complement the proof of knowledge for $M$. Finally, the adversary answers with the decommitment $a_1^*, u_1^*$ for $A^*$ and the rest of the proof of knowledge for $M^*$ with respect to challenge $a_1^* + b_1 \bmod q$.

Now we rewind the execution and select another random challenge $b_2$. The adversary then decides upon his value $b_2^*$ (possibly different from his previous choice $b_1^*$) and hands it to $\mathcal{S}$. Again, we open $A$ with $a_2$ such that $c = a_2 + b_2^* \bmod q$. The adversary finishes his commitment with $a_2^*, u_2^*$ as opening for $A^*$ and the missing values for the proof of knowledge.

The fundamental proof-of-knowledge paradigm [FFS88, FS89, BG92] says that we can extract the message $m^*$ if we learn two valid executions between $\mathcal{A}$ and $\mathcal{R}$ with the same commitment $M^*, S^*, A^*$ but different challenges. Hence, if the adversary's decommitments satisfy $a_1^* = a_2^*$ and we have $b_1 \neq b_2$ (which happens with probability $1 - 1/q$), then this yields different challenges $a_1^* + b_1$, $a_2^* + b_2$ in the executions between $\mathcal{A}$ and $\mathcal{R}$ and we get to know the message $m^*$. We are therefore interested in the event that the adversary is able to "cheat" by presenting different openings $a_1^* \neq a_2^*$. In Section 4.3 we prove that the adversary cannot find different openings for commitment $A^*$ too often, else we would derive a contradiction to the intractability of the discrete-log problem. Hence, under the discrete-log assumption this event hardly occurs and we extract $m^*$ with sufficiently high probability.

Note that that in the repetitions we force the coin-flipping protocol between $\mathcal{S}$ and $\mathcal{A}$ to result in the same challenge both times. The latter is necessary because if we were able to answer a different challenge than $c$ then we could extract the unknown message $m$ and would thus know $m$ (which is of course not the case).

**Extraction With Respect to Restricted Attacks.** We address a more formal approach to the extraction procedure, still considering a slightly restricted attack. Namely, as in the outline, we too adopt the convention that the adversary $\mathcal{A}$ does not "mix" the order of messages but rather catches up. We also presume for simplicity that the messages space $\mathsf{M}$ is independent of the adversarial parameters. Call this a *restricted* attack. We afterwards explain how to deal with *full-fledged* attacks.

Before we jump into restricted attacks, we first remark that the history value $\mathrm{Hist}(m)$ can be neglected for the analysis of the extraction procedure for both restricted and full-fledged attacks. We omit mentioning it since we use only black-box simulations to extract the adversary's message, hence, any value $\mathrm{Hist}(m)$ given to $\mathcal{A}'$ is simply forwarded to $\mathcal{A}$ in order to run the black-box simulation. Only the conclusive construction of the non-malleability simulator from the extraction procedure requires a more careful look at the history value.

Our aim is to extract the adversary's message from his commitment within a negligibly close bound to the adversary's success probability $\pi_{\mathsf{open}}(\mathcal{A})$. To this end, we repeat some basic facts about proofs of knowledge and knowledge extractors [FFS88, FS89, BG92]; we discuss them for the example of Okamoto's discrete-log-based proof of knowledge (see [O92] or Section 4.1) for a given $M = g_0^m h_0^r$.

The knowledge extractor interacting with the prover works in two phases. Namely, it first generates a random conversation $S, c, y, z$ by running the prover

to obtain $S$, by selecting $c$ and by letting the prover answer with $y, z$ to $S, c$. If this communication in the initial run is invalid, then the extractor aborts. Else the extractor also stops with probability $1/q$. Otherwise it extracts at all costs. That is, the extractor fixes this communication up to the challenge, and then loops (till success) to seek another accepting conversation with the same communication prefix $S$ and different $c$. This is done by rewinding the execution to the choice of the challenge and reselecting other random challenges. The extractor runs in expected polynomial time and outputs a representation of $M$ with respect to $g_0, h_0$ with probability $\pi - 1/q$. Here, $\pi$ denotes the probability that the prover makes the verifier accept, and $1/q$ is called the error of the protocol.

Assume that we communicate with some party $\mathcal{C}$ which is going to commit to an unknown message $m \in_{\mathrm{R}} \mathsf{M}$. We choose a group $G_q$ and two generators $g_0, h_0$ and send them to $\mathcal{C}$. Party $\mathcal{C}$ selects $r, s, t \in_{\mathrm{R}} \mathbb{Z}_q$ and sends $M := g_0^m h_0^r$, $S := g_0^s h_0^t$. We answer with a random challenge $c \in_{\mathrm{R}} \mathbb{Z}_q$ and $\mathcal{C}$ returns $y := s + cm, z := t + cr \bmod q$. Finally, we check the correctness. Put differently, we perform all the steps of the sender in our protocol except for the coin flipping.

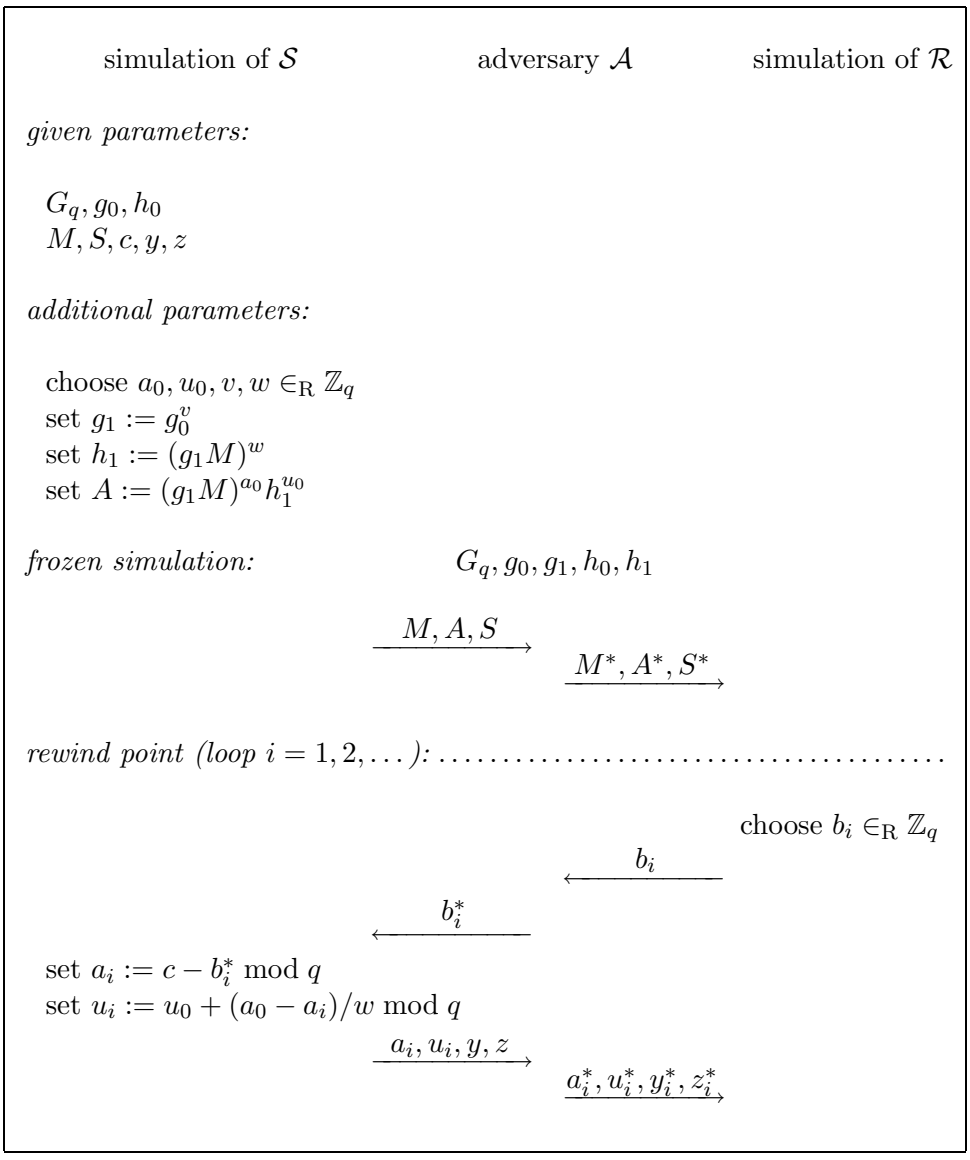
We describe our knowledge extraction procedure. The aim is to get the message $m^*$ of the PIM adversary when the adversary faces $\mathcal{C}$'s commitment. For this, the extractor chooses additional generators $g_1, h_1$ by setting $g_1 := g_0^v$ and $h_1 := (g_1 M)^w$ for random $v, w \in_{\mathrm{R}} \mathbb{Z}_q^*$, and computes $A := (g_1 M)^{a_0} h_1^{u_0}$ according to the protocol description for random $a_0, u_0 \in_{\mathrm{R}} \mathbb{Z}_q$.[5] Then the extractor starts to emulate the PIM attack by pretending to be $\mathcal{S}$ and $\mathcal{R}$ and with values $G_q, g_0, g_1, h_0, h_1, M, S, A$. A description is shown in Figure 6.

Because of the assumption about the order of messages, the adversary commits then to $M^*, S^*, A^*$. Next, we use the same stop-or-extract technique as in [FS89, BG92]. In our case, the rewind point (if we do rewind) is the step where the receiver sends $b$. In each repetition, we send a random value $b_i \in_{\mathrm{R}} \mathbb{Z}_q$ —the subscript denotes the number $i = 1, 2, \ldots$ of the loop— on behalf of the receiver and the adversary hands some value $b_i^*$ to the simulated sender. Knowing the trapdoor $w = \log_{(g_1 M)} h_1$ we open $A$ with $a_i, u_i = u_0 + (a_0 - a_i)/w \bmod q$ such that $a_i + b_i^*$ equals the given value $c$, and send the valid answer $y, z$ to the challenge $c$ in the proof of knowledge for $M$. The adversary replies with $a_i^*, u_i^*, y_i^*, z_i^*$ to the receiver. Again, see Figure 6.

An important modification of the knowledge extractor in comparison to the one in [FS89, BG92] is that, once having entered the loop phase, not only does our extractor stop in case of success; it also aborts with no output if in some repetitions $i, j$ the adversary both times successfully finishes the commitment phase —which includes a correct decommitment of $A^*$— but opens $A^*$ with distinct values $a_i^* \neq a_j^*$. We say that $\mathcal{A}$ *wins* if this happens. In this case, the extractor fails to extract

[5] Clearly, the choice of the generators requires that $M$ and therefore $m$ and $\mathsf{M}$ are determined *before* the adversary is presented PubPar and selects AdvPar.

FIGURE 6. Knowledge Extraction




a message. We remark that we are only interested in the case that $\mathcal{A}$ sends distinct openings of $A^*$ in *accepting* executions, because the extractor only relies on such executions.

Our first observation is that our knowledge extractor stops (either with success or aborting prematurely) in expected polynomial-time. This follows as in [FS89, BG92].

To analyze the success probability of our extractor let $\pi$ denote the probability of $\mathcal{A}$ completing the commitment phase with $\mathcal{R}$ successfully. The basic extraction paradigm says that we are able extract with probability $\pi - 1/q - \epsilon(n)$, where $\epsilon(n)$ denotes the probability that $\mathcal{A}$ wins ($n$ is the security parameter). The reason for this is that, given $\mathcal{A}$ does not win, the adversary's openings $a^*_{i_1} = a^*_{i_2} = \ldots$ in the valid commitment conversations are all equal. But then the values $b_{i_j} + a^*_{i_j} \bmod q$ for $j = 1, 2, \ldots$ of challenges in the proof of knowledge between $\mathcal{A}$ and $\mathcal{R}$ are independently distributed. Analogously to [FS89, BG92] it follows that the extractor finds a message with probability $\pi - 1/q - \epsilon(n)$ in this case.

Recall that we would like to guarantee that we extract with probability approximately $\pi_{\mathsf{open}}(\mathcal{A})$. Apparently, $\pi$ upper bounds $\pi_{\mathsf{open}}(\mathcal{A})$, and it would thus suffice to show that $\epsilon(n)$ roughly equals $\pi - \pi_{\mathsf{open}}(\mathcal{A})$, or put differently, that $\delta(n) = \epsilon(n) - (\pi - \pi_{\mathsf{open}}(\mathcal{A}))$ is negligible. One may think of the difference $\pi - \pi_{\mathsf{open}}(\mathcal{A})$ describing the probability of executions in which $\mathcal{A}$ successfully commits but never finds a related, valid opening (e.g., if $\mathcal{A}$ simply duplicates all messages of $\mathcal{S}$ in the commitment phase).

It remains to bound the probability $\delta(n)$. We will prove that $\delta(n)$ is negligible under the discrete-log assumption.

**Lemma 4.6.** *The probability that $\mathcal{A}$ wins is negligibly close to $\pi - \pi_{\mathsf{open}}(\mathcal{A})$.*

We remark that the proof of this lemma makes use of two important aspects. On one hand, we exploit that the message space is fixed before the adversarial parameters are chosen. On the other hand, we apply the fact that we merely demand non-malleability with respect to opening, i.e., that $\mathcal{A}$ also reveals a valid decommitment.

**Proof.** We show that if the claim of Lemma 4.6 does not hold this contradicts the intractability of the discrete-log problem. We are given a group $G_q$, a generator $g$, and a value $X \in G_q$ for which we are supposed to compute $\log_g X$. We show how to use $\mathcal{A}$ to do so.

Instead of using the commitment $M$ of the third party $\mathcal{C}$, this time we run the knowledge extraction procedure incorporating the given values $G_q, g, X$, but generating the same distribution as the extractor. That is, select a message $m \in_{\mathrm{R}} \mathsf{M}$, as well $v, w \in_{\mathrm{R}} \mathbb{Z}^*_q$, set

$$g_0 := g^{-1/m} X, \qquad g_1 := g, \qquad h_0 := X^v, \qquad h_1 := X^w,$$

and compute $M, A, S, c, y, z$ according to the protocol description. Wlog. assume that $X \neq 1$ and $X^m \neq g$, else we already know the discrete log of $X$. Then $g_0$, $g_1$, $h_0$ and $h_1$ are random generators of the subgroup $G_q$. Furthermore, $g_1 M = g g_0^m h_0^r = X^{m+rv}$ and thus $\log_{(g_1 M)} h_1 = (m + rv)/w \bmod q$.

Next we emulate $\mathcal{A}$ on values $G_q, g_0, g_1, h_0, h_1$ and $M, A, S$ by running the extraction procedure above —with the exception that this time we enter the rewind

phase only if the adversary successfully commits *and* also reveals a valid decommitment $(m^*, r^*)$ to a related message after learning our decommitment $(m, r)$ in the initial execution.

Once we have entered the rewind phase, whenever the extractor is supposed to open $A$ to determine the challenge $c$ in the loop, we also open the commitment such that the coin flipping protocol always yields the same value $c$. This is possible as we know $\log_{(g_1 M)} h_1$ and are therefore able to open $A$ ambiguously.

Unlike in the case of an actual extraction process, here we sometimes suspend before looping although the adversary's initial commitment is accepted (because we also stop if the adversary's decommitment in the initial execution is invalid or unrelated). This restriction decreases the probability of $\mathcal{A}$ winning at most by $\pi - \pi_{\mathsf{open}}(\mathcal{A})$. We call runs in which $\mathcal{A}$ also opens correctly in the initial execution *good*.

Observe that the communication in good experiments here is identically distributed to the one in the extraction procedure. Hence, given that $\mathcal{A}$ wins with probability $\epsilon(n) = \pi - \pi_{\mathsf{open}}(\mathcal{A}) + \delta(n)$ in the actual extraction procedure, $\mathcal{A}$ finds some $a_i^* \neq a_j^*$ for two accepting executions $i, j$ with probability at least $\delta(n)$ in a good run here. By assumption, $\delta(n)$ is noticeable, so it suffices to prove that if $\mathcal{A}$ wins in a good extraction then we can compute the discrete logarithm of $X$.

Let $u_i^*, u_j^*$ denote the corresponding portions of the decommitment to $a_i^*$ and $a_j^*$ for $A^*$ in loops $i$ and $j$. In a good run we have obtained some $m^*, r^*$ satisfying the verification equation $M^* = g_0^{m^*} h_0^{r^*}$ from the adversary by revealing $m, r$ in place of the sender in the initial execution. Particularly, we have:

$$(g_1 M^*)^{a_i^*} h_1^{u_i^*} = A^* = (g_1 M^*)^{a_j^*} h_1^{u_j^*}$$

and therefore

$$h_1^{(u_i^* - u_j^*)/(a_j^* - a_i^*)} = g_1 M^* = g_1 g_0^{m^*} h_0^{r^*} = g^{1 - m^*/m} X^{m^* + r^* v}$$

Since $h_1 = X^w$ we can transform this into

$$g^{1-m^*/m} = X^{\Delta} \qquad \text{for } \Delta = w(u_i^* - u_j^*)/(a_j^* - a_i^*) - (m^* + r^* v) \bmod q$$

Observe that $\Delta$ is computable from the data that we have gathered so far. From $m^* \neq m$ we conclude that $1 - m^*/m \neq 0 \bmod q$ and therefore $\Delta \neq 0 \bmod q$ has an inverse modulo $q$. Thus the discrete logarithm of $X$ to base $g$ equals $(1 - m^*/m)/\Delta \bmod q$. □

In summery, with probability $\pi_{\mathsf{open}}(\mathcal{A}) - 1/q - \delta(n)$ (which is negligibly close to the adversary's success probability) we extract some message $m'$. The final step is to show that indeed $m'$ equals the adversary's decommitment $m^*$ except with negligible probability (or, more precisely, that $m'$ is at least an appropriate substitution for $m^*$ insofar as it also satisfies $\mathsf{R}$ often enough). Denote by $\pi_{\mathsf{open}}(\mathcal{E})$

the probability that the extraction procedure returns $m'$ that is related to $m$ under $\mathsf{R}$.

**Lemma 4.7.** *The probabilities $\pi_{\mathsf{open}}(\mathcal{A}) - 1/q - \delta(n)$ and $\pi_{\mathsf{open}}(\mathcal{E})$ are negligibly close.*

Again, this lemma relies on the fact that the message space is independent of the adversarial parameters.

**Proof.** Similar to Lemma 4.6 if this were not the case we could compute the discrete logarithm of $X$ to $g$ in group $G_q$. Namely, define $g_0 := g$ and $h_0 := X$ and run the extraction procedure as before, only this time compute $M, S, c, y, z$ for yourself, in particular, sample $m \in_{\mathrm{R}} \mathsf{M}, r \in_{\mathrm{R}} \mathbb{Z}_q$ and set $M := g_0^m h_0^r$, and choose $g_1$ at random and set $h_1 := (g_1 M)^w$ for a random $w \in_{\mathrm{R}} \mathbb{Z}_q^*$.

In the initial run of the extraction procedure, if the adversary has finished the commitment phase successfully, hand the decommitment of $M$ to the adversary and try to elicit the opening $m^*, r^*$ of $M^*$. If the adversary refuses to decommit to $M^*$ correctly, then stop; else continue the extraction. According to Lemma 4.6 the extraction yields a representation $m', r'$ of $M^*$ with probability $\pi_{\mathsf{open}}(\mathcal{A}) - 1/q - \delta(n)$. We are interested in the probability that $m'$ also satisfies the relation.

Suppose that $\pi_{\mathsf{open}}(\mathcal{A}) - 1/q - \delta(n)$ and $\pi_{\mathsf{open}}(\mathcal{E})$ have noticeable difference. In particular, we conclude that $m' \neq m^*$ with noticeable probability. But this implies that sufficiently often we obtain distinct representations $(m^*, r^*)$, $(m', r')$ of $M^*$. We are thus able to compute the discrete logarithm of $h_0 = X$ to base $g_0 = g$ with noticeable probability. Hence, under the discrete logarithm assumption, the probability that the extraction procedure returns $m'$ that stands in relation to the sender's message is negligibly close to $\pi_{\mathsf{open}}(\mathcal{A}) - 1/q - \delta(n)$. □

**Thwarting Full-Fledged Attacks.** Our first observation is that the order of the messages in the PIM attack does not violate any of the discussions above. This is quite easy to see since any message on the sender's side can be predetermined at the outset of the knowledge extraction procedure.

So the final step is to remove the assumption about the message space. We have used three times the fact that $\mathsf{M}$ can be determined before the adversarial parameters are presented to the adversary. First, we have set $h_1$ equal to $g_1 M$, i.e., generated $h_1$ after seeing the commitment of $m \in_{\mathrm{R}} \mathsf{M}$ in the extraction procedure. Second, in the proof of Lemma 4.6, we have sampled $m \in_{\mathrm{R}} \mathsf{M}$ and then incorporated it into the generators. Third, Lemma 4.7 also requires to choose $M$ before the adversary generates AdvPar. We solve the former point first, and then show how to deal with the latter problems. In any case, this boils down to select the public parameters PubPar before sampling $m$, because AdvPar is a random variable depending on PubPar only. Note that we do not change our protocol, but only the extraction and simulation procedures.

In the knowledge extraction procedure, recall that we copy the commitment $M, S, c, y, z$ of party $\mathcal{C}$ into the extraction procedure and then set $h_1 := (g_1 M)^w$ for random $w$. To remove the dependency of a preselected message space, we modify $M, S$ before using it in the proof of knowledge. That is, one first selects a group $G_q$ and $M_{\mathsf{fake}} \in_{\mathrm{R}} G_q$. Then we present $G_q, g_0, h_0, g_1, h_1$ to the adversary, where $g_0, h_0, g_1$ are random generators and $h_1 := (g_1 M_{\mathsf{fake}})^w$. This also determines $\mathsf{M} = \mathsf{M}(\mathrm{AdvPar})$ and we invoke $\mathcal{C}$ on $G_q, g_0, h_0$ and $\mathsf{M}$ to obtain $M, S, c, y, z$. Instead of using $M, S$ in the extraction procedure, we run the knowledge extractor with $M_{\mathsf{fake}}$ and $S_{\mathsf{fake}} := S(M M_{\mathsf{fake}}^{-1})^c$ as well as $c, y, z$. Clearly, these values satisfy the verification equation $S_{\mathsf{fake}} M_{\mathsf{fake}}^c = g_0^y h_0^z$. Moreover, they are identically distributed to honestly generated ones, and hence the extractor achieves the same success probability. It is instructive to think of $M_{\mathsf{fake}}$ and $S_{\mathsf{fake}}$ as rerandomized versions of $M, S$.

The solution for the problem in Lemma 4.6 is similar to the previous case. There, we have chosen a group $G_q$ and $g_0 := g^{-1/m} X$, $g_1 := g$, $h_0 := X^v$ and $h_1 := X^w$. By this, we have possessed the discrete logarithm of $h_1 = X^w$ to base $g_1 M = g_1 g_0^m h_0^r = X^{(m+rv)}$. Instead, we now select $G_q$, choose a dummy message $m_0 \in_{\mathrm{R}} \mathbb{Z}_q^*$ and set $g_0 := g^{-1/m_0} X$, $g_1 := g$, $h_0 := (g^{-1/m_0} X)^v$ and $h_1 := X^w$ and $M := g_0^{m_0}$. The values $G_q, g_0, g_1, h_0, h_1$ fix $\mathsf{M} = \mathsf{M}(\mathrm{AdvPar})$ and enable us to choose now the genuine message $m \in_{\mathrm{R}} \mathsf{M}$. Since we know $v = \log_{g_0} h_0$ we can find an appropriate $r$ with $m + vr = m_0$. Thus, $g_1 M = g_1 g_0^m h_0^r = X^{m+rv}$ and, again, $\log_{(g_1 M)} h_1 = (m + rv)/w$. Except for the case that $m + rv = 0$ in Lemma 4.6, which happens with probability $1/q$, this way of selecting the public parameters is identical to the generation there.

We discuss that the proof carries over to the modification for Lemma 4.6. In the proof of Lemma 4.6 we finally find $\Delta$ with $g^{1-m^*/m} = X^\Delta$ and are able to compute the discrete logarithm of $X$ to $g$ since $m^* \neq m$. Here, we obtain the equation $g^{1-(m^*+vr^*)/(m+vr)} = X^\Delta$. If we would have $m^* + vr^* = m + vr$ with noticeable probability, then from $m^* \neq m$ it would follow that the adversary finds a different representation $m^*, r^*$ of $M = g_0^m h_0^r$ to base $g_0, h_0$ with noticeable probability. Specifically, given $G_q, g_0 := g, h_0 := X$ select random $g_1, h_1$ and then sample a message $m \in_{\mathrm{R}} \mathsf{M}(\mathrm{PubPar})$. Compute the commitment $M = g_0^m h_0^r$ for random $r$ as well as the values $S, A$ for the proof of knowledge. Run only the initial commitment and decommitment phase of Lemma 4.6. If the adverary sends $b^*$ for the coin-flipping subprotocol in this initial run, then open the commitment for $A$ with the previously selected values $a, u$ and evaluate $y, z$ for the proof of knowledge for $S, c = a \oplus b^*$. Finally, reveal $m, r$ to the adversary to obtain $m^*, r^*$.

Note that we do not need to know the discrete logarithm of $h_1$ to $g_1 M$ here, since we do not loop, but merely run the initial phase. By assumption, $m^* + r^* \log_{g_0} h_0 = m + r \log_{g_0} h_0$ with noticeable probability. This, in turn, yields the discrete logarithm of $h_0 = X$ to $g_0 = g$. Hence, under the discrete logarithm

assumption this happens with negligible probability only, and by analogy with Lemma 4.6 we therefore derive that the probability of $\mathcal{A}$ winning does not exceed $\pi - \pi_{\mathsf{open}}(\mathcal{A})$ noticeably.

Finally, to adapt Lemma 4.7, we need to show that extracting $m'$ different than $m^*$ is infeasible, even if we have to publish the public parameters ahead of the choice of $\mathsf{M}$. Remember that in Lemma 4.7 we have used the adversary to find distinct representations $(m^*, r^*), (m', r')$ of $M^*$ and to compute the discrete logarithm of $h_0 = X$ to $g_0 = g$ in $G_q$. Here, given $G_q, g, X$ we make the following selection for random $r, v, w \in \mathbb{Z}_q$:

$$g_0 := g, \qquad g_1 := g_0^v h_0^{-r}, \qquad h_0 := X, \qquad h_1 := g_0^w,$$

These parameters pin down $\mathsf{M} = \mathsf{M}(\text{AdvPar})$. We sample $m \in_{\mathrm{R}} \mathsf{M}$ and let $M := g_0^m h_0^r$ for the preselected value $r$; the values of the proof of knowledge are computed honestly. It is easy to see that all values have the correct distribution (unless $g_1 = 1$ or $h_1 = 1$, in which case we simply abort). Furthermore, we know the discrete logarithm $w/(v+m)$ of $h_1$ with respect to $g_1 M$.

This completes the analysis of the extraction procedure with respect to full-fledged attacks.

**Extraction Implies Non-Malleability.** A general construction of a non-malleability simulator $\mathcal{A}'$ from an extraction procedure has already appeared in [DDN00] (for the plain model, but it is straightforward to adapt it to the public parameter model, as done below). We briefly review the construction of $\mathcal{A}'$ for our case.

The non-malleability simulator $\mathcal{A}'$ prepares the public parameters as required for the extraction procedure, invokes the adversary $\mathcal{A}$ to obtain AdvPar and sets $\text{AdvPar}' := \text{AdvPar}$. Then the honest sender $\mathcal{S}$ is given a secret message $m \in_{\mathrm{R}} \mathsf{M}(\text{AdvPar}')$ and $\mathcal{A}'$ receives $\text{Hist}(m)$ (which is forwarded to $\mathcal{A}$ for the black-box simulation).

For the extraction procedure, $\mathcal{A}'$ also has to prepare a commitment $M$ of $m$ together with a proof of knowledge $S, c, y, z$, but without actually knowing the secret message $m$ of the sender. We let $\mathcal{A}'$ simply take an arbitrary message $m_0 \in \mathsf{M}(\text{AdvPar}')$ and compute $M, S, c, y, z$ from this message $m_0$ instead. Since the commitment $M$ is perfectly secret and $S, c, y, z$ are distributed independently of $m_0$, these values are equivalent to genuine values. This holds even if $m_0$ does not match the a-priori information $\text{Hist}(m)$ the adversary has about the sender's message.[6]

Finally, the simulator $\mathcal{A}'$ outputs the message it extracts from the PIM adversary. The results about the extraction procedure in the previous sections show

[6] In fact, a slightly more sophisticated argument shows that this would also be true if the commitment scheme was only computationally secret [DDN00].

that the success probability of $\mathcal{A}'$ is at most negligibly smaller than the probability of the PIM adversary. This completes the proof.

**The Multi-Party Case.** It is not hard to see that non-malleability in the multiple-sender scenario follows from the single-sender case for our protocols. Nevertheless, if we grant the adversary the possibility to commit in several executions then we are not aware if our proof technique still works. To bypass this dilemma we use the proposal from [DDN00] that the adversary announces some subset of indices $i_1, \ldots, i_k$ in the commitment phase. The adversary is then called successful if he finds valid openings for these commitments and if $m^*_{i_1}, \ldots, m^*_{i_k}$ stand in relation to $m$. That is, we can view $\mathsf{R}$ as a restricted relation $\mathsf{R}(\mathrm{AdvPar}, \mathrm{Hist}(m), m, m^*_{i_1}, \ldots, m^*_{i_k})$. It follows straightforwardly that, if we let the adversary in our case announce the subset after having sent all the commitments $M^*_1, \ldots, M^*_{\mathrm{poly}}$, then our scheme becomes liberal non-malleable with respect to opening in the multiple-sender/multiple-adversary setting.

## 5. Non-Malleable Commitments Based on RSA

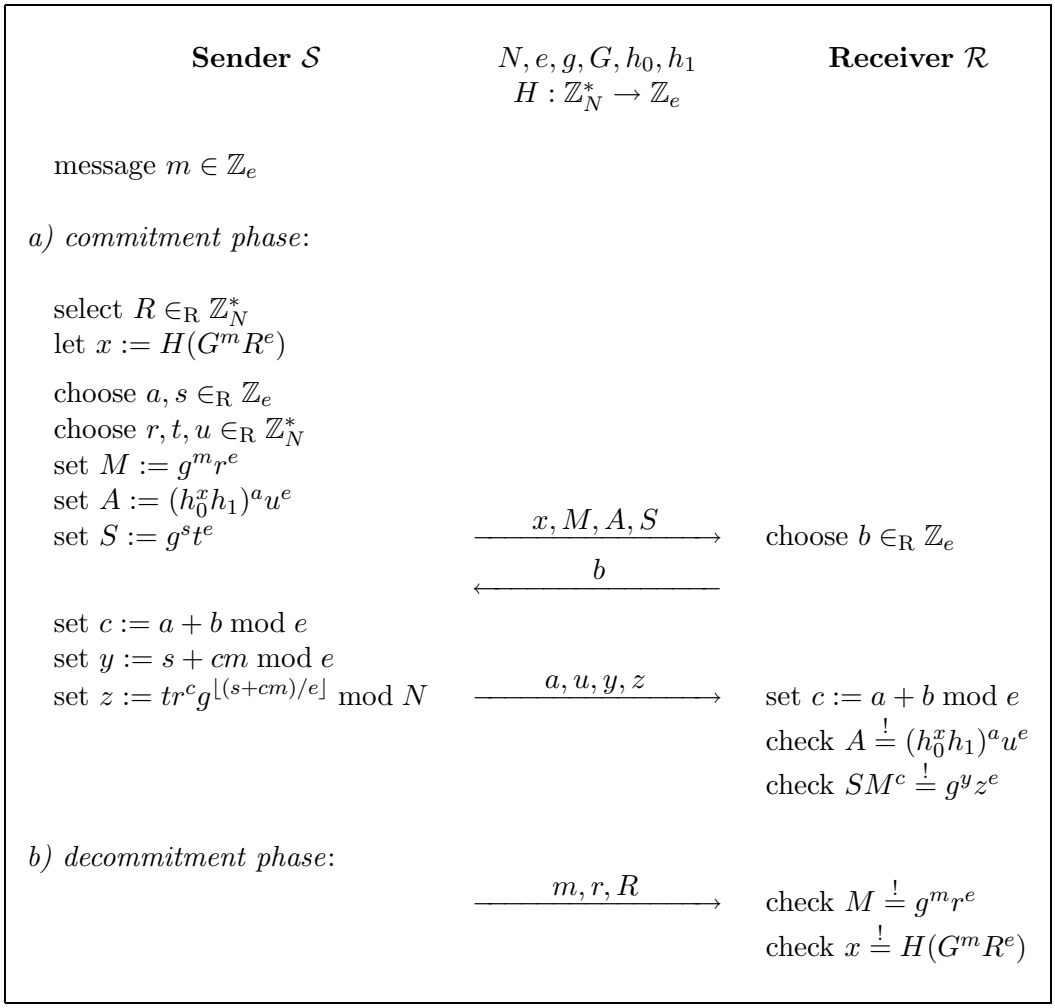
In this section, we present the protocols based on RSA. The basic ideas remain: add a proof of knowledge to a commitment of the message, where the challenge is determined by a coin-flip subprotocol which involves the commitment of the message. Some slight adjustments have to been done, though.

Let $N$ be an RSA modulus, i.e., the product of two large primes. An RSA exponent for $N$ is an integer $e$ which is co-prime to the Euler totient function $\varphi(N)$ and satisfies $e \not\equiv 1 \pmod{\varphi(N)}$. The RSA assumption says that computing $g^{1/e} \bmod N$ for a random $g \in_{\mathrm{R}} \mathbb{Z}^*_N$ is intractable.

The RSA-based non-malleable commitment scheme is built on the function $(m, r) \mapsto g^m r^e \bmod N$ for $m \in \mathbb{Z}_e$, $r \in \mathbb{Z}^*_N$ and $e$ prime [O92]. A commitment of $m \in \mathbb{Z}_e$ is given by $M := g^m r^e \bmod N$ for a random $r \in_{\mathrm{R}} \mathbb{Z}^*_N$. This commitment scheme is perfectly secret (as taking $e$-th powers is a permutation on $\mathbb{Z}^*_N$) and computationally binding, and it supports an efficient three-round witness-independent proof of knowledge similar to the discrete-log case. Furthermore, it also gives rise to a trapdoor property. If (and only if) one knows the trapdoor $g^{1/e} \bmod N$, then one can open the commitment with arbitrary messages. Finally, we notice that one can efficiently compute an $e$-th root of $h$ from $k, h, \Delta, N, e$ satisfying the equation $h^k = \Delta^e \bmod N$ for $k \neq 0 \bmod e$.

For our protocol we also require a family of universal one-way hash functions [NY89]. This is a sequence $H = (H_n)_{n \in \mathbb{N}}$ of function sets $H_n := \{H_{k,n} \mid k\}$, where each $H_{k,n}$ maps elements from the common domain $D_n$ to a common range $R_n$. Additionally, the family is target-resistent, i.e., for any probabilistic polynomial-time algorithm $\mathcal{A}$ the probability that $\mathcal{A}(1^n)$ generates some $x \in D_n$ and, after some function $H_{k,n}$ has been chosen uniformly from $H_n$ and has been presented

FIGURE 7. RSA-Based Non-Malleable Commitment Scheme




to $\mathcal{A}$, then $\mathcal{A}$ returns $x' \neq x$ with $H_{k,n}(x) = H_{k,n}(x')$, is negligible. In particular, every collision-intractable hash function is also universal one-way. In the following, we usually refer to an instance $H_{k,n}$ simply as $H$.

We describe our non-malleable commitment in Figure 7. The public parameters consist of a random RSA instance $N, e$ and four random elements $g, G, h_0, h_1 \in_R \mathbb{Z}_N^*$ together with a universal one-way hash function $H : \mathbb{Z}_N^* \to \mathbb{Z}_e$. To commit to $m \in \mathbb{Z}_e$, choose $r \in_R \mathbb{Z}_N^*$ and set $M := g^m r^e$. Furthermore, compute $x := H(G^m R^e)$ for random $R \in_R \mathbb{Z}_N^*$ and select $a \in_R \mathbb{Z}_e$, $r, u \in_R \mathbb{Z}_N^*$ to calculate $A := (h_0^x h_1)^a u^e$ for the coin-flipping protocol. We remark that, in constrast to the discrete-log case where $A = (g_1 M)^a h_1^u$, here a commitment of the message

enters $A$ vicariously by means of $h_0^x$ for the hash value $x$ of yet another commitment $G^m R^e$ of the message. In addition to the computations above, execute the proof of knowledge protocol given in [O92] for $M$. Clearly, the derived scheme is computationally binding and perfectly secret.

In comparison to the discrete-log case, we have to perform some extra work. Namely, we give two commitments of $m$ and we use a universal one-way hash function $H$. The reason for this basically stems from the lack of symmetry: in the discrete-log case we use two generators and two exponents, whereas here the party selects one exponent and a single value raised to the $e$-th power. Indeed, the second commitment $G^m R^e$ is only necessary if the message space depends on the adversarial parameters. Otherwise one could hash $M$ to $x$ and still achieve non-malleability with respect to such an "independent" message space.

It remains to prove non-malleability. The proof is very similar to the one of the discrete-log case, so we only sketch the necessary adaptions of the main steps. We again begin with the extraction procedure with respect to restricted attacks where the message space is independent of the adversarial parameters and then lift it to full-fledged attacks. Once more, the order of the messages in the executions between the sender and the adversary, and the adversary and the receiver is irrelevant to the discussion. Also, the construction of the non-malleability simulator from the extraction procedure is quasi identical to the discrete-log case and we do not address this part of the proof here.

**Restricted Attacks.** We first describe the extraction procedure in the RSA case. Given $N, e, g$ and a commitment $M = g^m r^e$ for an unknown messages $m \in \mathsf{M}$ together with a proof of knowledge, select $v \in_{\mathrm{R}} \mathbb{Z}_N^*$ and set $G := v^e \bmod N$. Also, let $x := H(R^e)$ for random $R \in \mathbb{Z}_N^*$ and define $h_0 \in_{\mathrm{R}} \mathbb{Z}_N^*$ as well as $h_1 := h_0^{-x} w^e$ for $w \in_{\mathrm{R}} \mathbb{Z}_N^*$. With these choices the $e$-th root of $h_0^x h_1$ equals $w$, hence the coin-flip commitment $A := (h_0^x h_1)^a u^e$ is openable with any value $a$, and the extraction process is therefore identical to the discrete-log case.

The extraction works as long as the adversary does not find ambiguous decommitments for the commitment $A^*$. In Lemma 4.6 it is shown that this probability is negligible close to $\pi - \pi_{\mathsf{open}}(\mathcal{A})$ under the discrete-log assumption. Basically, the technique is to choose appropriate parameters to be able to mimic the extraction procedure and to use the ambiguous opening to $A^*$ to compute the discrete logarithm of $X$ with respect to $g$ in group $G_q$.

In an intermediate step, we first show that we can essentially restrict ourselves to the case that the adversary sends a different hash value $x^* \neq x$. If the adversary would be able to find a related opening with noticeable probability for $x^* = x$, this would contradict either the one-wayness of $H$ or the RSA assumption. Namely, given $N, e$ and a random $X \in \mathbb{Z}_N^*$ let $G := X$ and compute the other public parameters correctly, and sample $m \in_{\mathrm{R}} \mathsf{M}$ and compute $M := g^m r^e$ and $G^m R^e$. Then, given the universal one-way hash function $H(\cdot)$, compute $x := H(G^m R^e)$

and run the adversary on these parameters. If the adversary chooses $x^* = x$ and later reveals a correct decommitment $m^*, r^*, R^*$ after learning $m, r, R$, we either have $G^m R^e = G^{m^*}(R^*)^e$ from which we can compute the $e$-th root of $G = X$, or we have $G^m R^e \neq G^{m^*}(R^*)^e$ yielding a collision $H(G^m R^e) = x = x^* = H(G^{m^*}(R^*)^e)$ for $H(\cdot)$. Hence, the adversary succeeds for $x = x^*$ only with negligible probability. Observe that this argument even holds if the message space depends on the adversarial parameters.

From now on we condition on the event that the adversary always selects $x^* \neq x$. Transferring Lemma 4.6 means that we are given $N, e, X$ and try to compute the $e$-th root of the random value $X \in \mathbb{Z}_N^*$. For this, we copy $N, e$, sample $m \in_{\mathrm{R}} \mathsf{M}$, compute $M := g^m r^e$ and $x := H(G^m R^e)$ for $r, g, G, R \in_{\mathrm{R}} \mathbb{Z}_N^*$ and, again, set $h_1 := h_0^{-x} w^e$ for random $w \in_{\mathrm{R}} \mathbb{Z}_N^*$ and $h_0 := X$. Analogously to Lemma 4.6 we run the extraction procedure (with the opening step in the initial execution to obtain $m^*, r^*, R^*$). Under this assumption, and following the approach in Lemma 4.6, from an ambiguous decommitment for $a^*$ for the values chosen above, we finally derive the equation

$$h_0^{(x^*-x)(a_i^*-a_j^*)} = \Delta^e \bmod N$$

for known $\Delta, x^* \neq x, a_i^* \neq a_j^*$. Since $(x^* - x)$, $(a_i^* - a_j^*) \neq 0 \bmod e$ we can compute an $e$-th root of $h_0 = X$. Hence, under the RSA assumption the extraction procedure succeeds with probability $\pi_{\mathsf{open}}(\mathcal{A}) - 1/e - \delta(n)$, where $\delta(n)$ is negligible.

The final step in the proof of the discrete-log case is Lemma 4.7, where we show that the extracted messages $m'$ is (at least) a suitable replacement of $m^*$. In that lemma, we prove that if this were not true, then we could compute discrete logarithms. The analogous proof here is the same as in the the adaption of Lemma 4.6: given $N, e, X$, choose $m \in_{\mathrm{R}} \mathsf{M}$, set $g := X$ and compute $M := g^m r^e$ as well as $x := H(G^m R^e)$ for random $G, R \in_{\mathrm{R}} \mathbb{Z}_N^*$. Moreover, let $h_0 \in_{\mathrm{R}} \mathbb{Z}_N^*$ and $h_1 := h_0^{-x} w^e$. Run the extraction procedure (with an initial decommitment step to get $m^*, r^*, R^*$) to obtain $m', r', R'$ with $M^* = g^{m^*}(r^*)^e = g^{m'}(r')^e$; since $m^* \neq m'$ this yields the $e$-th root of $g = X$.

**Full-Fledged Attacks.** Here, the messages space is not independent of the adversarial data anymore. Similar to the discrete-log case we have to ensure that we are able to produce appropriate public parameters before we get to know the message space.

For the extraction procedure we choose the same rerandomizing technique as in the discrete-log case. To adapt the modification of Lemma 4.6 we select $G := v^e \bmod N$ for a random $v \in_{\mathrm{R}} \mathbb{Z}_N^*$ and precompute $x := H(R_0^e)$ for $R_0 \in_{\mathrm{R}} \mathbb{Z}_N^*$; since we know the $e$-th root of $G$ is easy to find an appropriate value $R$ matching $x$ for the afterwards chosen message $m \in_{\mathrm{R}} \mathsf{M}(\mathrm{AdvPar})$. Choose the parameters $g, h_0 \in_{\mathrm{R}} \mathbb{Z}_N^*$ honestly, and set $h_1 := h_0^{-x} w^e \bmod N$ for random $w \in_{\mathrm{R}} \mathbb{Z}_N^*$. Conditioning again on the adversary sending $x^* \neq x$ the proof goes through in this case as well.

Finally, we have to prove that the extracted message equals the adversary's one (or, more precisely, satisfies the relation). Similar to the previous step, we select $G$ as $G := v^e \bmod N$ such that we are able to preselect the value $x$. The rest of the proof is as before, i.e., we finally derive different openings of $M$ yielding the $e$-th root of $g$.

**Theorem 4.8.** *If the RSA assumption holds and if $H$ is a family of universal one-way hash functions, then the protocol in Figure 7 is a perfectly secret commitment scheme which is liberal non-malleable with respect to opening.*

Although without practical significance, one can in principle construct collision-intractable hash functions and thus universal one-way hash functions under the RSA assumption. We may therefore reduce the prerequisite of the theorem to the RSA assumption solely.

## 6. Non-Malleable Commitments via Random Oracles

The random oracle methodology [FS86, BR93] exploits the very strong assumption that a hash function behaves like a truly random function. In this model, where all parties have oracle access to such a random function $H$, we devise non-interactive non-malleable commitments in the plain model. However, we remark that the random oracle heuristic provides only some evidence that the scheme is indeed secure if one uses appropriate instantiations for $H$. It might well be that there is no secure implementation in practice at all [CGH98].

The definition of non-malleability transfers to the random oracle model if we augment each party $\mathcal{S}, \mathcal{R}, \mathcal{A}$ and $\mathcal{A}'$ as well as $\mathsf{M}$ and $\mathsf{R}$ with the same oracle $H$ representing a random function with infinite domain and fixed output length. The probability that $\mathcal{A}$ and $\mathcal{A}'$, respectively, succeed is then taken over the random choice of $H$, too.

Let $\mathrm{Com}_{\mathrm{secret}}$ be the non-interactive statistically-secret commitment scheme described in [NY89, DPP93, HM96]. The protocol goes like this: first, the sender hashes the message $m$ to a short string $M$ with some collision-intractable hash function. Then the sender picks a pairwise independent function $h$ and a value $x$ such that $h(x) = M$. It computes the hash value $y$ of $x$ under the collision-intractable hash function and sends $(y, h)$ to $\mathcal{R}$. We omit further details.

Since the protocol $\mathrm{Com}_{\mathrm{secret}}$ merely requires a collision-intractable hash function and random oracles have this property by construction, we may use $H$ as the collision-intractable hash function in the scheme. Then $\mathrm{Com}^{H}_{\mathrm{secret}}$ is indeed non-intractive and still provides statistical secrecy as well as computational unambiguity. We claim that this scheme is even non-malleable with respect to opening in the random oracle model.

Basically, the protocol is non-malleable because any adversary $\mathcal{A}$ sending a commitment $(y^*, h^*)$ and later a correct decommitment $(m^*, x^*)$, each time after

having seen the sender's values $(y, h)$ and $(m, x)$, must have obtained the answers $M^* = H(m^*)$ and $y^* = H(x^*)$ from the oracle queries to $H$. Otherwise the probability that $\mathcal{A}$ finds a good decommitment is negligible, because predicting $H$ on a new value is infeasible. But then $\mathcal{A}$ already "knows" a related message $m^*$ to $m$ in the commitment phase, contradicting the secrecy.

It is now easy to formalize the intuition and define the simulator. $\mathcal{A}'$ first sends a dummy commitment $(y, h)$ on behalf of the sender to $\mathcal{A}$, say, by committing to the all-zero string. Then it records all queries of $\mathcal{A}$ to oracle $H$ and the answers —this is possible as $\mathcal{A}'$ simulates $\mathcal{A}$ and sees all queries of $\mathcal{A}$ before forwarding it to $H$. Since all hash values of $H$ are distinct with overwhelming probability we may assume that every image has a unique preimage in the list of recorded pairs. If finally $\mathcal{A}$ sends some commitment $(y^*, h^*)$ then the simulator looks up $y^*$ in the list and obtains the corresponding query $x^*$ yielding $y^*$. This gives the unique $M^* = h^*(x^*)$ and another search reveals the preimage $m^*$ of $M^*$ under $H$. Let $\mathcal{A}$' output $m' := m^*$. Clearly, the probability that $\mathcal{A}'$ returns a related message is negligibly close to $\mathcal{A}$'s success probability.

*Chapter 5*

# Identification Protocols Secure Against Reset Attacks

This chapter describes how to secure identification protocols by means of trapdoor commitments against so-called reset attacks. This result has been stimulated by a preprint of Bellare, Goldwasser and Micali [BGM00] about secure resettable identification and how to achieve it with the help of secure signature and encryption schemes. Both approaches have been merged for Eurocrypt 2001 in [BFGM01]. For space reasons we merely present a rewritten version of the part related to trapdoor commitments. I thank Mihir Bellare, Shafi Goldwasser and Silvio Micali for their cooperation, and Ran Canetti for discussions about resettable protocols.

Once more, if the reader is familiar with basic notions then this chapter should be apprehensible without the other chapters. Yet, for more background about trapdoor commitment protocols we refer to Chapter 3.

## 1. Introduction

A naive attack on an identification scheme is to try to intrude as another user and pass the examination of the verifier with knowledge of the user's public key only. A more sophisticated attack is to attempt to elicit information about the secret key from the key-owner prior to the identification attempt. That is, pretending to be the verifier and possibly deviating from the verifier's protocol, the adversary first sequentially performs several protocol executions with the prover. In each execution the prover's incarnation is assigned a new random tape, yet the secret key is the same in all runs. Based on this gathered knowledge the adversary then tries to successfully identify himself as the prover to an honest verifier. This is called an active attack.

Unfortunately, active attacks may be insufficient to capture real threats. Assume that the prover's procedure is somehow vulnerable to errors, say, because of

an instable system due to attacks on the prover's environment. Then the adversary can use even more powerful attacks. For instance, the adversary may now be able to reset the prover to some previous message of the same execution and repeat the run with the same prefix but different outcome. More formally, in the resettable setting the adversary has parallel access to many copies of the prover, yet, depending on the adversary's choice, the prover can be resetted and be forced to use the same random tape in a rewound execution. The adversary tries to successfully identify to the verifier after finishing these experiments (CR1 *attack*) or even during this phase (CR2 *attack*).

The most popular identification protocols today follow the proof-of-knowledge paradigm of Fiat-Shamir [FS86]. Although Canetti et al. [CGGM00] point out that proofs of knowledge must leak the witness in the resettable setting and therefore cannot be "secure", for proof-of-knowledge-based identification schemes this argument does not apply offhandedly: although resembling to this pattern, such identification schemes are not per se proofs of knowledge. This phenomenon also occurs when proving security against active (but not resetting) adversaries. Let us discuss this seemingly contradiction in more detail.

To demonstrate security of an identification scheme against active, non-resetting adversaries, one usually invokes a simulator that chooses a random secret key, computes the public key and then, by impersonating the honest parties, emulates an attack of an allegedly successful adversary. This simulation indeed includes rewinding techniques in order to extract some matching secret key from the adversary. Typically, the extracted secret key is different from the chosen one, and this property is used to refute an underlying cryptographic assumption like the intractability of computing discrete logarithms or factoring numbers.

In comparison to proofs of knowledge —if the prover passes the test then a witness can be extracted for *any fixed* input [FFS88, BG92]— security proofs for identification schemes merely signify that passing the check without knowing the secret key is infeasible for a *random* public key *under some cryptographic assumption*. Still, the proof-of-knowledge concept reflects the same paradigm in the context of identification: for proofs of knowledge, extracting a witness (alias secret key) for a given input (alias public key) implies that the prover either knows the secret, or that the prover is able to compute it from the public data. The latter, however, is only believed to be intractable and to provide security against infiltrators if the key is randomly generated according to some cryptographically strong distribution.

Nowadays, there are several well-known identification protocols which are secure against active adversaries but which are provably insecure in the resetting model. Among these schemes are the protocols of Feige-Fiat-Shamir [FFS88] (based on factoring), Ong-Schnorr [OS90, S96] (factoring), Okamoto-Guillou-Quisquater [GQ88, O92] (based on RSA), Okamoto-Schnorr [S91, O92] (discrete

log), Brickell-McCurley [BM92] (discrete log/factoring) and Shoup [S99] (factoring). These schemes are divided into two groups. One group includes the schemes that not only withstand active attacks, but also infiltrations where the adversary is allowed concurrent instead of serial executions with the prover while or before attempting to impersonate —under an additional technical restriction on the adversary in case of a simultaneous intrusion; the details are beyond the scope of this introductory discussion.[1] The protocols of Feige-Fiat-Shamir, Okamoto-Guillou-Quisquater, Okamoto-Schnorr, Brickell-McCurley and, for some system parameters (cf. [S96]), Ong-Schnorr belong to this group. The other group consists of schemes that do not seem to enjoy this additional property (Shoup and, for some parameters, Ong-Schnorr). For reasons that will become apparent soon, our transformation into identification protocols withstanding reset attacks merely applies to schemes of the former group in general.

For our transformation we exploit fundamental techniques introduced in the work by Canetti et al. [CGGM00] about resettable zero-knowledge. The idea is to adjust the methods in [CGGM00] to coordinate them with the ones providing security against active attacks for the corresponding identification scheme. We explain how to accomplish this.

The aforementioned proof-of-knowledge-based identification schemes all come in canonical commitment-challenge-response structure. That is, the prover first commits to a random value, then the verifier sends a random challenge, and the prover responds to this challenge according to his secret key and the initial commitment.

We would like to ensure that the adversary does not gain any advantage from resetting the prover. To this end, we insert a new first round into the identification protocol in which the verifier non-interactively commits to his challenge. The parameters for this commitment scheme become part of the public key. This keeps the adversary from resetting the prover to the challenge-message and completing the protocol with different challenges —a step that would compromise the security of all protocols above.

In addition, we let the prover determine the random values in his commitment (the first move in the original scheme) by applying a pseudorandom function to the verifier's commitment. Now, if the adversary resets the prover (with the same random tape) to the outset of the protocol and commits to a different challenge then the prover likewise answers with a commitment of a virtually independent random value, although having the same random tape. On the other side, using

[1] Apparently, without another restraint all schemes are insecure against adversaries talking to the prover while identifying, e.g., the man-in-the-middle-adversary simply forwarding messages of the verifier and the prover would mount such an attack and would break all schemes. Among others, the technical restriction excludes this adversary.

pseudorandom values instead of truly random coins does not weaken the original identification protocol noticeably.

The modifications above essentially prune the resetting adversary to an active, non-resetting intruder which is still allowed parallel runs with the prover while or before trying to intrude (and which, as we will show, also satisfies the additional technical property in case of a simultaneous intrusion). This is the reason for demanding security against such nested attacks for the basic scheme.

Recall that security proofs against active and interlacing adversaries normally require the simulated verifier to rewind the adversary and to repeat the adversary's identification attempt with different challenges. But we have just pinned down the verifier's challenge with the non-interactive commitment to overcome adversarial reset attacks! The way out are trapdoor commitment schemes: if one knows the trapdoor of such a commitment scheme then one can open any commitment with any value, but if one lacks knowledge of the trapdoor then the scheme is still binding. The idea is to let the simulator use the trapdoor, but not the adversary.

In the CR1 setting where the adversary finishes the executions with the prover before talking to the verifier it is easy to guarantee that the adversary will not abuse "our" trapdoor (basically, because he would have to do so before we disclose the trapdoor and deploy it ourself). To prevent the adversary from benefitting from our trapdoor in the CR2 model where the adversary's infiltration and prover executions run concurrently, we utilize something that we need anyway to define and show security in this setting: session IDs [BPR00]. A session ID assigns each execution of the identification protocol a unique number, and an adversary is only called successful if he passes the verification for a fresh session ID (where IDs in a protocol execution between the prover and the adversary are considered as used up).

In the CR2 setting we make the trapdoor in our commitment scheme depend on the session ID in the adversary's identification attempt. This basically means that either the adversary forgos using the trapdoor at all, or cannot complete a successful attack according to the definition if he uses this ID in a run with the prover. If he does not use the trapdoor ID, then all his initial commitments are binding and the argument before applies.

Efficient trapdoor commitment schemes for our prupose exist for example under the discrete-log assumption, the RSA assumption or intractability of factoring (see Chapter 3). Thus, we can build them from the same assumptions that the corresponding identification schemes rely on. The computational complexity of the assembled scheme roughly doubles compared to the original protocol.

## 2. Definitions

In this section we define secure resettable identification protocols and pseudorandom functions and briefly recall identity-based trapdoor commitments.

### 2.1. Resettable Identification Protocols

An identification protocol is an interactive protocol between two parties, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$. At the beginning a random key pair $(\mathrm{sk}, \mathrm{pk})$ is generated according to some efficiently samplable distribution, and the prover is given both the secret key sk as well as the public key pk as input, whereas the verifier only gets the public key pk. At the end of the execution, the verifier either outputs `accept` or `reject`, indicating a successful identification or failure.

A *passive* attacker on an identification protocol replaces the prover $\mathcal{P}$ in an execution with the honest verifier $\mathcal{V}$, but is given only the publik key pk and not the secret key sk. The aim of the attacker is to make $\mathcal{V}$ accept and impersonate as the prover. An *active* adversary may first perform sequential executions with the prover before enaging in a run with the verifier in order to deduce some information facilitating the intrusion. These two adversary types and further ones are summerized in Figure 1.

FIGURE 1. Adversary Types for Identification Schemes

| adversary type | interactions with prover $\mathcal{P}$ | interactions with $\mathcal{P}$ before/while intrusion | session IDs |
|---|---|---|---|
| passive | none | — | no |
| active | sequential | before | no |
| non-resetting CR1 | concurrent | before | no |
| non-resetting CR2 | concurrent | while | yes |
| CR1 | concurrent & resettable | before | no |
| CR2 | concurrent & resettable | while | yes |

In the resetting model the adversary is allowed to reset any execution with the prover. By this, the prover keeps the same random tape as in the previous run, but otherwise the protocol starts from scratch with one party sending the initial message. Formally, we allow the adversary to send a message `reset` to the prover (and appending an initial message if the adversary in the role of the verifier starts identifications). Consequently, the prover restarts the protocol having the

same random tape. Note that in this model the adversary can indirectly reset the prover to any round of an execution by resetting the prover to the outset and submitting the same communication up to this round again.

A CR1 adversary is an active adversary that is granted concurrent and resettable executions with the prover before starting the intrusion attempt. If the adversary never submits a reset-message to the prover then he is called a *non-resetting* CR1 adversary. Again, see Figure 1.

As explained in the introduction, for the CR2 setting we incorporate session IDs into identification protocols. That is, the verifier assigns each run a session ID of bit length $s$ such that the probability that the same ID appears twice is very small, e.g., it should be negligible in a polynomial number of executions. Hence, theoretically, session IDs should be about the size of the security parameter when selected at random. In practice, for fixed security parameter, 80 bits should be enough, or, if the verifier is stateful, then a counter or the current time will work.

We assume that, upon stopping, the prover and verifier output either `reject`, indicating an error, or `accept` together with an $s$-bit value SID. In a good execution, both parties should accept and agree on the same value SID. That is, in an execution between the honest prover $\mathcal{P}$ and the honest verifier $\mathcal{V}$, the verifier always outputs `accept` and both parties output the same value SID.

Next we define the adversary's success probability. In the sequel, let RID be an identification protocol and $\mathcal{A}$ an adversary attacking the protocol in one of the ways described above. Denote by $\mathsf{Intr}^{\mathrm{cr1}}_{\mathrm{RID}}(\mathcal{A})$ the probability that, finally, the verifier outputs `accept` in a CR1 attack and the adversary successfully intrudes. Write $\mathsf{Intr}^{\mathrm{cr2}}_{\mathrm{RID}}(\mathcal{A})$ for the corresponding probability in the CR2 model, where the adversary is only said to intrude victoriously if the verifier outputs `accept` and a session ID SID which no prover instance has ever output together with `accept` during the attack.[2] Denote by $\mathsf{Intr}^{\mathrm{nrcr1}}_{\mathrm{RID}}(\mathcal{A})$ and $\mathsf{Intr}^{\mathrm{nrcr2}}_{\mathrm{RID}}(\mathcal{A})$ the adversary's success probability in a non-resetting CR1 and CR2 attack, respectively. Note that all algorithms and all probabilities implicitly depend on a security parameter.

**Definition 5.1** (Secure Identification Protocol)**.** *An identification protocol* RID *is called*

- *non-resetting* CR1*-secure if* $\mathsf{Intr}^{\mathrm{nrcr1}}_{\mathrm{RID}}(\mathcal{A})$ *is negligible for any probabilistic polynomial-time non-resetting* CR1 *adversary* $\mathcal{A}$*,*

[2] Bellare et al. [BGM00] let both the prover and verifier determine the session ID and call the adversary also successful if he manages to make different copies of the prover-adversary interactions end up with the same ID. We do not consider this a successful attack, because the verifier's list of session IDs (maintained with respect to all interactions in a multi-user network) is not affected by such ID-collisions between the prover and the adversary. For example, protocols like SSL and TLS [T00] confirm this viewpoint: only the server, alias verifier, determines the session ID (although the ID serves rather administrative purposes in SSL and TLS).

- *non-resetting* CR2*-secure if* $\mathsf{Intr}^{\mathrm{nrcr2}}_{\mathrm{RID}}(\mathcal{A})$ *is negligible for any probabilistic polynomial-time non-resetting* CR2 *adversary* $\mathcal{A}$*,*
- CR1*-secure if* $\mathsf{Intr}^{\mathrm{cr1}}_{\mathrm{RID}}(\mathcal{A})$ *is negligible for any probabilistic polynomial-time* CR1 *adversary* $\mathcal{A}$*,*
- CR2*-secure if* $\mathsf{Intr}^{\mathrm{cr2}}_{\mathrm{RID}}(\mathcal{A})$ *is negligible for any probabilistic polynomial-time* CR2 *adversary* $\mathcal{A}$*.*

We sometimes refer to the time complexity $t$ and query complexity $q$ of adversary $\mathcal{A}$ attacking an identification scheme. Then $t$ reflects to the number of steps in some computational model like the Turing machine model. It also includes the steps of the prover and verifier since in an actual attack the adversary has to wait for the other parties' replies, too. The query complexity denotes the number of runs with the prover, counting each new or reset execution individually.

Note that in the descriptions of the attacks above the adversary tries to pass the examination of *a single* verifier in *a single* try. More generally, we could allow $v$ parallel accessible verifier incarnations. A CR2-secure protocol remains secure in this case: an adversary in the multiple-verifier case can be transformed into one for the single-verifier setting, although the success probability drops by a factor $1/v^2$ at most.

Extending the notions of reset attacks, we could introduce a CR3 model in which the adversary is allowed to reset the verifier(s), too. While the CR2-secure schemes in [BGM00] also satisfy this stronger notion, our protocols are not known to remain secure in this case. Thus, we leave out further discussions about this model.

### 2.2. Pseudorandom Functions

One of the primitives we utilize for our transformation are pseudorandom functions. These are the practical counterparts to truly random functions mapping each input to an independent random output. And while random function must have exponential description complexity, pseudorandom functions are efficiently computable.

We keep the definition as simple as possible for our purpose. A function family is a function $\mathrm{PRF}(\mathsf{eval}, \cdot, \cdot)$ in two arguments. For security parameter $n$ the first argument, called the key, has $n$ bits and defines in a straightforward way a function $\mathrm{PRF}(\mathsf{eval}, \kappa, \cdot)$ for any $\kappa \in \{0,1\}^n$. For every $\kappa \in \{0,1\}^n$ the function $\mathrm{PRF}(\mathsf{eval}, \kappa, \cdot)$ has input and output length $inl(n)$ and $outl(n)$ and can be efficiently computed given $\kappa$ and the input. The actual choices of $inl(\cdot)$ and $outl(\cdot)$ depend on the application and can be varied given arbitrary pseudorandom functions with certain input/output size [BCK96].

We adopt the definition of pseudorandom functions being indistinguishable from random functions [GGM86], i.e., it should be infeasible to detect whether

one communicates with a pseudorandom or a truly random function. For an algorithm $\mathcal{D}$ with oracle access consider the following random variables:

- $\mathcal{D}^{\text{prf}}$ equals the single bit output by $\mathcal{D}$ given oracle access to a function $\text{PRF}(\mathsf{eval}, \kappa, \cdot)$ for random $\kappa \in \{0,1\}^n$ and where $\mathcal{D}$ is allowed to adaptively query the oracle,
- $\mathcal{D}^{\text{rand}}$ is the single bit returned by $\mathcal{D}$ if instead a truly random function with input/output length $inl(n)$ and $outl(n)$ is given as oracle.

Define the distinguishing advantage of $\mathcal{D}$ as

$$\mathsf{Dist}_{\text{PRF}}(\mathcal{D}) := \left|\text{Prob}\left[\mathcal{D}^{\text{prf}} = 1\right] - \text{Prob}\left[\mathcal{D}^{\text{rand}} = 1\right]\right|$$

and let $\mathcal{D}$'s time and query complexity denote the running time of $\mathcal{D}$ and the maximal number $q$ of queries that $\mathcal{D}$ makes in either experiment. Intuitively, since pseudorandom functions are supposedly indistinguishable from random functions, the advantage should be very small:

**Definition 5.2.** *A pseudorandom function family* PRF *is a function family such that* $\mathsf{Dist}_{\text{PRF}}(\mathcal{D})$ *is negligible for any probabilistic poylnomial-time algorithm* $\mathcal{D}$*.*

We also set $\mathsf{maxDist}_{\text{PRF}}(t, q)$ to be the maximum $\mathsf{Dist}_{\text{PRF}}(\mathcal{D})$ over all $\mathcal{D}$ with time complexity $t$ and query complexity $q$.

### 2.3. Trapdoor Commitment Schemes

A non-interactive trapdoor commitment scheme TDC specifies a key generation algorithm $(\text{pk}_{\text{TDC}}, \text{sk}_{\text{TDC}}) \leftarrow \text{TDC}(\mathsf{keygen})$, a commitment algorithm $\text{TDCOM} \leftarrow \text{TDC}(\mathsf{commit}, \text{pk}_{\text{TDC}}, c)$ for value $c$, and a faking algorithm $(c', r'), \leftarrow \text{TDC}(\mathsf{fake}, \text{sk}_{\text{TDC}}, c, r, c')$ that allows to open a commitment with any value $c'$ given the secret key (and the value $c$ and the randomness $r$ used to produce the original commitment). We demand that a commitment and such a faked opening is identically distributed to a commitment with the correct opening for the same value $c'$, i.e., the trapdoor scheme is perfectly simulative. In particular, this implies that the commitment scheme provides perfect secrecy, in other words, a commitment is distributed independently of the actual value. Without knowing the secret trapdoor a commitment is still solidly binding, i.e., it should be infeasible to find $(c, r)$ and $(c', r')$ that map to the same commitment under $\text{pk}_{\text{TDC}}$ but such that $c \neq c'$.

For an identity-based trapdoor commitment scheme the key generation algorithm additionally returns a uniformly distributed string $\text{ID}_{\text{TDC}}$ as part of the secret key. Yet, the public key is distributed independently of this string $\text{ID}_{\text{TDC}}$. The commitment algorithm $\text{TDC}(\mathsf{commit}, \text{pk}_{\text{TDC}}, \cdot)$ now takes as input a string ID, a value $c$ and randomness $r$ and returns a commitment.

Security for identity-based trapdoor commitment schemes is defined with respect to a collision-finder that gets $\text{pk}_{\text{TDC}}$, $\text{sk}_{\text{TDC}}$ and $\text{ID}_{\text{TDC}}$ as input and is considered to win if he outputs a commitment with valid openings for two different

values $c, c'$ and the same ID, where ID is different from the output $\text{ID}_{\text{TDC}}$ of the key generation algorithm. The probability that any efficient algorithm wins should be negligible. Put differently, the trapdoor property is tied to $\text{ID}_{\text{TDC}}$ and does not help to overcome the binding property for other IDs.

Both for ordinary as well as for identity-based trapdoor schemes, write $\mathsf{Coll}_{\text{TDC}}(\mathcal{C})$ for the probability of collision-finder $\mathcal{C}$ outputting a commitment with ambiguous decommitments (in the identity-based case for ID different than $\text{ID}_{\text{TDC}}$). For a trapdoor commitment TDC scheme we demand that this probability is negligible for any efficient collision-finder $\mathcal{C}$. Let $\mathsf{maxColl}_{\text{TDC}}(t)$ be the maximum over $\mathsf{Coll}_{\text{TDC}}(\mathcal{C})$ for all algorithms $\mathcal{C}$ running in time $t$.

As an example of an identity-based trapdoor commitment scheme we review the discrete-log-based solution of Section 3 in Chapter 3. The public key consists of a group of prime order $q$ and two random generators $g_1, g_2$ of the group, as well as another generator $g_3$. The latter generator is defined by $g_3 = g_1^{-\text{ID}_{\text{TDC}}} g_2^z$ for random $\text{ID}_{\text{TDC}}$ and random $z \in \mathbb{Z}_q$. Clearly, $g_3$ hides $\text{ID}_{\text{TDC}}$ information-thereotically. A commitment to $(\text{ID}, c, r)$ is defined by $(g_1^{\text{ID}} g_3)^c g_2^r$. The trapdoor $\text{sk}_{\text{TDC}}$ equals $\text{ID}_{\text{TDC}}$ and $z$. Because $g_1^{\text{ID}_{\text{TDC}}} g_3 = g_2^z$ it is easy to adapt the opening for any commitment involving $\text{ID}_{\text{TDC}}$ by the discrete-log trapdoor property. On the other side, for distinct $c \neq c'$ an ambiguous decommitment $(c, r)$, $(c', r')$ for the same ID different than $\text{ID}_{\text{TDC}}$ yields $\log_{g_1} g_2$, contradicting the discrete-log assumption.

## 3. Secure Identification in the CR1 Setting

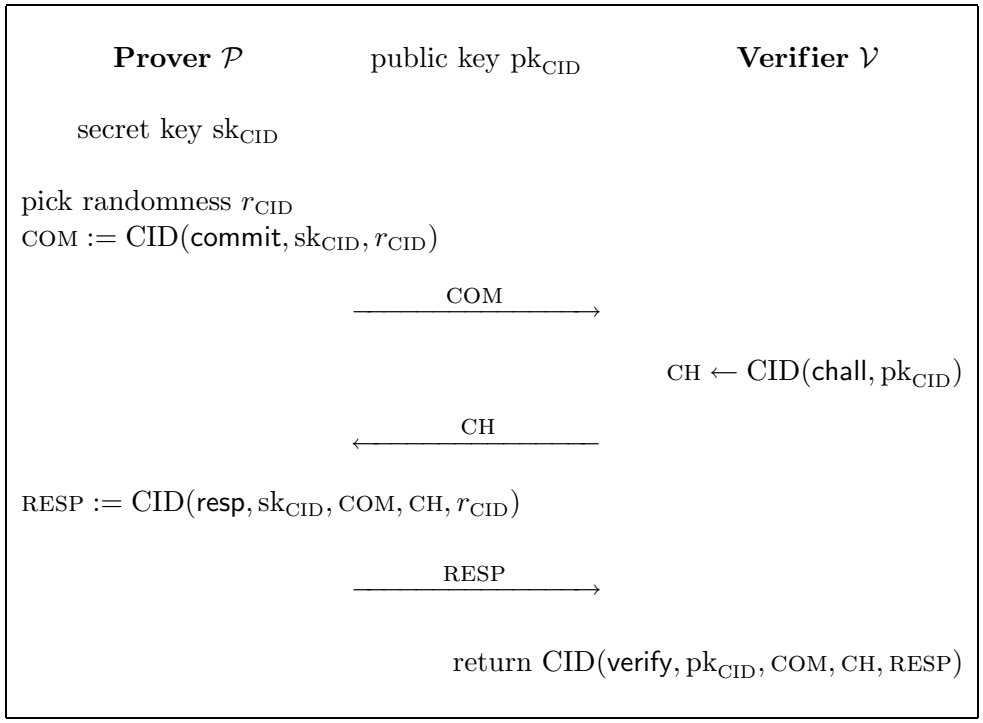
As discussed in the introduction, proof-of-knowledge-based identification protocols of the Fiat-Shamir type cannot be secure against reset attacks. In this section, however, we present a general transformation of such identification schemes into secure ones in the CR1 setting.

### 3.1. Canonical Identification Protocols

We start with identification schemes that consists of three moves, an initial commitment COM of the prover, a random challenge $\text{CH} \in \{0,1\}^{chl}$ of $chl$ bits of the verifier, and a conclusive response RESP from the prover. We call a protocol obeying this structure a CID-identification scheme, and we denote the algorithms generating the commitment, challenge and response message by $\text{CID}(\mathsf{commit}, \dots)$, $\text{CID}(\mathsf{chall}, \dots)$, $\text{CID}(\mathsf{resp}, \dots)$ and the verification step by $\text{CID}(\mathsf{verify}, \dots)$. See Figure 2. It is crucial to our construction that the challenge depends only on the public key but not the prover's commitment.

Loosely speaking, we will assume that the underlying CID-scheme is secure against non-resetting attacks in the CR1 model, i.e., against attacks where the adversary merely runs concurrent sessions with prover without resets before engaging

FIGURE 2. Canonical Identification Protocol CID




in a verification. In addition to the Feige-Fiat-Shamir system [FFS88], most of the well-known practical identification schemes achieve this security level, for example Ong-Schnorr [OS90, S96] for some system parameters, Okamoto-Guillou-Quisquater [GQ88, O92] and Okamoto-Schnorr [S91, O92]. Nonetheless, there are also protocols which are only known to be secure against sequential attacks (e.g. [S99]).

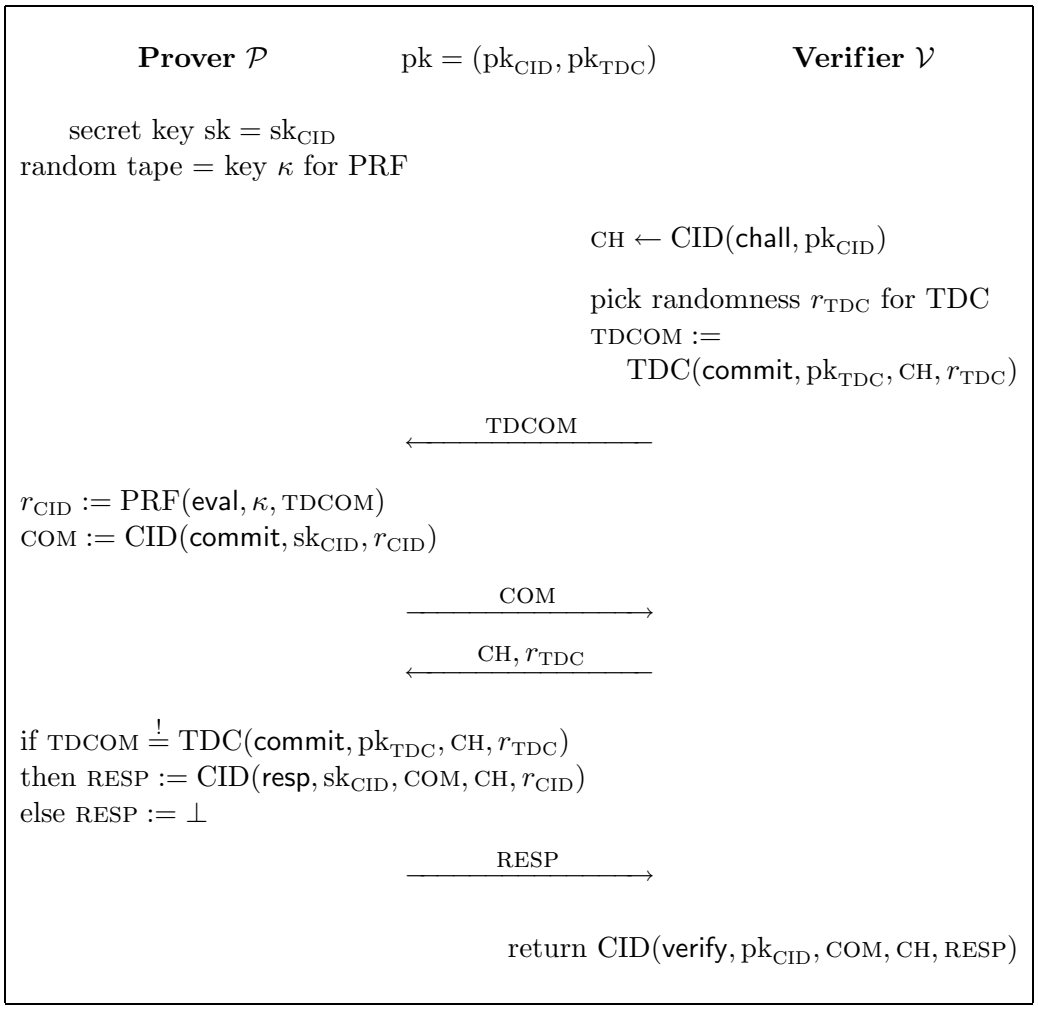
### 3.2. Construction of CR1-Secure Scheme

Our solution originates from the work by Canetti et al. [CGGM00] about resettable zero-knowledge. Recall from the introduction that, in order to decrease the power of a CR1 adversary to the one of a non-resetting CR1 adversary, we let the verifier commit to the challenge in a new initial step and let the prover compute the randomness for the execution by applying a pseudorandom function to the verifier's commitment. The remaining three rounds are as before, except that now the prover also checks the validity of the verifier's decommitment before sending the final response. The scheme is displayed in Figure 3.

For the verifier's commitment to the challenge, we use use a trapdoor commitment scheme TDC. This enables us to reduce an intrusion try of an impersonator $\mathcal{A}$ in the derived scheme RID to one for the CID-protocol. If $\mathcal{A}$ initiates a session

with the verifier in RID then we can first commit to a dummy value $0^{chl}$ without having to communicate with the verifier in CID. When $\mathcal{A}$ then takes the next step by sending COM, we forward this commitment to our verifier in CID and learn the verifier's challenge. Knowing the secret key $\mathrm{sk}_{\mathrm{TDC}}$ for the trapdoor scheme we can then find a valid opening for our dummy commitment with respect to the challenge. Finally, we forward $\mathcal{A}$'s response in our attack.

FIGURE 3. Secure Identification Protocol RID in the CR1 model




Note that the public key in our identification scheme consists of two independent parts, $\mathrm{pk}_{\mathrm{CID}}$ and $\mathrm{pk}_{\mathrm{TDC}}$. For concrete schemes the key generation may be combined and simplified. For instance, for Okamoto-Schnorr the public key of the identification protocol describes a group of prime order $q$, two generators $g_1, g_2$ of that group and the public key $X = g_1^{x_1} g_2^{x_2}$ for secret $x_1, x_2 \in \mathbb{Z}_q$. The

prover sends $\text{COM} = g_1^{r_1} g_2^{r_2}$ and replies to the challenge $\text{CH} \in \mathbb{Z}_q$ by transmitting $y_i = r_i + \text{CH} \cdot x_i \bmod q$ for $i = 1, 2$. In this case, the public key for the trapdoor commitment scheme could be given by $g_1, g_3 = g_1^z$ for random trapdoor $z \in \mathbb{Z}_q$, and the commitment function maps a value $c$ and randomness $r$ to $g_1^c g_3^r$.

### 3.3. Security Analysis

The discussion in the previous section indicates that any adversary $\mathcal{A}$ for RID does not have much more power than a non-resetting impersonator attacking CID and security of RID follows from the security of CID. We now state and prove this formally, where we consider both concrete as well as asymptotic security:

**Theorem 5.3.** *Let* CID *be a CID-identification protocol. Also, let* PRF *be a pseudorandom function family and denote by* TDC *a perfectly-simulative trapdoor commitment scheme. Let* RID *be the associated identification scheme as per Figure 3. If $\mathcal{A}$ is a* CR1 *adversary of time complexity $t$ and query complexity $q$ attacking* RID *then there exists a non-resetting* CR1 *adversary $\mathcal{A}_{\text{CID}}$ attacking* CID *such that*

$$\mathsf{Intr}^{\text{cr1}}_{\text{RID}}(\mathcal{A}) \;\leq\; q \cdot \mathsf{maxDist}_{\text{PRF}}(t, q) \;+\; \mathsf{maxColl}_{\text{TDC}}(t) \;+\; \mathsf{Intr}^{\text{nrcr1}}_{\text{CID}}(\mathcal{A}_{\text{CID}})$$

*Furthermore, $\mathcal{A}_{\text{CID}}$ has time complexity $t$ and query complexity $q$.*

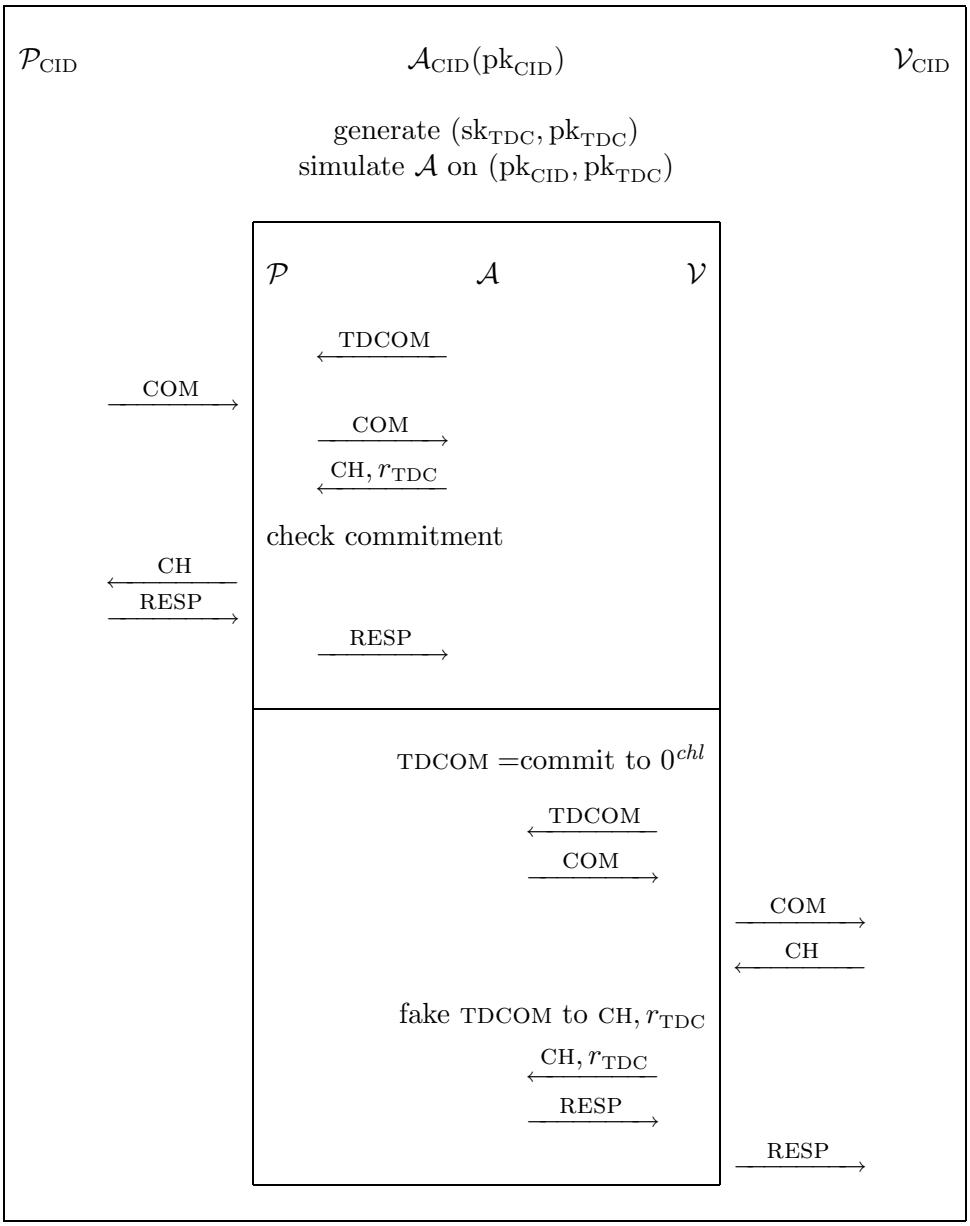
The notion of concrete security allows to deduce the exact security level by plugging in the corresponding figures of the underlying primitives. Concerning asymptotic behavior, for polynomially bounded $t = t(n)$ and $q = q(n)$ the terms $q \cdot \mathsf{maxDist}_{\text{PRF}}(t, q)$ and $\mathsf{maxColl}_{\text{TDC}}(t)$ are negligible for secure pseudorandom functions and trapdoor commitments, therefore:

**Corollary 5.4.** *Let* PRF *be a pseudorandom function family and let* TDC *be a perfectly-simulative trapdoor commitment scheme. If* CID *is a non-resetting* CR1*-secure CID-identification protocol, then the associated identification scheme* RID *in Figure 3 is* CR1*-secure.*

**Proof (of Theorem 5.3).** Figure 4 shows the adversary $\mathcal{A}_{\text{CID}}$ attacking the CID-identification protocol in the non-resetting CR1 model (some details are omitted in the figure and are explained below). This algorithm gets $\text{pk}_{\text{CID}}$ as input and tries to pass the verifier's examination by running the adversary $\mathcal{A}$ for RID as a subroutine.

Algorithm $\mathcal{A}_{\text{CID}}$ basically simulates the CR1-adversary $\mathcal{A}$ with the CID-protocol by assimilating all additional steps of RID. Specifically, $\mathcal{A}_{\text{CID}}$ generates a random key pair $(\text{pk}_{\text{TDC}}, \text{sk}_{\text{TDC}})$ of the trapdoor commitment scheme and starts the simulation of $\mathcal{A}$ on $\text{pk}_{\text{CID}}$ and $\text{pk}_{\text{TDC}}$. If this algorithm $\mathcal{A}$ commits to some TDCOM in some instance with the prover then $\mathcal{A}_{\text{CID}}$ calls the prover in CID to obtain COM and passes this commitment on to $\mathcal{A}$. If $\mathcal{A}$ opens a commitment TDCOM then $\mathcal{A}_{\text{CID}}$ checks the correctness; if the opening is valid then forward the challenge

FIGURE 4. Non-Resetting CR1 adversary for CID from $\mathcal{A}$




to the prover and hand the answer to $\mathcal{A}$. If the decommitment is incorrect then return $\perp$ to $\mathcal{A}$ without involving the prover. For a correct decommitment $\mathcal{A}_{\text{CID}}$ fetches the prover's response for the challenge and hands it to $\mathcal{A}$.

When $\mathcal{A}$ finishes the phase with the prover and starts an execution with the verifier, $\mathcal{A}_{\text{CID}}$ commits to a dummy value $0^{chl}$. Then $\mathcal{A}$ sends a commitment to the verifier which $\mathcal{A}_{\text{CID}}$ passes to the verifier in CID to obtain a challenge CH from the verifier. Exploiting the trapdoor property and knowledge of $\text{sk}_{\text{TDC}}$, adversary $\mathcal{A}_{\text{CID}}$ finds an appropriate opening for this challenge $\text{CH}_{\mathcal{V}}$ for the dummy commitment. Note that this decommitment is identically distributed as if $\mathcal{A}_{\text{CID}}$ would have committed to CH right away. $\mathcal{A}_{\text{CID}}$ gives this decommitment to $\mathcal{A}$ and returns the answer to the verifier in CID.

In contrast to the prover in protocol RID, the prover in CID uses random coins instead of a pseudorandom function. The first step is to verify that pseudorandom values $r_{\text{CID}} := \text{PRF}(\mathsf{eval}, \kappa, \text{TDCOM})$ instead of truly random $r_{\text{CID}}$ do not help $\mathcal{A}$ too much. To this end, we recall the hybrid model of [CGGM00] in which we replace the pseudorandom function by a random one. Namely, given protocol RID in the CR1 setting we denote by $\text{RID}^{\text{RAND}}$ the identification scheme in which each prover instance, instead of applying a pseudorandom function to TDCOM, evaluates a random function on this value, where an independent function is selected for each prover incarnation. Although random functions are not efficiently computable, they can be simulated by assigning each new argument an independent random string, and by repeating previously given answers for the same queries. The next claim relates the advantage the adversary $\mathcal{A}$ might gain in RID compared to $\text{RID}^{\text{RAND}}$ to the pseudorandomness of PRF:

Claim 1: Let RID be the identification protocol in Figure 3 and let PRF be a pseudorandom function family. If $\mathcal{A}$ is an adversary of time complexity $t$ and query complexity $q$ attacking RID in the CR1 setting then

$$\mathsf{Intr}^{\text{cr1}}_{\text{RID}}(\mathcal{A}) \;\leq\; q \cdot \mathsf{maxDist}_{\text{PRF}}(t, q) \;+\; \mathsf{Intr}^{\text{cr1}}_{\text{RID}^{\text{RAND}}}(\mathcal{A}).$$

Proof. Given an adversary $\mathcal{A}$ we construct a distinguisher $\mathcal{D}$ for the pseudorandom function ensemble PRF as follows. $\mathcal{D}$ essentially plays the role of the honest parties, i.e., the prover and verifier, but is given oracle access to a sequence of functions $f_1, \ldots, f_q$ which are either pseudorandom or truly random. $\mathcal{D}$ generates a random key pair $(\text{pk}, \text{sk})$ for RID and starts to emulate the attack. This is done by performing all steps of the prover's incarnations and the verifier as defined by the protocol, except for the step where some prover instance $i$ is supposed to compute $r_{\text{CID}} := \text{PRF}(\mathsf{eval}, \kappa, \text{TDCOM})$. Instead, algorithm $\mathcal{D}$ replies by querying oracle $f_i$ about TDCOM and continuing this prover's simulation for the answer $r_{\text{CID}}$. The distinguisher outputs 1 if and only if the adversary is successful.

Clearly, if $f_1, \ldots, f_q$ is a sequence of pseudorandom functions then $\mathcal{D}$ outputs 1 exactly if the adversary breaks RID. On the other hand, if the functions are truly random then $\mathcal{D}$ outputs 1 if and only if the adversary breaks $\text{RID}^{\text{RAND}}$. The running time of $\mathcal{D}$ is bounded by $t$ and the number of queries is at most $q$. An hybrid argument now shows that this yields an algorithm distinguishing a single

pseudorandom function from PRF and a random one; the distinguishing advantage drops by a factor $q$ at most (see [BCK96]). $\diamond$

Hence, if the adversary $\mathcal{A}$ never queries a prover copy for the same prefix twice, then the hybrid scheme corresponds to the setting where runs with the prover (even reset ones) involve an independent random tape, like the prover instances in CID. Because such double queries can be easily eliminated by table-lookup techniques, we assume in the sequel for simplicity that $\mathcal{A}$ never sends the same message to the same prover instance twice.

Next we bound the probability that $\mathcal{A}$ finds distinct openings to a commitment TDCOM sent to the prover in $\mathrm{RID}^{\mathrm{RAND}}$ by the maximal probability $\mathsf{maxColl}_{\mathrm{TDC}}(t)$ of an algorithm finding a commitment with ambiguous decommitments and running in time $t$. If this does not happen then $\mathcal{A}$ virtually mounts a non-resetting CR1 attack on $\mathrm{RID}^{\mathrm{RAND}}$, and therefore $\mathcal{A}_{\mathrm{CID}}$ a corresponing attack on CID.

Claim 2: If $\mathcal{A}$ is an adversary of time complexity $t$ and query complexity $q$ attacking $\mathrm{RID}^{\mathrm{RAND}}$ in the CR1 setting then for $\mathcal{A}_{\mathrm{CID}}$ attacking CID as defined in Figure 4 we have

$$\mathsf{Intr}^{\mathrm{cr1}}_{\mathrm{RID}^{\mathrm{RAND}}}(\mathcal{A}) \;\leq\; \mathsf{maxColl}_{\mathrm{TDC}}(t) \;+\; \mathsf{Intr}^{\mathrm{nrcr1}}_{\mathrm{CID}}(\mathcal{A}_{\mathrm{CID}}).$$

Proof. Conditioning on the event Unambiguity that the impersonator $\mathcal{A}$ does not send TDCOM with two valid decommitments to some prover incarnation, it is clear that $\mathcal{A}$ runs a non-resetting CR1 attack only. In this case, adversary $\mathcal{A}_{\mathrm{CID}}$ wins whenever $\mathcal{A}$ wins. It therefore suffices to bound the probability of event $\neg$ Unambiguity.

We claim that Prob[$\neg$ Unambiguity] is at most $\mathsf{maxColl}_{\mathrm{TDC}}(t)$. This can be seen as follows. Given a public key $\mathrm{pk}_{\mathrm{TDC}}$ of the trapdoor commitment scheme we choose a pair $(\mathrm{pk}_{\mathrm{CID}}, \mathrm{sk}_{\mathrm{CID}})$ for the identification protocol and run an attack of $\mathcal{A}$ on $\mathrm{RID}^{\mathrm{RAND}}$ by impersonating the honest prover and verifier. If $\mathcal{A}$ outputs a commitment TDCOM with distinct openings with respect to $\mathrm{pk}_{\mathrm{TDC}}$ then we output this commitment with the openings, too. Apparently, the probability that we find such ambiguous decommitments equals the probability Prob[$\neg$ Unambiguity], and the running time of our algorithm is bounded by $t$. This completes the proof. $\diamond$

Collecting the probabilities from Claims 1 and 2 yields the theorem. $\square$

## 4. Secure Identification in the CR2 Setting

We modify the CR1-secure identification scheme in Section 3 to achieve CR2-security. Recall that this requires session IDs and that the only way for the adversary to win is by passing the verifier's examination for a fresh session ID.

### 4.1. Construction of CR2-Secure Scheme

The key to accomplish CR2-security lies in the extension of the trapdoor commitment scheme to an identity-based one where we use the session IDs of the identification protocols as the IDs in the trapdoor commitments. Roughly, an identity-based trapdoor commitment schemes links a session ID to the trapdoor property. So if we simulate the adversary $\mathcal{A}$ to derive an impersonater for $\mathcal{A}_{\text{CID}}$, as done in the CR1 setting, we can later use the previously generated $\text{SID}_{\text{TDC}}$ in the adversary's intrusion attempt. This means that the adversary cannot use this session ID in its executions with the prover (otherwise the adversary is not considered victorious according to the definition). But if the impersonator forgos using $\text{SID}_{\text{TDC}}$ then all his commitments for other session IDs are binding and a similar argument to the one in the CR1 model applies. Since the public key of the trapdoor scheme hides $\text{SID}_{\text{TDC}}$ perfectly, we can later claim that the verifier has chosen $\text{SID}_{\text{TDC}}$ only then. The protocol is shown in Figure 5.

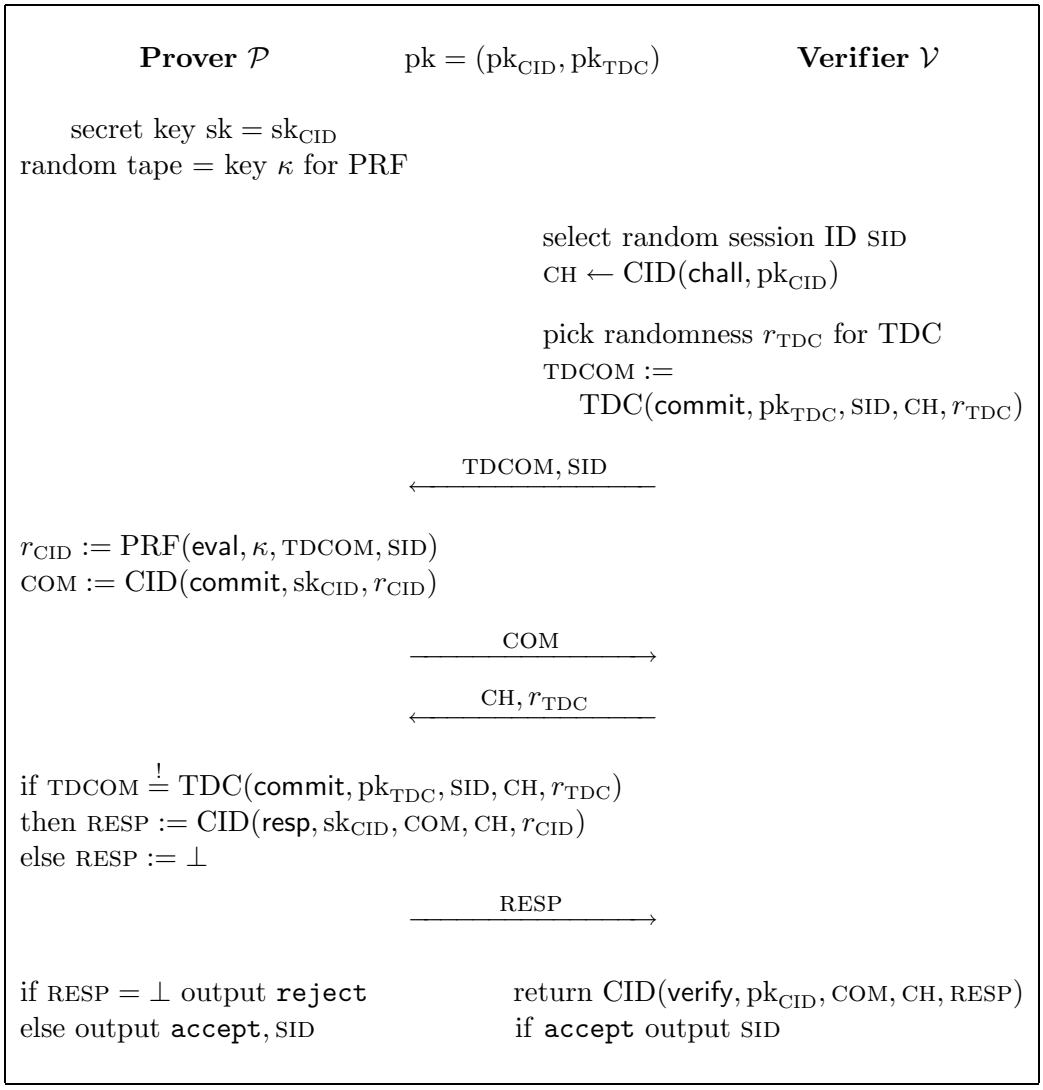
### 4.2. Security Analysis

The difference to the CR1 setting is that the impersonator $\mathcal{A}$ may now interleave the execution with the verifier and the ones with prover. Although CID-protocols fail to be secure against such attacks in general, e.g., the man-in-the-middle adversary breaks such schemes in this setting, luckily they are still secure under a technical restriction on the adversary. Therefore, we will still be able to reduce security to CID-protocols.

To specify the condition under which CID-schemes remain secure, consider an execution of an impersonator $\mathcal{A}_{\text{CID}}$ attacking CID in a non-resetting CR2 attack. At some step the verifier sends a random challenge $\text{CH}_{\mathcal{V}}$ to $\mathcal{A}_{\text{CID}}$ and the adversary then finishes the attack, either successfully or not. The subscript $\mathcal{V}$ indicates that this is the challenge sent by $\mathcal{V}$ in an execution with the adversary. Define a *challenge repetition* to be the following action: reset the state of the prover, the adversary and the verifier to the point before sending $\text{CH}_{\mathcal{V}}$; then transmit another random $\text{CH}'_{\mathcal{V}}$ instead and continue the adversary's attack on this new challenge. The reason for considering such challenge repetitions is that they are normally used to prove security for CID-schemes, see for example [FFS88] for details.

Next we look at what happens on the prover's side in challenge repetitions. We are especially interested in so-called pending executions in which the prover had sent a commitment COM before the adversary has received $\text{CH}_{\mathcal{V}}$, and in which the impersonator has answered with some challenge CH in that execution with the prover after the verifier had sent $\text{CH}_{\mathcal{V}}$. This implies that after a challenge repetition the adversary may now decide to send a different challenge $\text{CH}'$ instead of CH. Figure 6 depicts the situation. We say that the impersonator never finishes an execution with the prover ambiguously if this never happens. We say that the impersonator $\mathcal{A}_{\text{CID}}$ is *chr-challenge-repeatable* if $\mathcal{A}_{\text{CID}}$ never finishes an

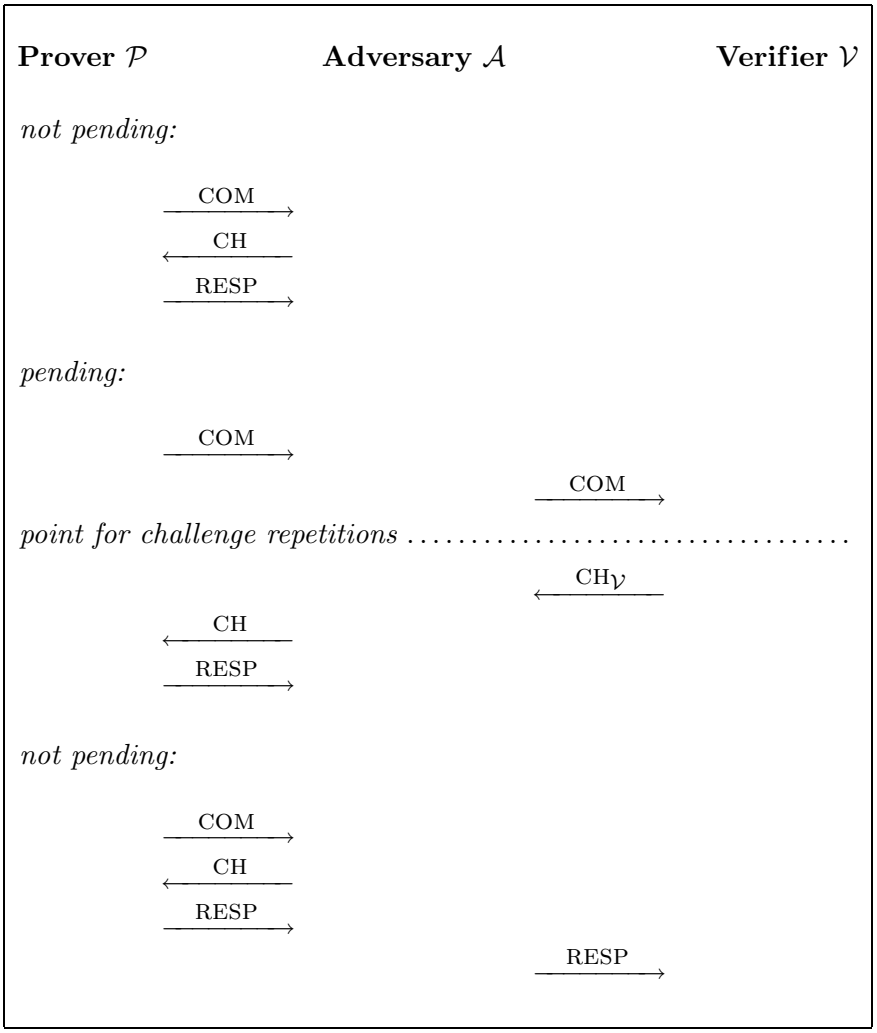
FIGURE 5. Secure Identification Protocol RID in the CR2 model




execution with the prover ambiguously, even if *chr* challenge repetitions occur. It is understood that a challenge-repeatable non-resetting CR2-secure CID-protocol refers to security against any polynomially-bounded, non-resetting CR2-adversary $\mathcal{A}_{\mathrm{CID}}$ which is *chr*-challenge-repeatable for any polynomial $chr(n)$.

To clarify the notion we consider two examples. No CID-scheme is even 2-challenge-repeatable for the man-in-the-middle adversary. The reason is such an adversary duplicates all messages of the prover and the verifier and if we execute a challenge repetitions then the adversary imitates this, too. In contrast, for any

FIGURE 6. Pending Executions in Challenge Repetitions for CID




non-resetting CR1-adversary any CID-protocol is challenge-repeatable because the executions with the prover are already finished when the intrusion try starts.

In comparison to the CR1-secure scheme, here the verifier chooses a random session ID and the identity-based trapdoor scheme is applied to commit to the challenge with the session ID at the beginning of an execution. The session ID is also transmitted in clear together with the commitment and the prover applies the pseudorandom function on both the verifier's trapdoor commitment and the session ID. Except for these modifications the rest of the protocol remains unchanged. The common session ID is set to the verifier's choice SID.

**Theorem 5.5.** *Let* CID *be a CID-identification protocol. Also, let* PRF *be a pseudorandom function family and denote by* TDC *a perfectly-simulative identity-based trapdoor commitment scheme. Let* RID *be the associated identification scheme as per Figure 5. If $\mathcal{A}$ is a* CR2 *adversary of time complexity $t$ and query complexity $q$ attacking* RID *then for any $chr$ there exists a $chr$-challenge-repeatable non-resetting* CR2 *adversary $\mathcal{A}_{\mathrm{CID}}$ attacking* CID *such that*

$$\mathsf{Intr}^{\mathrm{cr2}}_{\mathrm{RID}}(\mathcal{A}) \;\leq\; q \cdot \mathsf{maxDist}_{\mathrm{PRF}}(t, q) \;+\; \mathsf{maxColl}_{\mathrm{TDC}}(t \cdot chr) \;+\; \mathsf{Intr}^{\mathrm{nrcr2}}_{\mathrm{CID}}(\mathcal{A}_{\mathrm{CID}}).$$

For polynomially bounded $chr = chr(n)$ we have:

**Corollary 5.6.** *Let* PRF *be a pseudorandom function family and let* TDC *be a perfectly-simulative identity-based trapdoor commitment scheme. If* CID *is a challenge-repeatable non-resetting* CR2*-secure CID-identification protocol then the associated identification scheme* RID *in Figure 5 is* CR2*-secure.*

**Proof (of Theorem 5.5).** Because of the identity-based trapdoor commitment scheme the proof is almost identical to the one for the CR1 case. As long as the attacker $\mathcal{A}$ uses $\textsc{sid} \neq \textsc{sid}_{\mathrm{TDC}}$ in the executions with the prover, the initial commitment is binding. If $\mathcal{A}$ sends an initial commitment involving $\textsc{sid}_{\mathrm{TDC}}$ to the prover, e.g., after learning this session ID in the execution with the verifier, and later opens this initial commitment correctly, then this session ID is considered as used up and $\mathcal{A}$ cannot win anymore. Hence, if $\mathcal{A}_{\mathrm{CID}}$ stops whenever $\mathcal{A}$ transmits such a valid decommitment then $\mathcal{A}_{\mathrm{CID}}$'s success probability is not affected by this.

If we consider at most $chr$ challenge repetitions then $\mathcal{A}_{\mathrm{CID}}$ only finishes an execution with the prover ambiguously if $\mathcal{A}$ finds an ambiguous decommitment for some commitment given to the prover with respect to $\textsc{sid} \neq \textsc{sid}_{\mathrm{TDC}}$. The probability that this happens is at most $\mathsf{maxColl}_{\mathrm{TDC}}(t \cdot chr)$, because otherwise we could easily devise an algorithm simulating $\mathcal{A}$ and performing $chr$ challenge repetitions, each repetition taking time at most $t$, and outputting a commitment with distinct, valid openings for some $\textsc{sid} \neq \textsc{sid}_{\mathrm{TDC}}$. □

*Chapter 6*

# Universally Composable Commitments

This chapter deals with securely composable commitments schemes. It is a joint work with Ran Canetti; an extended abstract has been presented at Crypto 2001 [CF01]. We remark that this chapter does not discuss the recently announced constructions of such composable commitments by Damgård and Nielsen [DN01].

We thank Yehuda Lindell for suggesting to use non-malleable encryptions for achieving non-malleability of commitments in the common reference string model. This idea underlies our scheme that allows to reuse the common string for multiple commitments. (The same idea was independently suggested in [DKOS01].) We would also like to thank Roger Fischlin for help with the oblivious element generation in case of non-erasing parties.

## 1. Introduction

*Commitment* is one of the most basic and useful cryptographic primitives. On top of being intriguing by itself, it is an essential building block in many cryptographic protocols, such as Zero-Knowledge protocols (e.g., [GMW91, BCC88, D89]), general function evaluation protocols (e.g., [GMW87, GHY87, G00]), contract-signing and electronic commerce, and more. Indeed, commitment protocols have been studied extensively in the past two decades (e.g., [B82, N91, DDN00, NOVY98, B96, DIO98, FF00, DKOS01] ).

The basic idea behind the notion of commitment is attractively simple: A *committer* provides a *receiver* with the digital equivalent of a "sealed envelope" containing a value $x$. From this point on, the committer cannot change the value inside the envelope, and, as long as the committer does not assist the receiver in opening the envelope, the receiver learns nothing about $x$. When both parties cooperate, the value $x$ is retrieved in full.

Formalizing this intuitive idea is, however, non-trivial. Traditionally, two quite distinct basic flavors of commitment are formalized: *unconditionally binding* and *unconditionally secret* commitment protocols (see, e.g., [G98]). These basic definitions are indeed sufficient for some applications (see there). But they treat commitment as a "stand alone" task and do not in general guarantee security when a commitment protocol is used as a building-block within other protocols, or when multiple copies of a commitment protocol are carried out together. A good first example for the limitations of the basic definitions is the *selective decommitment* problem [DNRS99], that demonstrates our inability to prove some very minimal composition properties of the basic definitions.

Indeed, the basic definitions turned out to be inadequate in some scenarios, and stronger variants that allow to securely "compose" commitment protocols —both with the calling protocol and with other invocations of the commitment protocol— were proposed and successfully used in some specific contexts. One such family of variants make sure that knowledge of certain trapdoor information allows "opening" commitments in more than a single way. These include *chameleon commitments* [BCC88], *trapdoor commitments* [FS90] and *equivocable commitments* [B96]. Another strong variant is *non-malleable commitments* [DDN00], where it is guaranteed that a dishonest party that receives an unopened commitment to some value $x$ will be unable to commit to a value that depends on $x$ in any way, except for generating another commitment to $x$. (A more relaxed variant of the [DDN00] notion of non-malleability is *non-malleability with respect to opening* [DIO98, FF00, DKOS01].)

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

This work proposes a measure of security for commitment protocols that guarantees the "envelope-like" intuitive properties of commitment even when the commitment protocol is *concurrently composed* with an arbitrary set of protocols. In particular, protocols that satisfy this measure (called universally composable (UC) commitment protocols) remain secure even when an unbounded number of copies of the protocol are executed concurrently in an adversarially controlled way; they

are resilient to selective decommitment attacks; they are non-malleable both with respect to other copies of the same protocol and with respect to arbitrary commitment protocols. In general, a UC commitment protocol successfully emulates an "ideal commitment service" for any application protocol (be it a Zero-Knowledge protocol, a general function evaluation protocol, an e-commerce application, or any combination of the above).

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

### 1.1. On the new Measure

Providing meaningful security guarantees under composition with arbitrary protocols requires using an appropriate framework for representing and arguing about such protocols. Our treatment is based in a recently proposed such general framework [C01]. This framework builds on known definitions for function evaluation and general tasks [GL90, MR91, B91, PW94, C00a, DM00, PW01], and allows defining the security properties of practically any cryptographic task. Most importantly, in this framework security of protocols is maintained under general *concurrent* composition with an unbounded number of copies of arbitrary protocols. We briefly summarize the relevant properties of this framework. See more details in Section 2.1 and in [C01].

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

In order to allow proving the concurrent composition theorem, the notion of emulation in [C01] is considerably stronger than previous ones. Traditionally, the model of computation includes the parties running the protocol and an adversary, $\mathcal{A}$, and "emulating an ideal process" means that for any adversary $\mathcal{A}$ there should exist an "ideal process adversary" (or, simulator) $\mathcal{S}$ that results in similar distribution on the outputs for the parties. Here an additional adversarial entity, called the environment $\mathcal{Z}$, is introduced. The environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. (Allowing the environment to freely interact with the adversary is crucial for the composability properties ot he definition.) A protocol is said to securely realize a given ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists an "ideal-process adversary" $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that interact with $\mathcal{F}$ in the ideal process. (In a sense, here $\mathcal{Z}$ serves as an "interactive distinguisher" between a run of the protocol and the ideal process with access to $\mathcal{F}$. See [C01] for more motivating discussion on the role of the environment.) Note that the definition requires the "ideal-process adversary" (or, simulator) $\mathcal{S}$ to interact with $\mathcal{Z}$ throughout the computation. Furthermore, $\mathcal{Z}$ cannot be "rewound".

The following *universal composition* theorem is proven in [C01]. Consider a protocol $\pi$ that operates in a *hybrid* model of computation where parties can communicate as usual, and in addition have ideal access to (an unbounded number of copies of) some ideal functionality $\mathcal{F}$. Let $\rho$ be a protocol that securely realizes $\mathcal{F}$ as sketched above, and let $\pi^\rho$ be the "composed protocol". That is, $\pi^\rho$ is identical to $\pi$ with the exception that each interaction with some copy of $\mathcal{F}$ is replaced with a call to (or an invocation of) an appropriate instance of $\rho$. Similarly, $\rho$-outputs are treated as values provided by the appropriate copy of $\mathcal{F}$. Then, $\pi$ and $\pi^\rho$ have essentially the same input/output behavior. In particular, if $\pi$ securely realizes some ideal functionality $\mathcal{G}$ given ideal access to $\mathcal{F}$ then $\pi^\rho$ securely realizes $\mathcal{G}$ from scratch.

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

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

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

### 1.2. On the Constructions

At a closer look, the requirements from a UC commitment protocol boil down to the following two requirements from the ideal-process adversary (simulator) $\mathcal{S}$. *(a).* When the committer is corrupted (i.e., controlled by the adversary), $\mathcal{S}$ must be able to "extract" the committed value from the commitment. (That is, $\mathcal{S}$ has to come up with a value $x$ such that the committer will almost never be able to successfully decommit to any $x' \neq x$.) This is so since in the ideal process $\mathcal{S}$ has to explicitly provide $\mathcal{F}_{\text{COM}}$ with a committed value. *(b).* When the committer is uncorrupted, $\mathcal{S}$ has to be able to generate a kosher-looking "simulated commitment" $c$ that can be "opened" to any value (which will become known only later). This is so since $\mathcal{S}$ has to provide adversary $\mathcal{A}$ and environment $\mathcal{Z}$ with the

simulated commitment $c$ before the value committed to is known. All this needs to be done *without rewinding the environment $\mathcal{Z}$.* (Note that non-malleability is not explicitly required in this description. It is, however, implied by the above requirements.)

From the above description it may look plausible that no simulator $\mathcal{S}$ exists that meets the above requirements in the plain model. Indeed, we formalize and prove this statement for the case of protocols that involve only a committer and a receiver. (In the case where the committer and the receiver are assisted by third parties, a majority of which is guaranteed to remain uncorrupted, standard techniques for multiparty computation are sufficient for constructing UC commitment protocols. See [C01] for more details.)

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

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

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

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

**Related Work.** Pfitzmann et. al. [PW94**,** PW01] present another definitional framework that allows capturing the security requirements of general reactive tasks, and prove a concurrent composition theorem with respect to their framework. Potentially, our work could be cast in their framework as well; however, the composition theorem provided there is considerably weaker than the one in [C01].

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

## 2. Defining Universally Composable Commitments

Section 2.1 shortly summarizes the relevant parts of the general framework of [C01], including the general framework for defining security and the composition theorem. Section 2.3 defines the CRS model. Section 2.4 defines the ideal commitment functionalities, $\mathcal{F}_{\text{COM}}$ and $\mathcal{F}_{\text{MCOM}}$.

### 2.1. The General Framework

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

**Protocol Syntax.** Following [GMR89, G98], a protocol is represented as a system of interactive Turing machines (ITMs), where each ITM represents the program to be run within a different party. Specifically, the input and output tapes model inputs and outputs that are received from and given to other programs running on the same machine, and the communication tapes model messages sent to and received from the network. Adversarial entities are also modeled as ITMs; we concentrate on a non-uniform complexity model where the adversaries have an arbitrary additional input, or an "advice".

**The Adversarial Model.** [C01] discusses several models of computation. We concentrate on a model where the network is *asynchronous* without guaranteed delivery of messages. The communication is public (i.e., all messages can be seen by the adversary) but ideally authenticated (i.e., messages cannot be modified by the adversary). In addition, parties have unique *identities*.[1] The adversary

[1] Indeed, the communication in realistic networks is typically *unauthenticated,* in the sense that messages may be adversarially modified en-route. In addition, there is no guarantee that identities will be unique. Nonetheless, since authentication and the guarantee of unique identities

is adaptive in corrupting parties, and is active (or, *Byzantine*) in its control over corrupted parties. Any number of parties can be corrupted. Finally, the adversary and environment are restricted to probabilistic polynomial time (or, "feasible") computation.

**Protocol Execution in the Real-life Model.** We sketch the process of executing a given protocol $\pi$ (run by parties $P_1, ..., P_n$) with some adversary $\mathcal{A}$ and an environment machine $\mathcal{Z}$ with input $z$. All parties have a security parameter $k \in \mathbf{N}$ and are polynomial in $k$. The execution consists of a sequence of *activations,* where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$, or some $P_i$) is activated. The activated participant reads information from its input and incoming communication tapes, executes its code, and possibly writes information on its outgoing communication tapes and output tapes. In addition, the environment can write information on the input tapes of the parties, and read their output tapes. The adversary can read messages off the outgoing message tapes of the parties and *deliver* them by copying them to the incoming message tapes of the recipient parties. (It is stressed that only messages that were generated by parties can be delivered. The adversary cannot modify or duplicate messages.) The adversary can also corrupt parties, with the usual consequences that it learns the internal information known to the corrupted party and that from now on it controls that party.

The environment is activated first; once activated, it may write information on the input tape of either one of the parties or of the adversary. That entity is activated once the activation of the environment is complete (i,e, once the environment enters a special waiting state.) If no input tape was written into then the execution halts. Once a party completes its activation the environment is activated again. Whenever the adversary delivers a message to some party $P$ in some activation, then this party is activated next. Once $P$'s activation is complete, the environment is activated again. If in some activation the adversary delivers no messages then the environment is activated as soon as the adversary completes its activation. Notice that this mechanism allows environment and the adversary to exchange information freely using their input and output tapes, between each two activations of some party. The output of the protocol execution is the output of $\mathcal{Z}$. (Without loss of generality $\mathcal{Z}$ outputs a single bit.)

Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \vec{r})$ denote the output of environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and parties running protocol $\pi$ on security parameter $k$, input $z$ and random input $\vec{r} = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1 \dots r_n$ as described above ($z$ and $r_{\mathcal{Z}}$ for $\mathcal{Z}$, $r_{\mathcal{A}}$ for $\mathcal{A}$; $r_i$ for party $P_i$). Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ denote the random variable describing $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \vec{r})$ when $\vec{r}$ is uniformly chosen. Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

can be added independently of the rest of the protocol, we allow ourselves to assume ideally authenticated channels and unique identities. See [C01] for further discussion.

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

The order of events in the ideal process is the same as in the real-life process, with the exception that here, if a dummy party $\tilde{P}$ is activated by an input value coming from the environment then (this value is copied to the outgoing communication tape of $\tilde{P}$ and) the ideal functionality is activated next. Once the ideal functionality completes its activation (having perhaps sent messages to the adversary or dummy parties), $\tilde{P}$ is activated one again. It is stressed that in the ideal process there is no communication among the parties. The only "communication" is in fact idealized transfer of information between the parties and the ideal functionality.

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

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

$\mathcal{Z}$ can provide the process in question with *adaptively chosen* inputs throughout the computation.)

**Definition 6.1.** *Let $\mathcal{X} = \{X(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ and $\mathcal{Y} = \{Y(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ be two distribution ensembles over $\{0,1\}$. We say that $\mathcal{X}$ and $\mathcal{Y}$ are* indistinguishable *(written $\mathcal{X} \stackrel{c}{\approx} \mathcal{Y}$) if for any $c \in \mathbf{N}$ there exists $k_0 \in \mathbf{N}$ such that $|\Pr(X(k,a) = 1) - \Pr(Y(k,a) = 1)| < k^{-c}$ for all $k > k_0$ and all $a$.*

**Definition 6.2** ([C01])**.** *Let $n \in \mathbf{N}$. Let $\mathcal{F}$ be an ideal functionality and let $\pi$ be an $n$-party protocol. We say that $\pi$ securely realizes $\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \stackrel{c}{\approx} \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$.*

### 2.2. On the Composition Theorem

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

The communication between the parties and each one of the copies of $\mathcal{F}$ mimics the ideal process. That is, once a party sends a message to some copy of $\mathcal{F}$, that copy is immediately activated and reads that message off the party's tape. Furthermore, although the adversary in the hybrid model is responsible for delivering the messages from the copies of $\mathcal{F}$ to the parties, it does not have access to the contents of these messages. It is stressed that the environment does not have direct access to the copies of $\mathcal{F}$. (Indeed, here the security definition will require that the environment will be unable to tell whether it is interacting with the real-life model or the hybrid model.)

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

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

**Theorem 6.3.** *Let $\mathcal{F}$ be an ideal functionality. Let $\pi$ be a protocol in the $\mathcal{F}$-hybrid model, and let $\rho$ be a protocol that securely realizes $\mathcal{F}$. Then for any real-life adversary $\mathcal{A}$ there exists a hybrid-model adversary $\mathcal{H}$ such that for any environment machine $\mathcal{Z}$ we have* $\text{REAL}_{\pi^\rho,\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},\mathcal{Z}}$.

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

**Theorem 6.4** ([C01])**.** *Let $\mathcal{F}, \mathcal{G}$ be ideal functionalities. Let $\pi$ be an $n$-party protocol that realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model and let $\rho$ be an $n$-party protocol that securely realizes $\mathcal{F}$. Then protocol $\pi^\rho$ securely realizes $\mathcal{G}$.*

### 2.3. The Common Reference String (CRS) Model

In the common reference string (CRS) model it is assumed that all the participants have access to a common string that is drawn from some specified distribution. (This string is chosen ahead of time and is made available before any interaction starts.) In the present framework we re-cast the CRS model framework as a hybrid model with ideal access to a functionality $\mathcal{F}_{\text{CRS}}$, that is parameterized by a distribution $D$ and described in Figure 1 below.

FIGURE 1. The Common Reference String functionality

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

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

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

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

- In the real-life model of computation the parties have access to a common and public string that is chosen in advance according to some distribution (specified by the protocol run by the parties).

- In the ideal process for some functionality (say, for $\mathcal{F}_{\text{COM}}$ defined below) there is no use of the random string. Consequently an ideal process adversary that operates by simulating a real-life adversary may play the role of $\mathcal{F}_{\text{CRS}}$ for the simulated adversary. This means that the ideal process adversary may choose the common string in any way it wishes.

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

**Protocol Composition in the CRS Model.** Some words of clarification are in order with respect to the composition theorem in the CRS model. It is stressed that each copy of protocol $\rho$ within the composed protocol $\pi^\rho$ should have its own copy of the reference string, i.e. a separate instance of $\mathcal{F}_{\text{CRS}}$, (or equivalently uses a separate portion of a long string). If this is not the case then the theorem no longer holds in general. As seen below, the security requirements from protocols where several copies of the protocol use the same instance of the reference string can be captured using ideal functionalities that represent multiple copies of the protocol within a single copy of the functionality.

### 2.4. The Commitment Functionalities

We propose ideal functionalities that represent the intuitive "envelope-like" properties of commitment, as sketched in the introduction. Two functionalities are presented: functionality $\mathcal{F}_{\text{COM}}$ that handles a single commitment-decommitment process, and functionality $\mathcal{F}_{\text{MCOM}}$ that handles multiple such processes. Recall that the advantage of $\mathcal{F}_{\text{MCOM}}$ over $\mathcal{F}_{\text{COM}}$ is that protocols that securely realize $\mathcal{F}_{\text{MCOM}}$ may use the same short common string for multiple commitments. (In contrast, applying the composition theorem to protocols that realize $\mathcal{F}_{\text{COM}}$ requires using a different common string for each commitment.) Indeed, realizing $\mathcal{F}_{\text{MCOM}}$ is more challenging than realizing $\mathcal{F}_{\text{COM}}$. Some further discussion on the functionalities and possible variants appears at the end of this section.

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

Functionality $\mathcal{F}_{\text{COM}}$, described in Figure 2, proceeds as follows. The commitment phase is modeled by having $\mathcal{F}_{\text{COM}}$ receive a value $(\texttt{Commit}, sid, P_i, P_j, b)$, from some party $P_i$ (the committer). Here $sid$ is a Session ID used to distinguish among various copies of $\mathcal{F}_{\text{COM}}$, $P_j$ is the identity of another party (the receiver), and $b \in \{0,1\}$ is the value committed to. In response, $\mathcal{F}_{\text{COM}}$ lets the receiver

FIGURE 2. The Ideal Commitment functionality for a single commitment

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

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

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

*$P_j$ and the adversary $\mathcal{S}$* know that $P_i$ has committed to some value, and that this value is associated with session ID *sid*. This is done by sending the message $(\texttt{Receipt}, sid, P_i, P_j)$ to $P_j$ and $\mathcal{S}$. The opening phase is initiated by the committer sending a value $(\texttt{Open}, sid, P_i, P_j)$ to $\mathcal{F}_{\text{COM}}$. In response, $\mathcal{F}_{\text{COM}}$ hands the value $(\texttt{Open}, sid, P_i, P_j, b)$ to $P_j$ and $\mathcal{S}$.

Functionality $\mathcal{F}_{\text{MCOM}}$, presented in Figure 3, essentially mimics the operation of $\mathcal{F}_{\text{COM}}$ for multiple commitments. In addition to the session ID *sid*, functionality $\mathcal{F}_{\text{MCOM}}$ uses an additional identifier, a Commitment ID *cid*, that is used to distinguish among the different commitments that take place within a single run of $\mathcal{F}_{\text{MCOM}}$. The record for a committed value now includes the Commitment ID, plus the identities of the committer and receiver. To avoid ambiguities, no two commitments with the same committer and verifier are allowed to have the same Commitment ID. It is stressed that the various `Commit` and `Open` requests may be interleaved in an arbitrary way. Also, note that $\mathcal{F}_{\text{MCOM}}$ allows a committer to open a commitment several times (to the same receiver).

FIGURE 3. The Ideal Commitment functionality for multiple commitments

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

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

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

**Definition 6.5.** *A protocol is a universally composable (UC) commitment protocol if it securely realizes functionality $\mathcal{F}_{\text{COM}}$. If the protocol securely realizes $\mathcal{F}_{\text{MCOM}}$ then it is called a reusable-CRS UC commitment protocol.*

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

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

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

1. The functionalities can be modified so that the adversary does not receive the opened value $x$. This captures the concern that the opening of the commitment should be available only to the receiver.

2. The functionalities can be modified so that the receiver of the commitment provides the functionality with acknowledgments for obtaining the commitment and the opening, and the functionality forwards these acknowledgments to the committer. This may be useful in cases where the committer has to make sure that the receiver accepted the commitment and/or the opening.

3. The functionalities can be modified so that the adversary receives no messages whatsoever. This captures the concern that the adversary does not learn whether a commitment protocol took place at all. (This requirement has a flavor of protection against traffic analysis.)

4. Functionalities $\mathcal{F}_{\text{COM}}$ and $\mathcal{F}_{\text{MCOM}}$ don't specify an "error message," to be generated by the receiver, in case where the committer provides the receiver with an invalid opening of some committed value. (Instead, the current specification instructs the receiver to ignore invalid decommitments.) An alternative formulation would instruct the functionality to notify the receiver when it receives an invalid `(Open,...)` message from the committer.

## 3. Impossibility of UC Commitments in the Plain Model

This section demonstrates that in the plain model (i.e., without access to some ideal functionality) there cannot exist universally composable commitment protocols that do not involve third parties in the interaction and allow for successful completion when both the sender and the receiver are honest. This impossibility result holds even under the more liberal requirement that for any real-life adversary and any environment there should be an ideal-model adversary (i.e., under a relaxed definition where the ideal-model simulator may depend on the environment).

We remark that universally composable commitment protocols exist in the plain model if the protocol makes use of third parties (namely, servers), as long as a majority of the servers remain uncorrupted. This follows from a general result in [C01], where it is shown that practically any functionality can be realized in this setting.

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

**Theorem 6.6.** *There exist no bilateral, terminating protocol $\pi$ that securely realizes functionality $\mathcal{F}_{\text{COM}}$ in the plain model. This holds even if the ideal-model adversary $\mathcal{S}$ is allowed to depend on the environment $\mathcal{Z}$.*

**Proof.** The idea of the proof is as follows. Consider a protocol execution between an adversarially controlled committer $P_i$ and an honest receiver $P_j$, and assume that the adversary merely sends messages that are generated by the environment, and relays to the environment the messages sent to $P_i$. The environment secretly picks a random bit $b$ at the beginning and generates the messages for $P_i$ by running the protocol of the honest committer for $b$ and $P_j$'s answers. In order to simulate this behavior, the ideal-model adversary $\mathcal{S}$ must be able to provide the ideal functionality with a value for the committed bit. In other words, the simulator has

to "extract" the committed bit from the messages generated by the environment, without the ability to rewind the environment. However, as will be seen below, if the commitment scheme allows the simulator to successfully extract the committed bit, then the commitment is not secure in the first place (in the sense that a corrupted receiver can obtain the value of the committed bit from interacting with an honest committer).

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

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

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

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

By the termination property, we obtain from the virtual simulator $\mathcal{S}$ a bit $b'$ with non-negligible probability. This bit is a good approximation of the actual bit $b$, since $\mathcal{S}$ simulates the real protocol $\pi$ except with negligible error. Hence, the guess of $\mathcal{A}'$ for $b$ is correct with $1/2$ plus a non-negligible probability. But for a putative ideal-model adversary $\mathcal{S}'$ predicting this bit $b$ with more than non-negligible probability over $1/2$ is impossible, since the view of $\mathcal{S}'$ in the ideal

process is statistically independent from the bit $b$. (Recall that the commitment to $b$ is never opened). □

## 4. UC Commitment Schemes in the CRS Model

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

### 4.1. One-Time Common Reference String

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

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

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

**Description of Commitment Scheme.** We describe a UC bit commitment protocol $\mathsf{UCC}_{\mathsf{OneTime}}$ (for universally composable commitment scheme in the one-time-usable common reference string model). The idea is to use the [DIO98] scheme with a special pseudorandom generator that has a trapdoor property. Specifically, we use the Blum-Micali-Yao generator but with trapdoor permutations instead

ofone-way permutations [Y82, BM84]. Let $\mathsf{KGen}$ denote an efficient algorithm that on input $1^n$ generates a random public key $pk$ and the trapdoor $td$. The key $pk$ describes a trapdoor permutation $f_{pk}$ over $\{0,1\}^n$. Let $B(\cdot)$ be a hard core predicate for $f_{pk}$. Define a pseudorandom generator expanding $n$ bits to $4n$ bits with public description $pk$ by

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

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

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

FIGURE 4. Commitment Scheme in the One-Time-Usable Common Reference String Model

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

*public string:*

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

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

compute $G_{pk_b}(r)$ for random $r \in \{0,1\}^n$
set $y = G_{pk_b}(r)$ for $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$
send $(\mathtt{Com}, sid, y)$ to the receiver
Upon receiving $(\mathtt{Com}, sid, y)$ from $P_i$,
$P_j$ outputs $(\mathtt{Receipt}, sid, cid, P_i, P_j)$

*decommitment for $y$:*

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

In order to commit to a bit $b \in \{0,1\}$, the sender picks a random string $r \in \{0,1\}^n$, computes $G_{pk_b}(r)$, and sets $y = G_{pk_b}(r)$ if $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$

for $b = 1$. The sender passes $y$ to the receiver. In the decommitment step the sender gives $(b, r)$ to the receiver, who verifies that $y{=}G_{pk_b}(r)$ for $b = 0$ or that $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$. See also Figure 4.

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

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

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

**Theorem 6.7.** *Protocol* $\mathsf{UCC}_{\mathsf{OneTime}}$ *securely realizes functionality* $\mathcal{F}_{\text{COM}}$ *in the* CRS *model.*

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

1. At the outset the simulator $\mathcal{S}$ prepares $\sigma$ by selecting key pairs $(pk_0, td_0) \leftarrow \mathsf{KGen}(1^n)$ and $(pk_1, td_1) \leftarrow \mathsf{KGen}(1^n)$ and setting $\sigma = G_{pk_0}(r_0) \oplus G_{pk_1}(r_1)$ for random $r_0, r_1 \in \{0,1\}^n$. We call this a *fake* string $\sigma$ with respect to preselected values $pk_0$, $pk_1$, $G_{pk_0}(r_0)$ and $G_{pk_1}(r_1)$. Next, $\mathcal{S}$ starts the simulation of $\mathcal{A}$ and the execution with $\mathcal{Z}$ on the fake string $\sigma$ and $pk_0, pk_1$.
2. If at some point in the execution the environment $\mathcal{Z}$ writes a message $(\texttt{Commit}, sid, P_i, P_j, b)$ on the tape of the *uncorrupted* party $\tilde{P}_i$, and $\tilde{P}_i$ copies this to the functionality $\mathcal{F}_{\text{COM}}$, then the ideal-model simulator — who cannot read the actual bit, but is informed about the commitment by receiving $(\texttt{Receipt}, sid, P_i, P_j)$— tells $\mathcal{A}$ that $P_i$ has sent $y = G_{pk_0}(r_0)$ to $P_j$.

3. If at some point in the execution $\mathcal{Z}$ instructs an *uncorrupted* party $\tilde{P}_i$ to decommit and this party has previously correctly committed to some secret bit $b$. Then the ideal-model adversary $\mathcal{S}$ must have sent the value $y = G_{pk_0}(r_0)$ on behalf of $P_i$ in the black-box simulation of $\mathcal{A}$. In the ideal model, $\mathcal{S}$ now learns $b$ from $\tilde{P}_i$ via $\mathcal{F}_{\text{COM}}$ and opens $y$ in the simulation of $\mathcal{A}$ accordingly, using the equivocability property.
4. If the simulated adversary $\mathcal{A}$ lets some *corrupted* party $P_i$ send `(Com,`$sid$`,`$y^*$`)` to an honest party $P_j$ then $\mathcal{S}$ verifies with the help of the trapdoor $td_0$ whether $y^*$ is in the range of $G_{pk_0}(\cdot)$ or not. If so, $\mathcal{S}$ sends a message $(\texttt{Commit}, sid, P_i, P_j, 0)$ on behalf of the party to the functionality; else $\mathcal{S}$ sends $(\texttt{Commit}, sid, P_i, P_j, 1)$ to $\mathcal{F}_{\text{COM}}$.
5. If $\mathcal{A}$ tells a *corrupted* party $P_i$ to open a valid commitment $y^*$ correctly with bit $b^*$, then $\mathcal{S}$ compares $b^*$ to the previously extracted bit and stops if they differ; otherwise $\mathcal{S}$ sends $(\texttt{Open}, sid, P_i, P_j)$ in the name of the party to $\mathcal{F}_{\text{COM}}$. If $P_i$ is supposed to decommit incorrectly, then $\mathcal{S}$ also sends an incorrect opening to the functionality.
6. Whenever the simulated $\mathcal{A}$ demands to corrupt a party, $\mathcal{S}$ corrupts this party in the ideal model and learns all internal information of the party. Now $\mathcal{S}$ first adapts possible decommitment information about a previously given but yet unopened commitment of this party, like in the case of an honest party decommitting. After this, $\mathcal{S}$ gives all this adjusted information to $\mathcal{A}$.

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

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

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

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

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

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

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

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

- generation of public string:
  - pick a bit $c$ at random and set $pk_{1-c} = pk$ for the given public key (the bit $c$ is our guess for the bit of an honest party committing)
  - generate another key pair $(pk_c, td_c) \leftarrow \mathsf{KGen}(1^n)$
  - select $r_c \in \{0,1\}^n$ at random and set $\sigma = G_{pk_c}(r_c) \oplus z$
- emulation:
  - simulate the real-life protocol with $\mathcal{A}, \mathcal{Z}$ on $\sigma, pk_0, pk_1$
  - if an uncorrupted party $P_i$ is told by $\mathcal{Z}$ to commit to a bit $b$, then we stop immediately with output 0 if $b \neq c$ (i.e., our guess is wrong). In the case $b = c$ we send $G_{pk_c}(r_c)$ for $b = c = 0$ and $z$ for $b = c = 1$ in the name of $P_i$ and continue the simulation; when $\mathcal{Z}$ later instructs $P_i$ to decommit, we transmit $b(= c)$ and $r_c$. Analogously, we present $b, r_c$ to $\mathcal{A}$ if this party is corrupted before decommitting.
  - if the adversary $\mathcal{A}$ corrupts the sender $P_i$ before this party is giving the commitment, then we stop with probability 1/2 (this provides symmetry to the first case and simplifies the analysis); otherwise we go on with the real-life simulation.
- output:
  - given that we have not stopped yet, simply copy $\mathcal{Z}$'s output.

To analyze the advantage of our algorithm we start with the case that $z$ is a uniformly distributed $4n$-bit string. Then $\sigma$ is also random and our prediction $c$ is hidden information-theoretically from $\mathcal{A}$ and $\mathcal{Z}$ at the outset of the execution. Therefore, the probability that we stop prematurely with output 0 is 1/2, independent of the fact whether $\mathcal{A}$ plays the committer or lets an honest party

commit. Conditioning on that we enter the final output step, it is easy to see that $\mathcal{Z}$'s output is identically distributed to a sample of Real/Genuine.

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

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

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

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

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

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

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

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

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

But for random $z$ the probability that

$$z \in \left\{ G_{pk_0}(r_0) \oplus G_{pk_0}(r_0^*) \oplus G_{pk_1}(r_1^*) \;\middle|\; r_0, r_0^*, r_1^* \in \{0,1\}^n \right\}$$

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

### 4.2. Reusable Common Reference String: Erasing Parties

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

**Preliminaries.** Basically, a claw-free trapdoor permutation pair is a pair of trapdoor permutations with a common range such that it is hard to find two elements that are preimages of the same element under the two permutations. More formally, a key generation $\mathsf{KGen}_{\mathsf{claw}}$ outputs a random public key $pk_{\mathsf{claw}}$ and a trapdoor $td_{\mathsf{claw}}$. The public key defines permutations $f_{0,pk_{\mathsf{claw}}}, f_{1,pk_{\mathsf{claw}}} : \{0,1\}^n \to \{0,1\}^n$, whereas the secret key describes the inverse functions $f^{-1}_{0,pk_{\mathsf{claw}}}, f^{-1}_{1,pk_{\mathsf{claw}}}$. It should be infeasible to find a claw $x_0, x_1$ with $f_{0,pk_{\mathsf{claw}}}(x_0) = f_{1,pk_{\mathsf{claw}}}(x_1)$ given only $pk_{\mathsf{claw}}$. For ease of notation we usually omit the keys and write $f_0, f_1, f_0^{-1}, f_1^{-1}$ instead. Claw-free trapdoor permutation pairs exist for example under the assumption that factoring is hard [GMR88]. For a more formal definition see [G98].

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

**Description of the Commitment Scheme.** The commitment scheme $\mathsf{UCC}_{\mathsf{ReUse}}$ (for universally composable commitment with reusable reference string) is displayed in Figure 5. The (reusable) public string contains random public keys $pk_{\mathsf{claw}}$ and $pk_{\mathcal{E}}$. For a commitment to a bit $b$ the sender $P_i$ obtains a value $y$ by applying the trapdoor permutation $f_b$ to a random $x \in \{0,1\}^n$, computes

FIGURE 5. Commitment Scheme with Reusable Reference String

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

*public string:*

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

*commitment by party $P_i$ to party $P_j$ to $b \in \{0,1\}$ with identifier sid, cid:*

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

*decommitment for $(P_i, P_j, sid, cid, b, x, r_b)$:*

Send $(\mathtt{Dec}, sid, cid, b, x, r_b)$ to $P_j$.
Upon receiving $(\mathtt{Dec}, sid, cid, b, x, r_b)$, $P_j$ verifies that $y \stackrel{?}{=} f_b(x)$,
that $c_b$ is encryption of $(x, P_i)$ under randomness $r_b$
where $P_i$ is the committer's identity
and that *cid* has not been used with this committer before.
If the verification succeeds then $P_j$ outputs $(\mathtt{Open}, sid, cid, P_i, P_j, b)$.

$c_b \leftarrow \mathsf{Enc}_{pk_{\mathcal{E}}}(x, P_i)$ and $c_{1-b} \leftarrow \mathsf{Enc}_{pk_{\mathcal{E}}}(0^n, P_i)$, and sends the tuple $(y, c_0, c_1)$ to the receiver. The sender is also instructed to erase the randomness used for the encryption of $(0^n, P_i)$ before the commitment message is sent. This ciphertext is called a dummy ciphertext.

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

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

The fact that the dummy ciphertext is never opened buys us equivocability. Say that the ideal-model simulator knows the trapdoor of the claw-free permutation pair. Then it can compute the preimages $x_0, x_1$ of some $y$ under both functions $f_0, f_1$ and send $y$ as well as encryptions of $(x_0, P_i)$ and $(x_1, P_i)$. To open it as 0 hand $0, x_0$ and the randomness for ciphertext $(x_0, P_i)$ to the receiver and claim to have erased the randomness for the other encryption. For a 1-decommitment send $1, x_1$, the randomness for the encryption of $(x_1, P_i)$ and deny to know the randomness for the other ciphertext. If the encryption scheme is secure then it is intractable to distinguish dummy encryptions from fake ones. Hence, this procedure is indistinguishable from the actual steps of the honest parties.

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

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

**Theorem 6.8.** *Protocol* $\mathsf{UCC}_{\mathsf{ReUse}}$ *securely realizes functionality* $\mathcal{F}_{\text{MCOM}}$ *in the* CRS *model.*

**Proof.** As in the proof of Theorem 6.7 we present an ideal-process adversary $\mathcal{S}$ simulating a virtual copy of the real-life adversary $\mathcal{A}$ and relaying messages of $\mathcal{A}$ and the environment $\mathcal{Z}$. The ideal-process adversary is defined by the following actions:

1. the simulator $\mathcal{S}$ chooses keys $(pk_{\mathsf{claw}}, td_{\mathsf{claw}}) \leftarrow \mathsf{KGen}_{\mathsf{claw}}(1^n)$ and $(pk_{\mathcal{E}}, sk_{\mathcal{E}}) \leftarrow \mathsf{KGen}_{\mathcal{E}}(1^n)$, defines the public string to be the pair $pk_{\mathsf{claw}}, pk_{\mathcal{E}}$, and simulates an execution of $\mathcal{A}$ with $\mathcal{Z}$ on $pk_{\mathsf{claw}}, pk_{\mathcal{E}}$.
2. If during this execution the environment $\mathcal{Z}$ lets an *uncorrupted* party $\tilde{P}_i$ send a message $(\texttt{Commit}, sid, cid, P_i, P_j, b)$ to the functionality then the ideal-model simulator is informed about the commitment but not the bit itself. The simulator picks a random $x_0 \in \{0,1\}^n$, computes $y = f_0(x_0)$ and $x_1 = f_1^{-1}(y)$ as well as encryptions $c_0 \leftarrow \mathsf{Enc}(x_0, P_i)$ and $c_1 \leftarrow \mathsf{Enc}(x_1, P_i)$. Tell $\mathcal{A}$ that party $P_i$ has sent $sid, cid, (y, c_0, c_1)$.
3. If an *uncorrupted* party $\tilde{P}_i$ is instructed by $\mathcal{Z}$ to open a commitment to some bit $b$, then the ideal-model adversary learns $b$ from $\mathcal{F}_{\text{MCOM}}$. Pretend in the simulation of $\mathcal{A}$ that the previously sent $(y, c_0, c_1)$ equals a $b$-commitment by sending $b, x_b$ and the randomness to encrypt $c_b$; claim that the randomness for the other encryption has been deleted.

4. If the simulated $\mathcal{A}$ lets some *corrupted* party $P_i$ commit to an honest party $P_j$ by sending $(\texttt{Com}, sid^*, cid^*, (y^*, c_0^*, c_1^*))$, then $\mathcal{S}$ decrypts $c_0^*$ with $sk_{\mathcal{E}}$ to $(x^*, P_i^*)$ and checks whether $P_i^* = P_i$ and if $cid^*$ has not been used in a commitment of $P_i$ to $P_j$ before; if either condition is violated then ignore this message. Else, $\mathcal{S}$ sends a message $(\texttt{Commit}, sid^*, cid^*, P_i, P_j, b)$ on behalf of $P_i$ to the functionality, where the bit $b$ is determined as follows. If $c_0^*$ was previously used in a (simulated) commitment $(y, c_0^*, c_1)$ or $(y, c_0, c_0^*)$ of $P_i$ when $P_i$ was still uncorrupted, then the bit $b$ is set to the bit that this previous commitment was opened to (either by an instruction of $\mathcal{Z}$ or upon corruption of $P_i$); otherwise, if $f_0(x^*) = y^*$ then $b = 0$, else $b = 1$.
5. If $\mathcal{A}$ tells a *corrupted* party $P_i$ to open a commitment $(\texttt{Com}, sid^*, cid^*, (y^*, c_0^*, c_1^*))$ correctly with bit $b^*$, then $\mathcal{S}$ compares $b^*$ to the previously extracted bit for these IDs and aborts if the bits are different; in case of equality $\mathcal{S}$ sends $(\texttt{Open}, sid, cid, P_i, P_j)$ in the name of the party to $\mathcal{F}_{\text{MCOM}}$. If $\mathcal{A}$ lets $P_i$ give an incorrect opening, then $\mathcal{S}$ can ignore this message because the functionality does not open it.
6. Assume that $\mathcal{A}$ demands to corrupt a party in the black-box simulation. Then $\mathcal{S}$ gets all internal information from this party by corrupting it in the ideal model. $\mathcal{S}$ modifies all decommitment information about unopened commitments of this party to match the received data and hands this modified internal information to $\mathcal{A}$.

The proof that the $\mathcal{Z}$'s output in the real-life is indistinguishable from its output in the ideal process is in the line of the proof for Theorem 6.7. We investigate again three hybrid variables:

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

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

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

Suppose that the extreme hybrids Real/Genuine and Ideal/Fake are distinguishable. This means either that the hybrids Real/Genuine and Real/Fake iare distinguishable or that the hybrids Real/Fake and Ideal/Fake are distinguishable. We

will show that this leads to a contradiction to the claw-freeness or to the chosen ciphertext security of the encryption scheme.

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

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

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

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

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

4. if the honest party is asked to decommit (or, similarly, is corrupted before decommitting) then $\mathcal{D}_{CPA}$ presents $b, x_b$ and the randomness for producing $c_b$.
5. at the end, copy $\mathcal{Z}$'s output

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

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

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

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

We would like to demonstrate that last statement via reduction to the security of the claw-free pair $(f_0, f_1)$. However, a direct reduction does not seem to work. We thus first show that if event $\mathcal{B}$ occurs in Ideal/Fake with non-negligible probability, then this should also be true if we replace the simulated commitments of honest parties in $\mathcal{A}$'s simulation with commitments where we correctly put a dummy ciphertext into the tuple instead of an encryption of $(x_{1-b}, P_i)$. Call this

new experiment Ideal/Genuine. That is, experiment Ideal/Genuine is identical to experiment Ideal/Fake with the exception that in Ideal/Genuine the simulator $\mathcal{S}$ 'magically knows' the real values committed to by the uncorrupted parties, and generates genuine commitments for these values. We show that if the probability of event $\mathcal{B}$ in the two experiments differs by non-negligible amount then it is possible to break the CCA security of the encryption scheme $\mathcal{E}$.

CLAIM 3: The probability of event $\mathcal{B}$ in experiment Ideal/Fake differs from the probability of event $\mathcal{B}$ in experiment Ideal/Genuine by at most a negligible amount.

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

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

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

(a) If $c_0^*$ has not been returned from the left-or-right encryption oracle of $\mathcal{D}_{CCA}$ before, then $\mathcal{D}_{CCA}$ asks its decryption oracle to decrypt $c_0^*$ and proceeds like the ideal-model adversary $\mathcal{S}$.
(b) Otherwise, if $c_0^*$ has been returned from the left-or-right encryption oracle, then $\mathcal{D}_{CCA}$ has sent this value in a commitment $(y, c_0^*, c_1)$ or $(y, c_0, c_0^*)$ in the name of some honest party. If this has been a different party than $P_i$ then ignore the adversary's message. Else ($c_0^*$ appeared in a commitment of $P_i$ before $P_i$ was corrupted), recall the corresponding bit from the previous commitment and proceed like the ideal-model adversary $\mathcal{S}$.

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

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

The analysis of $\mathcal{D}_{CCA}$ is almost identical to the case of distinguisher $\mathcal{D}_{CPA}$ above and is omitted. This completes the proof of Claim 3. $\diamond$

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

**Relaxing the Need for Claw-free Pairs.** The above scheme was presented and proven using any claw-free pair of trapdoor permutations. However, it is easy to see that the claw-free pair can be substituted by chameleon (aka. trapdoor) commitments a la [BCC88]. That is, any non-interactive perfectly-secret trapdoor commitment works. Such commitments exist for instance under the hardness of the discrete logarithm or factoring problem. Further relaxing the underlying hardness assumptions is an interesting open problem..

### 4.3. Reusable Common Reference String: Non-Erasing Parties

A careful look at the proof of Theorem 6.8 shows that, instead of letting the sender generate a ciphertext and erase the randomness, it is sufficient to enable the parties to obliviously generate a "ciphertext-like" string without knowing the plaintext, but such that a simulator can produce a correct ciphertext and a fake random string suggesting that the ciphertext has been obtained by the oblivious sampling procedure. Then the honest parties can use the sampling mechanism to produce the dummy ciphertext, while the simulator is still able to place the fake encryption into the commitment and to find fake randomness making it look like a dummy ciphertext. We show how this can be done under certain conditions, and show that these conditions can be met if the encryption scheme in use is that of [CS98].

**Preliminaries: Obliviously Samplable Encryption Scheme.** We formalize the requirement for the oblivious sampling procedure of the encryption scheme in the following definition:

**Definition 6.9.** *A public-key encryption scheme* $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is obliviously samplable with respect to chosen-plaintext attacks if there are probabilistic polynomial-time algorithms* $\mathsf{sample}, \mathsf{fake}$ *such that for any probabilistic polynomial-time algorithm* $\mathcal{A}$ *the probability that* $Experiment_{\mathcal{A}}(1^n) = 1$ *is negligibly close to* $1/2$*, where*

*$Experiment_{\mathcal{A}}(1^n)$:*

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

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

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

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

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

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

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

The oblivious element generation for $G$ is invertible in the sense that, given a random group element $h \in G$ we can efficiently generate a random element $h_p$ in $\mathbb{Z}_p^*$ (and a corresponding bit string of length $2|p|$) mapping to $h$ if raised to the $w$-th power. Namely, let $g$ be a generator of $\mathbb{Z}_p^*$. Solve the equation $xw = 1 \bmod q$ for $x$, pick a random integer $i$ between 0 and $w-1$ and define the element $h_p := h^x g^{iq} \bmod p$. Since the $g^{iq}$'s are $w$-th roots of unity, it is readily verified that indeed $h_p^w = h \bmod p$ and that $h_p$ is uniformly distributed among the preimages of $h$ under exponentiation with $w$. Adding for random $j$ between 0 and $p-1$ the value $jp$ to $h_p$ over the integers gives a $2|p|$-bit string whose distribution is statistically close to the uniform distribution on bit strings mapping to $h$ with the oblivious element generation.

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

The fact that the outputs of sample and fake are indistinguishable under the Decisional Diffie-Hellman assumption follows from the proof in [CS98]. This is

true even if the adversary has access to the decryption oracle. Altogether, the Cramer-Shoup scheme is obliviously samplable with respect to chosen-ciphertext attacks.

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

**Theorem 6.10.** *Protocol* $\mathsf{UCC}_{\mathsf{ReUse/NotErase}}$ *securely realizes functionality* $\mathcal{F}_{\text{MCOM}}$ *in the* CRS *model.*

**Proof.** The proof of the theorem is almost identical to the one in the case of erasing parties. Only this time the ideal-model simulator works slightly different when an uncorrupted party commits or is corrupted or decommits. Namely, for a commitment the simulator in Theorem 6.8 sends encryptions of $(x_0, P_i)$ and $(x_1, P_i)$ in the name of this party; after having learned the actual bit $b$ in case of corruption or decommitment, the simulator there then claims to have erased the randomness for the wrong value $x_{1-b}$. In our case, the simulator also encrypts both values in the commitment phase, but in the reveal step it invokes algorithm $\mathsf{fake}$ on public key $pk_{\mathcal{E}}$ and the ciphertext for the wrong value $x_{1-b}$ to produce a fake random string. Besides the true randomness used to produce the encryption of $x_b$, the simulator hands the fake randomness to the adversary in order to prove that the ciphertext for $x_{1-b}$ has been sampled obliviously.

In the proof of Theorem 6.8, the indistinguishability of the simulator's way to commit and decommit on behalf of honest parties and the behavior of the actual sender relies on the indistinguishability of fake and dummy encryptions. Specifically, we have reduced indistinguishability of simulations twice to the left-or-right security of the encryption system, one time in a chosen-plaintext attack and the other time in a chosen-ciphertext attack. In these reductions the left-or-right oracle encrypts either all left messages $(0^n, P_i)$ or all right messages $(x_{1-b}, P_i)$. Which messages are encrypted, the left or right ones, corresponds to the behavior of honest parties or the simulator.

Except for the reductions to left-or-right security the proof of Theorem 6.8 remains unchanged. In particular, the behavior of $\mathcal{S}$ in case that the committer is corrupted remains unchanged. The simulation remains valid since the encryption scheme remains uniquely decipherable.

To adapt the proof to the simulation here it is sufficient to extend the notion of left-or-right security to obliviously samplable encryption schemes. Namely, the

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

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

Here, instead of passing $(0^n, P_i)$ and $(x_{1-b}, P_i)$ to the left-or-right oracle, we forward the message $(x_{1-b}, P_i)$ to the sample-or-encrypt oracle to obtain either an oblivious sample $c_{\mathsf{sample}}$ and the randomness $r_{\mathsf{sample}}$, or a ciphertext of the message $(x_{1-b}, P_i)$ together with a fake random string. Denote the answer by $(c_{1-b}, r_{1-b})$ and let the simulator transmit $c_{1-b}$ as part of the commitment. Later, in the decommitment phase or upon corruption, the simulator reveals $r_{1-b}$ on behalf of the sender. As the choice of the sample-or-encrypt oracle determines whether we simulate honest parties (if the oracle returns oblivious samples) or the simulator (if the oracle produces correct ciphertexts and fake randomness), it is easy to see that the proof of Theorem 6.8 carries over to this case. □

## 5. Application to Zero-Knowledge

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

Functionality $\mathcal{F}_{\text{ZK}}$, described in Figure 6, is parameterized by a binary relation $R(x, w)$.. It first waits to receive a message $(\texttt{verifier}, id, P_i, P_j, x)$ from some party $P_i$, interpreted as saying that $P_i$ wants $P_j$ to prove to $P_i$ that it knows a value

$w$ such that $R(x, w)$ holds. Next, $\mathcal{F}_{\text{ZK}}$ waits for $P_j$ to explicitly provide a value $w$, and notifies $P_i$ whether $R(x, w)$ holds. (Notice that the adversary is notified whenever either the prover or the verifier starts an interaction. It is also notified whether the verifier accepts. This represents the fact that ZK is not traditionally meant to hide this information.)

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

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

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

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

We demonstrate a protocol for securely realizing $\mathcal{F}_{\text{ZK}}^R$ with respect to any NP relation $R$. The protocol is a known one: It consists of $n$ parallel repetitions of the 3-round protocol of Blum for graph Hamiltonicity, where the provers commitments are replaced by invocations of $\mathcal{F}_{\text{COM}}$. The protocol (in the $\mathcal{F}_{\text{COM}}$-hybrid model) is presented in Figure 7.

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

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

**Theorem 6.11.** *Protocol* HC *securely realizes $\mathcal{F}_{\text{ZK}}^H$ in the $\mathcal{F}_{\text{COM}}$-hybrid model.*

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

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

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

**Protocol Hamilton-Cycle (HC)**

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

1. If $\mathcal{A}$, controlling a corrupted party $P$, starts an interaction as a prover with an uncorrupted party $V$, then $\mathcal{S}$ records the values that $\mathcal{A}$ sends to $\mathcal{F}_{\text{COM}}$, plays the role of $V$ (i.e., $\mathcal{S}$ provides $\mathcal{A}$ with a random set of bits $c_1, ..., c_n$), and records $\mathcal{A}$'s responses. Now $\mathcal{S}$ simulates $V$'s decision algorithm and if $V$ accepts then $\mathcal{S}$ finds a Hamiltonian cycle $h$ in $G$ and hands $g$ to $\mathcal{F}_{\text{ZK}}^{H}$. Else $\mathcal{S}$ hands an invalid cycle $h'$ in $G$ (say, the all-zero cycle) to $\mathcal{F}_{\text{ZK}}^{H}$. It remains to describe how $\mathcal{S}$ finds a Hamiltonian cycle $h$ in

$G$. This is done as follows: $\mathcal{S}$ looks for a $k$ such that $c_k = 1$ and the cycle $h$ decommitted to by $\mathcal{A}$, combined with the committed permutation $\pi_k$, point to a Hamiltonian cycle in $G$. If such a $k$ is not found then $\mathcal{S}$ aborts; but, as claimed below, this will occur only with probability $2^{-n/2}$.

2. If an uncorrupted party $P$ starts an interaction with a corrupted party $V$ then $\mathcal{S}$ learns from $\mathcal{F}_{\text{ZK}}^{H}$ whether $V$ should accept or reject, and simulates the view of $\mathcal{A}$ accordingly. Notice that $\mathcal{S}$ has no problem carrying out the simulation since it simulates for $\mathcal{A}$ an interaction with $\mathcal{F}_{\text{COM}}$ where $\mathcal{F}_{\text{COM}}$ is played by $\mathcal{S}$ himself. Thus, $\mathcal{S}$ is not bound by the "commitments" and can "open" them in whichever way it pleases.
3. If two uncorrupted parties $P$ and $V$ interact then $\mathcal{S}$ simulates for $\mathcal{A}$ the appropriate protocol messages. This case is very similar to the case of corrupted verifier, since this is an Arthur-Merlin protocol.
4. Party corruptions are dealt with in a straightforward way. Corrupting the verifier provides the adversary with no extra information (again, since the protocol is Arthur-Merlin). When the prover is corrupted $\mathcal{S}$ corrupts the prover in the ideal process, obtains $w$, and generates an internal state of the prover that matches the protocol stage and whether $R(x, w)$ holds. Generating such a state is not problematic since $\mathcal{S}$ is not bound by any "commitments", and it can freely choose $\pi_1, ...\pi_k$ to match the (simulated) conversation up to the point of corruption.

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

**Remark.** Notice that Theorem 6.11 holds even if the environment and the real-life adversary are allowed to be *computationally unbounded.* In this case, the complexity of $\mathcal{S}$ is polynomial in the complexity of $\mathcal{A}$ (and is independent of the complexity of $\mathcal{Z}$). This means that the only place where cryptographic assumptions are needed is in realizing $\mathcal{F}_{\text{COM}}$.

# Bibliography


[AFK89] M. Abadi, J. Feigenbaum, J. Kilian: On Hiding Information from an Oracle, *Journal of Computer and System Sciences, Vol. 39, pp. 21–50*, 1989.

[BDG95] J. Balcázar, J. Díaz, J. Gabarró: Structural Complexity I (Second Edition), *Springer-Verlag*, 1995.

[B01] B. Barak: How to Go Beyond the Black-Box Simulation Barrier, *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press*, 2001.

[B91] D. Beaver: Secure Multi-Party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority, *Journal of Cryptology, Vol. 4, pp. 75–122, Springer-Verlag*, 1991.

[B96] D. Beaver: Adaptive Zero-Knowledge and Computational Equivocation, *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC), ACM Press*, 1996.

[BBM00] M. Bellare, A. Boldyreva, S. Micali: Public-Key Encryption in a Multi-User Setting: Security Proofs and Improvements, *Advances in Cryptology — Proceedings of Eurocrypt 2000, Lecture Notes in Computer Science, Vol. 1807, pp. 259–274, Springer-Verlag*, 2000.

[BCK96] M. Bellare, R. Canetti, H. Krawczyk: Pseudorandom Functions Revisited: The Cascade Construction and its Concrete Security, *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press*, 1996.

[BDJR97] M. Bellare, A. Desai, E. Jokipii, P. Rogaway: A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operations, *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 394–403, IEEE Computer Society Press*, 1997.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval, P. Rogaway: Relations Among Notions of Security for Public-Key Encryption Schemes, *Advances in Cryptology — Proceedings of Crypto '98, Lecture Notes in Computer Science, Vol. 1462, pp. 26–40, Springer-Verlag*, 1998.

[BG92] M. Bellare, O. Goldreich: On Defining Proofs of Knowledge, *Advances in Cryptology — Proceedings of Crypto '92, Lecture Notes in Computer Science, Vol. 740, pp. 390–420, Springer-Verlag*, 1992.

[BFGM01] M. Bellare, M. Fischlin, S. Goldwasser, S. Micali: Identification Protocols Secure Against Reset Attacks, *Advances in Cryptology — Proceedings of Eurocrypt 2001, Lecture Notes in Computer Science, Vol. 2045, pp. 495–511, Springer-Verlag*, 2001.

[BGM00] M. Bellare, S. Goldwasser, S. Micali: Identification Protocols Secure Against Reset Attacks, *Cryptology ePrint Archive Report 2000/015, available at* `http://eprint.iacr.org`, 2000.

[BMO90] M. Bellare, S. Micali, R. Ostrovsky: Perfect Zero-Knowledge in Constant Rounds, *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing (STOC), pp. 482–493, ACM Press*, 1990.

[BPR00] M. Bellare, D. Pointcheval, P. Rogaway: Authenticated Key Exchange Secure Against Dictionary Attacks, *Advances in Cryptology — Proceedings of Eurocrypt 2000, Lecture Notes in Computer Science, Vol. 1807, pp. 139–155, Springer-Verlag*, 2000.

[BR94] M. Bellare, P. Rogaway: Optimal Asymmetric Encryption, *Advances in Cryptology — Proceedings of Eurocrypt '94, Lecture Notes in Computer Science, Vol. 950, pp. 92–111, Springer Verlag*, 1994.

[BR93] M. Bellare, P. Rogaway: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols, *ACM Conference on Computer and Communication Security, pp. 62–73, ACM Press*, 1993.

[B82] M. Blum: Coin Flipping by Telephone, *Proceedings of the 24th IEEE Compcon, pp. 133–137, IEEE Computer Society Press*, 1982.

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

[B98] D. Boneh: The Decision Diffie-Hellman Problem, *Third Algorithmic Number Theory Symposium, Lecture Notes in Computer Science, Vol. 1423, pp. 48–63, Springer-Verlag*, 1998.

[B99] D. Boneh: Twenty years of attacks on the RSA cryptosystem, *Notices of the American Mathematical Society (AMS), Vol. 46, No. 2, pp. 203–213*, 1999.

[BV98] D. Boneh, R. Venkatesan: Factoring is not Easier than RSA, *Advances in Cryptology — Proceedings of Eurocrypt '98, Lecture Notes in Computer Science, Vol. 1233, pp. 59–71, Springer-Verlag*, 1998.

[BCY91] G. Brassard, C. Crépeau, M. Yung: Constant-Round Perfect Zero-Knowledge Computationally Convincing Proofs, *Theoretical Computer Science, Vol. 84, No. 1*, 1991.

[BCC88] G. Brassard, D. Chaum, C. Crépeau: Minimum Disclosure Proofs of Knowledge, *Journal of Computer and Systems Science, Vol. 37(2), pp. 156–189*, 1988.

[BM92] E. Brickel, K. McCurley: An Interactive Identification Scheme Based on Discrete Logarithms and Factoring, *Journal of Cryptology, Vol. 5, pp. 29–39, Springer-Verlag*, 1992.

[C00a] R. Canetti: Security and Composition of Multiparty Cryptographic Protocols, *Journal of Cryptology, Vol. 13, No. 1, pp. 143–202, Springer-Verlag*, 2000.

[C01] R. Canetti: Universally Composable Security: A new Paradigm for Cryptographic Protocols, *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press; preliminary version available at:*

*Cryptology ePrint Archive Report 2000/067, available at* `http://eprint.iacr.org`, 2001.

[CF01] R. Canetti, M. Fischlin: Universally Composable Commitments, *Advances in Cryptology — Proceedings of Crypto 2001, Lecture Notes in Computer Science, Vol. 2139, pp. 19–40, Springer-Verlag*, 2001.

[CGGM00] R. Canetti, S. Goldwasser, O. Goldreich, S. Micali: Resettable Zero-Knowledge, *Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing (STOC), ACM Press*, 2000.

[CGH98] R. Canetti, O. Goldreich, S. Halevi: The Random Oracle Methodology, Revisited, *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), pp. 209–218, ACM Press*, 1998.

[CD97] R. Cramer, I. Damgård: Linear Zero-Knowledge: A Note on Efficient Zero-Knowledge Proofs and Arguments, *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC), pp. 436–445, ACM Press*, 1997.

[CDN01] R. Cramer, I. Damgård, J. Nielsen: Multiparty Computations from Threshold Homomorphic Encryption, *Advances in Cryptology — Proceedings of Eurocrypt 2001, Lecture Notes in Computer Science, Vol. 2045, pp. 495–511, Springer-Verlag*, 2001.

[CS98] R. Cramer, V. Shoup: A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attacks, *Advances in Cryptology — Proceedings of Crypto '98, Lecture Notes in Computer Science, Vol. 1462, pp. 13–25, Springer-Verlag*, 1998.

[CS00] R. Cramer, V. Shoup: Signature Schemes Based on the Strong RSA Assumption, *ACM Transactions on Information and System Security (ACM TISSEC), Vol. 3(3), pp. 161–185*, 2000.

[D89] I. Damgard: On the Existence of Bit Commitment Schemes and Zero-Knowledge Proofs, *Advances in Cryptology — Proceedings of Crypto '89, Lecture Notes in Computer Science, Vol. 435, pp. 17–29, Springer-Verlag*, 1990.

[D95] I. Damgård: Practical and Provable Secure Release of a Secret and Exchange of Signature, *Journal of Cryptology, Vol. 8, pp. 201–222, Springer-Verlag*, 1995.

[D00] I. Damgård: Efficient Concurrent Zero-Knowledge in the Auxiliary String Model, *Advances in Cryptology — Proceedings of Eurocrypt 2000, Lecture Notes in Computer Science, Vol. 1807, pp. 418–430, Springer-Verlag*, 2000.

[DN01] I. Damgård, J. Nielsen: Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor, *Cryptology ePrint Archive Report 2001/091, available at* `http://eprint.iacr.org/`, 2001.

[DPP93] I. Damgård, T. Pedersen, B. Pfitzmann: On the Existence of Statistically Hiding Bit Commitment Schemes and Fail-Stop Signatures, *Advances in Cryptology — Proceedings of Crypto '93, Lecture Notes in Computer Science, Vol. 773, pp. 250–255, Springer-Verlag*, 1993.

[DIO98] G. Di Crescenzo, Y. Ishai, R. Ostrovsky: Non-interactive and Non-Malleable Commitment, *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), pp. 141–150, ACM Press*, 1998.

[DKOS01] G. Di Crescenzo, J. Katz, R. Ostrovsky, A. Smith: Efficient And Non-Interactive Non-Malleable Commitment, *Advances in Cryptology — Proceedings Eurocrypt 2001, Lecture Notes in Computer Science, Vol. 2045, pp. 40–59, Springer Verlag*, 2001.

[DO99] G. Di Crescenzo, R. Ostrovsky: On Concurrent Zero-Knowledge with Pre-Processing, *Advances in Cryptology — Proceedings of Crypto '99, Lecture Notes in Computer Science, Vol. 1666, pp. 485–502, Springer-Verlag*, 1999.

[DH76] W. Diffie, M. Hellman: New Directions in Cryptography, *IEEE Transaction on Information Theory, Vol. 22, pp. 644–654*, 1976.

[DM00] Y. Dodis, S. Micali: Parallel Reducibility for Information-Theoretically Secure Computation, *Advances in Cryptology — Proceedings of Crypto 2000, Lecture Notes in Computer Science, Vol. 1880, pp. 414–432, Springer-Verlag*, 2000.

[DDN00] D. Dolev, C. Dwork, M. Naor: Non-Malleable Cryptography, *SIAM Jornal on Computing, Vol. 30, No. 2, pp. 391–437*, 2000.

[DNRS99] C. Dwork, M. Naor, O. Reingold, L. Stockmeyer: Magic Functions, *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 523–534, IEEE Computer Society Press*, 1999.

[EG85] T. ElGamal: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, *IEEE Transaction on Information Theory, Vol. 31, pp. 469–472*, 1985.

[FFS88] U. Feige, A. Fiat, A. Shamir: Zero Knowledge Proofs of Identity, *Journal of Cryptology, Vol. 1, pp. 77–94, Springer-Verlag*, 1988.

[FS89] U. Feige, A. Shamir: Zero-Knowledge Proofs in Two Rounds, *Advances in Cryptology — Proceedings of Crypto '89, Lecture Notes in Computer Science, Vol. 435, pp. 526–544, Springer-Verlag*, 1990.

[FS90] U. Feige, A. Shamir: Witness Indistinguishable and Witness Hiding Protocols, *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing (STOC), pp. 416–426, ACM Press*, 1990.

[FS86] A. Fiat, A. Shamir: How to Prove Yourself: Practical Solutions to Identification and Signature Schemes, *Advances in Cryptology — Proceedings of Crypto '86, Lecture Notes in Computer Science, Vol. 263, pp. 186–194, Springer-Verlag*, 1986.

[F97a] M. Fischlin: Incremental Cryptography and Memory Checkers, *Advances in Cryptology — Proceedings of Eurocrypt '97, Lecture Notes in Computer Science, Vol. 1233, pp. 393–408, Springer-Verlag*, 1997.

[F97b] M. Fischlin: Practical Memory Checkers for Stacks, Queues and Deques, *Information Security and Privacy — ACISP '97, Lecture Notes in Computer Science, Vol. 1270, pp. 114–125, Springer-Verlag*, 1997.

[F97c] M. Fischlin: Lower Bounds for the Signature Size of Incremental Schemes, *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 438–447, IEEE Computer Society Press*, 1997.

[F99] M. Fischlin: Pseudorandom Function Tribe Ensembles Based on One-Way Permutations: Improvements and Applications, *Advances in Cryptology — Proceedings of Eurocrypt '99, Lecture Notes in Computer Science, Vol. 1592, pp. 429–444, Springer-Verlag*, 1999.

[F00] M. Fischlin: A Note on Security Proofs in the Generic Model, *Advances in Cryptology — Proceedings of Asiacrypt 2000, Lecture Notes in Computer Science, Vol. 1976, pp. 458–469, Springer-Verlag*, 2000.

[FF00] M. Fischlin, R. Fischlin: Efficient Non-Malleable Commitment Schemes, *Advances in Cryptology — Proceedings of Crypto 2000, Lecture Notes in Computer Science, Vol. 1880, pp. 414–432, Springer-Verlag*, 2000.

[F01a] M. Fischlin: A Cost-Effective Pay-Per-Multiplication Comparison Method for Millionaires, *RSA Security 2001 Cryptographer's Track, Lecture Notes in Computer Science, Vol. 2020, pp. 457–471, Springer-Verlag*, 2001.

[F01b] M. Fischlin: Cryptographic Limitations on Parallelizing Membership and Equivalence Queries with Applications to Random Self-Reductions, *Theoretical Computer Science, Vol. 268, pp. 199-219, Elsevier Science B.V. (a preliminary version appeared in 9th International Conference on Algorithmic Learning Theory — ALT '98, Lecture Notes in Artificial Intelligence/Computer Science, Vol. 1501, pp. 72–84, Springer-Verlag, 1998)*, 2001.

[F02] M. Fischlin: On the Impossibility of Constructing Non-Interactive Statistically-Secret Protocols from any Trapdoor One-Way Funtion, *RSA Security 2002 Cryptographer's Track, Lecture Notes in Computer Science, Springer-Verlag*, 2002.

[FF02] M. Fischlin, R. Fischlin: The Representation Problem Based on Factoring, *RSA Security 2002 Cryptographer's Track, Lecture Notes in Computer Science, Springer-Verlag*, 2002.

[GHY87] Z. Galil, S. Haber, M. Yung: Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public-Key Model, *Advances in Cryptology — Proceedings of Crypto '87, Lecture Notes in Computer Science, Vol. 293, pp. 135–155, Springer-Verlag*, 1987.

[GHR99] R. Gennaro, S. Halevi, T. Rabin: Secure Hash-and-Sign Signatures Without the Random Oracle, *Advances in Cryptology — Proceedings of Eurocrypt '99, Lecture Notes in Computer Science, Vol. 1592, pp. 123–139, Springer-Verlag*, 1999.

[G98] O. Goldreich: Foundations of Cryptography (Fragments of a Book), *version 2.03, available at* `http://www.wisdom.weizmann.ac.il/home/oded/public_html/index.html`, 1998.

[G00] O. Goldreich: Secure Multi-Party Computation, *(working draft, version 1.2), available at* `http://www.wisdom.weizmann.ac.il/home/oded/public_html/pp.html`, March 2000.

[GMW91] O. Goldreich, S. Micali, A. Wigderson: Proofs that Yield Nothing but Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems, *Journal of the ACM, Vol. 38, No. 1, pp. 691–729, ACM Press*, 1991.

[GMW87] O. Goldreich, S. Micali, A. Wigderson: How to Play any Mental Game, *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing (STOC), pp. 218–229, ACM Press*, 1987.

[GL01] O. Goldreich, Y. Lindell: Session-Key Generation Using Human Passwords Only, *Advances in Cryptology — Proceedings Crypto 2001, Lecture Notes in Computer Science, vol. ???, , pp. ???–???, Springer-Verlag*, 2001.

[GGM86] S. Goldwasser, O. Goldreich, S. Micali: How to Construct Random Functions, *Journal of the ACM, Vol. 33, pp. 792–807*, 1986.

[GL90] S. Goldwasser, L. Levin: Fair Computation of General Functions in Presence of Immoral Majority, *Advances in Cryptology — Proceedings Crypto 90, Lecture Notes in Computer Science, Vol. 537, pp. 75–84, Springer-Verlag*, 1990.

[GMR89] S. Goldwassser, S. Micali, C. Rackoff: The Knowledge Complexity of Interactive Proof Systems, *SIAM Journal on Computation, Vol. 18, pp. 186–208*, 1989.

[GMR88] S. Goldwasser, S. Micali, R. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks, *SIAM Journal on Computing, Vol. 17, No. 2, pp. 281–308*, 1988.

[GQ88] L. Guillou, J.-J. Quisquater: A Practical Zero-Knowledge Protocol Fitted to Security Microprocessors Minimizing Both Transmission and Memory, *Advances in Cryptology — Proceedings of Eurocrypt '88, Lecture Notes in Computer Science, Vol. 330, pp. 123–128, Springer-Verlag*, 1988.

[H99] S. Halevi: Efficient Commitment Schemes with Bounded Sender and Unbounded Receiver, *Journal of Cryptology, Vol. 12, No. 2, pp. 77–90, Springer-Verlag*, 1999.

[HM96] S. Halevi, S. Micali: Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing, *Advances in Cryptology — Proceedings of Crypto '96, Lecture Notes in Computer Science, Vol. 1109, pp. 201–215, Springer-Verlag*, 1996.

[HILL99] J. Håstad, R. Impagliazzo, L. Levin, M. Luby: A Pseudorandom Generator from any One-way Function, *SIAM Journal on Computing, Vol. 28, No. 4, pp. 1364–1396*, 1999.

[IL89] R. Impagliazzo, M. Luby: One-Way Functions are Essential for Complexity Based Cryptography, *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 230–235, IEEE Computer Society Press*, 1989.

[JIS96] M. Jakobsson, R. Impagliazzo, K. Sako: Designated Verifier Proofs and Their Appliactions, *Advances in Cryptology — Proceedings of Eurocrypt '96, Lecture Notes in Computer Science, Vol. 1070, pp. 143–154, Springer-Verlag*, 1996.

[KR99] H. Krawczyk, T. Rabin: Chameleon Hashing and Signatures, *Network and Distributed System Security Symposium 2000, pp. 143–154*, 2000.

[L00] Y. Lindell: Private Communication, *communicated through Ran Canetti*, 2000.

[MR91] S. Micali, P. Rogaway: Secure Computation, *Advances in Cryptology — Proceedings of Crypto '91, Lecture Notes in Computer Science, Vol. 576, pp. 392–404, Springer-Verlag*, 1991.

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

[NOVY98] M. Naor, R. Ostrovsky, R. Venkatesan, M. Yung: Perfect Zero-Knowledge Arguments for NP Using Any One-Way Permutation, *Journal of Cryptology, Vol. 11, pp. 87–108, Springer-Verlag*, 1998.

[NY89] M. Naor, M. Yung: Universal One-Way Hash Functions and Their Cryptographic Applications, *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing (STOC), pp. 33–43, ACM Press*, 1989.

[O92] T. Okamoto: Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes, *Advances in Cryptology — Proceedings of Crypto '92, Lecture Notes in Computer Science, Vol. 740, pp. 31–53, Springer-Verlag*, 1992.

[OS90] H. Ong, C.P. Schnorr: Fast Signature Generation with a Fiat-Shamir Identification Scheme, *Advances in Cryptology — Proceedings of Eurocrypt '90, Lecture Notes in Computer Science, Vol. 473, pp. 432–440, Springer-Verlag*, 1990.

[OW93] R. Ostrovsky, A. Wigderson: One-Way Functions are Essential for Non-Trivial Zero-Knowledge, *Proceedings of the Second Israel Symposium on Theory of Computing and Systems*, 1993.

[P91] T. Pedersen: Non-Interactive and Information-Theoretical Secure Verifiable Secret Sharing, *Advances in Cryptology — Proceedings of Crypto '91, Lecture Notes in Computer Science, Vol. 576, pp. 129–140, Springer-Verlag*, 1991.

[PW94] B. Pfitzmann, M. Waidner: A General Framework for Formal Notions of Secure Systems, *Hildesheimer Informatik-Berichte 11/94, Universität Hildesheim, available at* `http://www.semper.org/sirene/`, 1994.

[PW01] B. Pfitzmann, M. Waidner: A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission, *IEEE Symposium on Security and Privacy, IEEE Computer Society Press*, 2001.

[RS91] C. Rackoff, D. Simon: Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack, *Advances in Cryptology — Proceedings of Crypto '91, Lecture Notes in Computer Science, Vol. 576, pp. 433–444, Springer-Verlag*, 1991.

[RSA78] R. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Communication of the ACM, Vol. 21, No. 2, pp. 120–126, ACM Press*, 1978.

[S91] C.P. Schnorr: Efficient Signature Generation by Smart Cards, *Journal of Cryptology, Vol. 4, pp. 161–174, Springer-Verlag*, 1991.

[S96] C.P. Schnorr: Security of $2^t$-Root Identification and Signatures, *Advances in Cryptology — Proceedings of Crypto '96, Lecture Notes in Computer Science, Vol. 1109, pp. 143–156, Springer-Verlag*, 1996.

[ST01] A. Shamir, Y. Tauman: Improved On-line/Off-line Signature Schemes, *Advances in Cryptology — Proceedings of Crypto 2001, Lecture Notes in Computer Science, Vol. 2139, pp. 355–367, Springer-Verlag*, 2001.

[S99] V. Shoup: On the Security of a Practical Identification Scheme, *Journal of Cryptology, Vol. 12, pp. 247–260, Springer-Verlag*, 1999.

[T00] S. Thomas: SSL and TLS Essentials — Securing the Web, *John Wiley & Sons, Inc.*, 2000.

[Y82] A. Yao: Theory and Applications of Trapdoor Functions, *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (FOCS), pp. 80–91, IEEE Computer Society Press*, 1982.

# Index

*Kapitel A*

# Zusammenfassung

Die klassische Fragestellung der Kryptographie behandelt Praktikabilität und Sicherheit von Verschlüsselungsverfahren. Allerdings rücken mit der zunehmenden Digitalisierung auch andere Gebiete wie digitale Unterschriften, Identifikation, elektronisches Geld, elektronische Wahlen etc. in den Vorderung. Ein möglicher Baustein zum Entwurf sicherer Lösungen stellen Hinterlegungsverfahren dar.

## 1. Hinterlegungsverfahren

Hinterlegungsverfahren lassen sich durch abschließbare Kästen veranschaulichen. In der Hinterlegungsphase des Protokols legt der Sender eine Nachricht in den Kasten, schließt ihn ab und übergibt ihn dem Empfänger. Einerseits erhält der Empfänger zwar keine Informationen über die Nachricht, aber andererseits kann der Sender die so hinterlegte Nachricht nicht mehr verändern. Die erste Eigenschaft nennt man *Geheimhaltung*, die zweite *Eindeutigkeit*. In der Aufdeckphase schickt der Sender dem Empfänger den Schlüssel des Kastens, so dass der Empfänger die Nachricht entnehmen kann.

Digital implementieren kann man ein Hinterlegungsverfahren beispielsweise unter der Diskreten-Logarithmus-Annahme (DL-Annahme). Sei $p$ prim und $g$ ein Generator ein Untergruppe $G_q$ von $\mathbb{Z}_p^*$ mit primer Ordnung $q$. Dann besagt die DL-Annahme, dass man zu gegebenem $h = g^x$ für zufälliges $x \in \mathbb{Z}_q^*$ den diskreten Logarithmus $x = \log_g h$ von $h$ zu $g$ nicht effizient berechnen kann. Das Hinterlegungsverfahren sieht damit wie folgt aus: Gegeben $g, h$ und die Nachricht $m \in \mathbb{Z}_q$ wählt der Sender ein zufälliges $r \in \mathbb{Z}_q$, berechnet $M = g^m h^r$ und schickt diesen "Kasten" $M$ als Hinterlegung an den Empfänger. Später in der Aufdeckphase öffnet der Sender den Kasten, indem er dem Empfänger $m, r$ übergibt. Der Empfänger überprüft dann, dass $m, r \in \mathbb{Z}_q$ und dass diese Werte tatsächlich auf die Hinterlegung $M$ abgebildet werden.

Das Gruppenelement $M$ is unabhängig von der Nachricht $m$ uniform in $G_q$ verteilt und hält damit $m$ geheim. Während die Nachricht somit informationstheoretisch geschützt ist, wird die Eindeutigkeit durch die DL-Annahme impliziert. Denn um ein $M = g^m h^r$ später mit einer anderen Nachricht $m' \neq m$ und $r'$ aufdecken zu können, muß der Sender die DL-Annahme widerlegen. Für solche Werte $m, r$ und $m', r'$ gilt nämlich

$$g^m h^r = M = g^{m'} h^{r'} \qquad \text{bzw.} \qquad g^{m-m'} = h^{r'-r}.$$

Damit ist $\log_g h = (m - m')(r' - r)^{-1} \bmod q$, wobei $(r' - r)^{-1}$ das Inverse zu $r' - r \neq 0$ in $\mathbb{Z}_q$ ist (da $m \neq m'$ gilt auch $r \neq r'$).

Hinterlegungsverfahren kann man beispielsweise bei Auktionen mit geheimen Geboten verwenden. Jeder Teilnehmer hinterlegt zunächst sein Gebot beim Auktionator. Nachdem alle Gebote eingegangen sind, öffnen alle Teilnehmer ihre Hinterlegungen und der Auktionator verkündet den Gewinner. Die beiden grundlegenden Eigenschaften von Hinterlegungen spiegeln sich hier wider: Geheimhaltung garantiert, dass kein Teilnehmer die Gebote der anderen Teilnehmer in der ersten Phase erfährt, und Eindeutigkeit erlaubt es keinem Teilnehmer, sein Gebot in der Aufdeckphase nachträglich zu ändern.

Allerdings benötigt man im Auktionsfall eine weitere, zusätzliche Eigenschaft neben Geheimhaltung und Eindeutigkeit: die *Robustheit*. Diese Eigenschaft besagt unter anderem, dass man aus einer gegebenen Hinterlegung keine Hinterlegung einer verwandten Nachricht erzeugen kann. Im Fall der Auktion verhindert dies beispielsweise, dass man aus der Hinterlegung eines Gebots die Hinterlegung eines höheren Wertes machen kann. In diesem Fall würde der zweite Teilnehmer stets den ersten Teilnehmer überbieten.

Das oben vorgestellte Hinterlegungsprotokol basierend auf dem diskreten Logarithmus ist nicht robust. Gegeben die Hinterlegung $M = g^m h^r$ eines Teilnehmers kann ein Angreifer die Hinterlegung $M^* = gM = g^{m+1} h^r$ erzeugen, ohne $m$ zu kennen. Öffnet nun der erste Teilnehmer der Auktion seine Hinterlegung $M$ mit $m, r$, so kann der Angreifer seine Hinterlegung $M^*$ mit $m^* = m+1$ und $r$ aufdecken und überbietet damit das Gebot $m$.

## 2. Hinterlegungsverfahren mit Geheimtür

In der vorliegenden Arbeit befassen wir uns mit Hinterlegungsverfahren mit Geheimtüren. Informell sind dies Kästen, bei denen jeweils eine kleine geheime Tür eingebaut wurde. Wer diese Tür kennt, kann dadurch später die Nachricht ändern und die Eindeutigkeitseigenschaft aushebeln. Zunächt scheint dies zu implizieren, dass die Eindeutigkeit damit ad absurdum geführt wird. Jedoch ist diese kleine Tür geheim, und nur wer dieses Geheimnis kennt, kann sie benutzen. Ohne Kenntnis dieser Tür bleibt die Eindeutigkeitseigenschaft erhalten.

Als Beispiel betrachten wir erneut das Verfahren basierend auf dem diskreten Logarithmus. Hier ist $M = g^m h^r$ und um eine solche Hinterlegung zu ändern, genügt die Kenntnis des diskreten Logarithmus $x = \log_g h$ von $h$ zu $g$. Denn dann kann man $M$ für jede Nachricht $m'$ mit $r' = r + (m - m')x^{-1} \bmod q$ öffnen:

$$M = g^m h^r = g^m h^{r'-(m-m')x^{-1}} = g^m h^{r'} g^{m'-m} = g^{m'} h^{r'}.$$

Hinterlegungsverfahren mit Geheimtüren erweisen sich als sehr nützlich für den Entwurf sicherer kryptographischer Protokolle. Wir erläutern dies am oben angeführten Beispiel von robusten Hinterlegungsverfahren. Ein Hinterlegungsverfahren heißt robust, wenn es gleichgültig ist, ob der Angreifer zunächst die Hinterlegung der ursprünglichen Nachricht sieht oder nicht: seine Erfolgsaussicht z.B. ein höheres Gebot abzugeben, wird dadurch nicht beeinflußt. Dies bedeutet, dass ein robustes Hinterlegungsverfahren Unabhängigkeit zwischen den Hinterlegungen garantiert.

Wie können wir nun robuste Hinterlegungsverfahren mit Hilfe von Geheimtüren konstruieren? Angenommen im Auktionsbeispiel gibt der Teilnehmer sein Gebot durch eine Hinterlegung mit Geheimtür ab, während die des Angreifers keine solche Geheimtür besitzt. Im folgenden Gedankenexperiment kennen wir die Geheimtür und präsentieren dem Angreifer zunächst eine Hinterlegung des Gebots 0 im Namen des ehrlichen Senders. Der Angreifer gibt sein Gebot ab, und wir ändern —unsichtbar für den Angreifer— durch die Geheimtür anschließend die ursprüngliche Hinterlegung auf den korrekten Wert. Der Angreifer kann sein Gebot nicht mehr abändern, da keine Geheimtür existiert und die Hinterlegung eindeutig ist. Zusammenfassend erhält der Angreifer bei diesem Experiment zum Zeitpunkt, zu dem er sein Gebot unwiderruflich festlegt, nur eine Hinterlegung des Wertes 0 als einzige, redundante Information über das Gebot des ersten Teilnehmers. Da für den Angreifer dieses Experiment und ein wirklicher Protokollablauf nicht zu unterscheiden sind, ist die Erfolgswahrscheinlichkeit des Angreifers in beiden Fällen gleich. Dies deutet an, dass das Verfahren robust ist.

Bei Auktionen verwenden allerdings alle Teilnehmer den selben Kastentyp, so dass entweder die Kästen aller Teilnehmer eine Geheimtür besitzen, oder aber kein Kasten. Dadurch ist das obige Argument nicht mehr ohne weiteres anwendbar. Dies zeigt auch das Beispiel des diskreten Logarithmus: dieses Hinterlegungsverfahren besitzt zwar eine Geheimtür, ist aber trotdem nicht robust. Die Lösung ist es, individuelle Kästen zu verwenden, so dass der erste Teilnehmer einen Kasten mit Geheimtür erhält, während der Angreifer einen ohne verwenden muß. In diesem Fall ist das Argument wieder gültig. Mit anderen Worten, wir suchen Hinterlegungsverfahren mit einer Geheimtür, die nur in Verbindung mit einer bestimmten Identität benutzt werden kann, etwa mit der des ersten Teilnehmers.

In der vorliegenden Arbeit führen wir solche Hinterlegungsverfahren mit identitätsabhängiger Geheimtür ein, beschreiben Konstruktionen und betrachten Anwendungen, z.B. für den Entwurf robuster Hinterlegungen.

Das Beispiel des diskreten Logarithmus läßt sich auf Hinterlegung mit identitätsabhängiger Geheimtür fortsetzen. Statt zweier Generatoren $g, h$ stehen diesmal drei Generatoren $g_1, g_2, h$ zur Verfügung. Um eine Nachricht $m \in \mathbb{Z}_q$ unter seiner Identität $\mathrm{id} \in \mathbb{Z}_q$ zu hinterlegen, wählt der Sender wieder $r \in \mathbb{Z}_q$ zufällig und berechnet

$$M = (g_1^{\mathrm{id}} g_2)^m h^r.$$

Er sendet diesen Wert $M$ and den Empfänger und deckt später erneut mit $m, r$ auf.

Sei $x$ der Logarithmus von $h$ zu $g_1^{\mathrm{id}_0} g_2$ für eine feste Identität $\mathrm{id}_0$. Dann ist die Geheimtür $x$ an diese Identität $\mathrm{id}_0$ gebunden, denn aus $M = (g_1^{\mathrm{id}_0} g_2)^m h^r$ kann man wie vorher eine Hinterlegung für $M = (g_1^{\mathrm{id}_0} g_2)^{m'} h^{r'}$ erzeugen. Für andere Identitäten $\mathrm{id} \neq \mathrm{id}_0$ gilt allerdings nachwievor die Eindeutigkeit, denn eine korrekte Aufdeckung der Hinterlegung $M$ durch $m, r$ und $m' \neq m, r'$ impliziert:

$$\begin{aligned}(g_1^{\mathrm{id}} g_2)^m h^r = (g_1^{\mathrm{id}} h^x g_1^{-\mathrm{id}_0})^m h^r = g_1^{m(\mathrm{id}-\mathrm{id}_0)} h^{r+xm} \\ = M = (g_1^{\mathrm{id}} g_2)^{m'} h^{r'} = (g_1^{\mathrm{id}} h^x g_1^{-\mathrm{id}_0})^{m'} h^{r'} = g_1^{m'(\mathrm{id}-\mathrm{id}_0)} h^{r'+xm'}\end{aligned}$$

und damit

$$g_1^{(m-m')(\mathrm{id}-\mathrm{id}_0)} = h^{(r'-r)+x(m'-m)}.$$

Dies widerspricht selbst bei Kenntnis von $x$ der DL-Annahme, da $\log_{g_1} h$ wegen $m - m', \mathrm{id} - \mathrm{id}_0 \neq 0 \bmod q$ daraus berechenbar ist.

## 3. Übersicht

In Kapitel 2 der Arbeit führen wir die kryptographischen Grundlagen wie die DL-Annahme ein und definieren Hinterlegungsverfahren mit und ohne Geheimtür, sowie mit identitätsabhängiger Geheimtür. Kapitel 3 behandelt dann die Konstruktion solcher Hinterlegungsverfahren mit (identitätsabhängiger) Geheimtür, basierend auf kryptographisch schwierigen Problemen wie Brechen der RSA- oder DL-Annahme.

### 3.1. Effiziente Robuste Hinterlegungsverfahren

In Kapitel 4 stellen wir effiziente robuste Hinterlegungsverfahren basierend auf dem diskreten Logarithmus und der RSA-Annahme vor. Das DL-Protokoll ist in Abbildung 1 dargestellt. Dabei führt der Sender eine nicht-robuste Hinterlegung $M = g_0^m h_0^r$ aus, und beweist zusätzlich mit einem interaktivem Beweis, dass er die "versteckte" Nachricht $m$ in $M$ wirklich kennt (Werte $A, S, a, b, c, s, t, u, y, z$). Versucht ein Angreifer nun beispielsweise eine solche Hinterlegung zu modifizieren,

und mit $m+1$ den anderen Teilnehmer zu überbieten, so kann er zwar $M$ modifizieren, den interaktiven Beweis allerdings nicht. Dies liegt im wesentlichen an der identitätsbasierten Geheimtür der Hinterlegung $A := (g_1 M)^a h_1^u$, bei der $M$ die Rolle der Identität übernimmt. Daher muß der Angreifer einen "unabhängigen" interkativen Beweis führen, dass er $m+1$ und damit $m$ kennt —dies widerspricht allerdings der Geheimhaltung der Hinterlegung $M$.

ABBILDUNG 1. Robustes Hinterlegungsverfahren

| **Sender** | $G_q, g_0, g_1, h_0, h_1$ | **Empfänger** |
|---|---|---|
| Nachricht $m \in \mathbb{Z}_q^*$ | | |
| *a) Hinterlegungsphase*: | | |
| wähle $a, r, s, t, u \in_{\mathrm{R}} \mathbb{Z}_q$ | | |
| setze $M := g_0^m h_0^r$ | | |
| setze $A := (g_1 M)^a h_1^u$ | | |
| setze $S := g_0^s h_0^t$ | $\xrightarrow{M, A, S}$ | wähle $b \in_{\mathrm{R}} \mathbb{Z}_q$ |
| | $\xleftarrow{b}$ | |
| setze $c := a + b \bmod q$ | | |
| setze $y := s + cm \bmod q$ | | |
| setze $z := t + cr \bmod q$ | $\xrightarrow{a, u, y, z}$ | berechne $c := a + b \bmod q$ |
| | | überprüfe $A \stackrel{!}{=} (g_1 M)^a h_1^u$ |
| | | überprüfe $SM^c \stackrel{!}{=} g_0^y h_0^z$ |
| *b) Aufdeckphase*: | | |
| | $\xrightarrow{m, r}$ | überprüfe $M \stackrel{!}{=} g_0^m h_0^r$ |

Ferner wird in Kapitel 4 die Definition von Robustheit untersucht. Soll der Angreifer nicht in der Lage sein, eine Hinterlegung einer verwandten Nachricht zu erzeugen, oder ist dies zulässig, aber dann soll der Angreifer seine Hinterlegung nicht korrekt aufdecken können? Beide Möglichkeiten wurden implizit in der Literatur verwendet, und wir arbeiten die Unterschiede heraus und zeigen unter kryptographischen Annahmen, dass der zweite Begriff nur ein schwächeres Sicherheitsniveau garantiert.

Die Ergebnisse dieses Kapitels stammen aus einer gemeinsamen Arbeit mit Roger Fischlin, erschienen auf der Konferenz Crypto 2000 [FF00].

### 3.2. Zurücksetzbare Identifikation

In Kapitel 5 betrachten wir Identifikationsprotokolle. Durch ein Identifikationsprotokoll beweist ein Proponent $\mathcal{P}$ dem Opponenten $\mathcal{V}$ seine Identität, indem er zeigt, dass er den geheimen Schlüssel zu einem öffentlichen Schlüssel kennt. Als Beispiel ist das Okamoto-Schnorr-Identifikationsprotokoll in Abbildung 2 angegeben.

ABBILDUNG 2. Okamoto-Schnorr-Identifikation

| **Proponent** $\mathcal{P}$ | öffentlicher Schlüssel: $G_q, g_0, g_1, X$ | **Opponent** $\mathcal{V}$ |
|---|---|---|
| geheimer Schlüssel: $x_1, x_2 \in \mathbb{Z}_q$ mit $X = g_1^{x_1} g_2^{x_2}$ | | |
| wähle $r_1, r_2 \in_{\mathrm{R}} \mathbb{Z}_q$ | | |
| setze $R := g_1^{r_1} g_2^{r_2}$ | $\xrightarrow{R}$ | wähle $c \in_{\mathrm{R}} \mathbb{Z}_q$ |
| | $\xleftarrow{c}$ | |
| setze $y_1 := r_1 + cx_1 \bmod q$ | | |
| setze $y_2 := r_2 + cx_2 \bmod q$ | $\xrightarrow{y_1, y_2}$ | überprüfe $RX^c \stackrel{!}{=} g_1^{y_1} g_2^{y_2}$ |

Ein passiver Angreifer versucht, sich als der Proponent $\mathcal{P}$ auszugeben, und sich ohne Kenntnis des geheimen Schlüssels gegenüber $\mathcal{V}$ zu identifizieren. Ein aktiver Angreifer führt zunächst eine Experimentierphase aus, bei der er einige Protokollabläufe mit dem ehrlichen Proponenten $\mathcal{P}$ in der Rolle des Opponenten ausführt. Dadurch kann der aktive Angreifer versuchen, Informationen über den geheimen Schlüssel zu erhalten, um so die Identifikation mit dem ehrlichen Opponenten $\mathcal{V}$ leichter zu bestehen.

Eine stärkere Sicherheitsanforderung erlaubt es dem aktiven Angreifer nun sogar, Ausführungen mit dem Proponenten zurückzusetzen. So kann der Angreifer in der Experimentierphase beim Angriff auf das Okamoto-Schnorr-Verfahren etwa eine Ausführung mit Daten $(R, c, y_1, y_2)$ zurücksetzen und diese Ausführung mit dem gleichen Initialwert $R$ aber anderem $c'$ (und damit anderer Antwort $y_1', y_2'$) beenden. Das Okamoto-Schnorr-Protokoll ist zwar sicher gegen aktive Angriffe, aber beweisbar unsicher gegen Angriffe mit Zurücksetzen. In diesem Kapitel stellen wir basierend auf Hinterlegungen mit Geheimtüren eine allgemeine Modifikation vor, wie dieses und andere bekannte Protokolle sicher gegen solche Angriffe werden.

Um das Zurücksetzen unwirksam zu machen, lassen wir im modifizierten Protokolle den Opponenten den Wert $c$ zunächst durch $C = g^c h^s$ hinterlegen (siehe

ABBILDUNG 3. Zurücksetzbare Okamoto-Schnorr-Identifikation

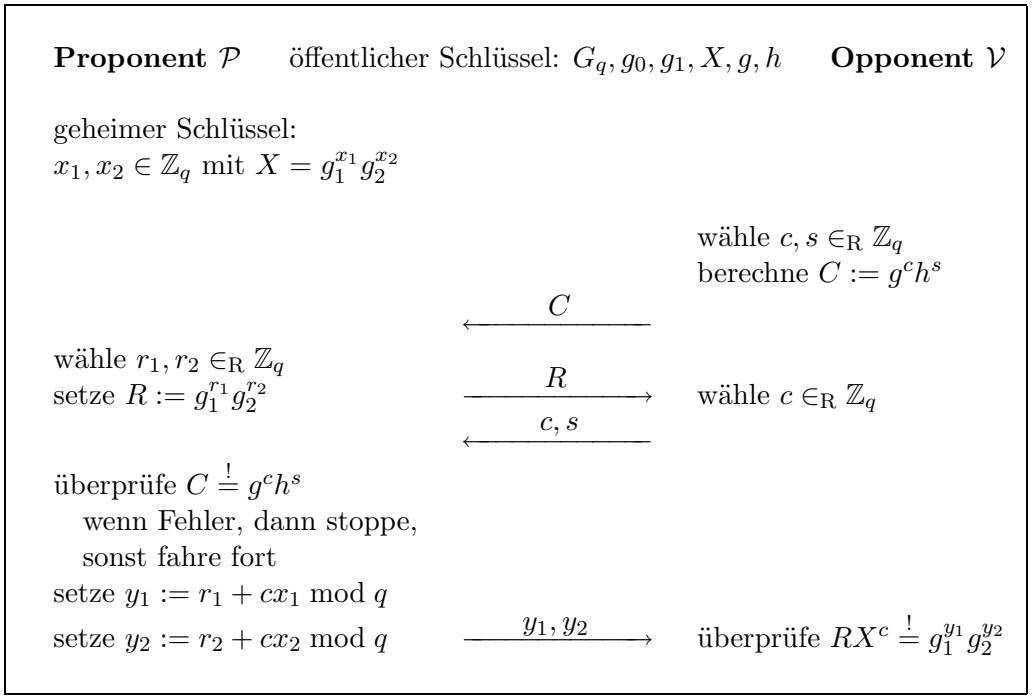



Abbildung 3). Dadurch kann der Opponent in seinem zweiten Schritt nur einen gültigen Wert $c$ aufdecken, und der oben angeführte Angriff mit Zurücksetzen und Senden von $c'$ versagt.

Durch das Einführen einer Hinterlegung von $c$ kann man allerdings auch den ursprünglichen Sicherheitsbeweis gegen aktive Angreifer nicht auf den zurücksetzbaren Fall übertragen. Dieser Beweis beruht nämlich auf der Eigenschaft, dass man in der Ausführung zwischen Angreifer (als Proponent) und Opponent zurücksetzen kann. Durch die Hinterlegung ist dies auch im Beweis nicht mehr möglich. Wählen wir aber stattdessen ein Hinterlegungsverfahren mit Geheimtür (wie in Abbildung 3 bereits mit dem Diskreten-Logarithmus-Verfahren geschehen), kann der Angreifer einerseits nicht zurücksetzen, im Beweis können wir dies jedoch durch die Geheimtür. Der Sicherheitsbeweis vom aktiven Fall überträgt sich dann unmittelbar. Dies gilt allerdings nur für Angreifer, die sich als Proponent ausgeben, nachdem die Experimentierphase vorbei ist. Soll der Angreifer sogar seinen Einbruchversuch während der Experimentierphase starten dürfen, leisten Hinterlegungen mit identitätsabhängigen Geheimtüren Abhilfe.

Die hier vorgestellte Lösungen wurden mit anderen Ansätzen in einer gemeinsamen Arbeit mit Mihir Bellare, Shafi Goldwasser und Silvio Micali auf der Eurocrypt 20001 vorgestellt [BFGM01].

### 3.3. Hinterlegung als Sicheres Unterprotokoll

Im Auktionsfall zeigt das Beispiel der Robustheit, oder genauer: der fehlenden Robustheit, dass Abhängigkeiten zwischen Protokollausführungen ungewollte Nebeneffekte verursachen können. Der Grund dafür ist die übliche Definition von Hinterlegungsverfahren als isolierte Protokolle. In Kapitel 6 betrachten wir daher Hinterlegungen im Zusammenhang mit anderen Protokollen. Dazu führen wir eine Definition von Hinterlegungen ein, die es erlaubt, solche Verfahren als Unterprotokolle in anderen sicheren Protokollen einzusetzen, so dass die Sicherheit des gesamten Protokolls erhalten bleibt. Wir zeigen, dass diese Definition im Fall, dass lediglich zwei Teilnehmer, Sender und Empfänger, aktiv werden, nicht zu erfüllen ist. Hilft allerdings ein zusätzlicher Teilnehmer aus, so könen wir effiziente Verfahren angeben. Diese Lösungen beruhen erneut auf Hinterlegungen mit Geheimtüren.

Dieses Kapitel stellt die Ergebnisse einer gemeinsamen Arbeit mit Ran Canetti dar. Die Arbeit wurde auf der Konferenz Crypto 2001 vorgestellt [CF01].

# Lebenslauf

| | |
|---|---|
| ***Persönliche Daten*** | |
| *Name* | Marc Fischlin |
| *Geburtsdatum* | 23. Februar 1973 |
| *Geburtsort* | Offenbach/Main |
| *Staatsangehörigkeit* | deutsch |
| *Familienstand* | ledig, keine Kinder |
| ***Schulische Ausbildung*** | |
| *1979 – 1992* | Grundschule, Förderstufe, Gymnasium in Hanau, Allgemeine Hochschulreife mit Note 1,2 |
| ***Universitäre Ausbildung*** | |
| *Oktober 1992 – April 1998* | Informatikstudium, Goethe-Universität Frankfurt<br>Diplom mit Note Sehr gut, Diplomarbeitsthema: Fairer Austausch Digitaler Unterschriften, Betreuung von Prof. Dr. Schnorr |
| *April 1995 – Juni 1997* | Mathematikstudium, Goethe-Universität Frankfurt<br>Diplom mit Note Sehr gut, Diplomarbeitsthema: Editierfreundliche Kryptographie, Betreuung von Prof. Dr. Schnorr |
| *Juli 1997 – Februar 1998 und ab April 1999* | Promotionstudium, Fachbereich Mathematik der Goethe-Universität Frankfurt |
| ***Berufserfahrung*** | |
| *1995 – 1997* | Studentische Hilfskraft, Fachbereiche Mathematik und Informatik der Goethe-Universität (mit Unterbrechungen) |
| *Juli 1997 – Februar 1998 und ab April 1999* | Wissenschaftlicher Angestellter, Arbeitsgruppe von Prof. Dr. Schnorr am Fachbereich Mathematik der Goethe-Universität Frankfurt |
| ***Sonstiges*** | |
| *März 1998 – März 1999* | Zivildienst, Röntgenabteilung Klinikum Hanau |