Basic Research in Computer Science

On Protocol Security in the Cryptographic Model

Jesper Buus Nielsen

BRICS Dissertation Series **DS-03-8**

ISSN 1396-7002 **August 2003**

See back inner page for a list of recent BRICS Dissertation Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `DS/03/8/`

On Protocol Security in the Cryptographic Model

Jesper Buus Nielsen

Ph.D. Dissertation

Department of Computer Science
University of Aarhus
Denmark

On Protocol Security in the Cryptographic Model

A Dissertation
Presented to the Faculty of Science
of the University of Aarhus
in Partial Fulfillment of the Requirements for the
Ph.D. Degree

by
Jesper Buus Nielsen
June 21, 2003

Abstract

It seems to be a generally acknowledged fact that you should never trust a computer and that you should trust the person operating the computer even less. This in particular becomes a problem when the party that you do not trust is one which is separated from you and is one on which you depend, e.g. because that party is the holder of some data or some capability that you need in order to operate correctly. How do you perform a given task involving interaction with other parties while allowing these parties a minimal influence on you and, if privacy is an issue, allowing them to learn as little about you as possible. This is the general problem of secure *multiparty computation*. The usual way of formalizing the problem is to say that a number of parties who do not trust each other wish to compute some function of their local inputs, while keeping their inputs as secret as possible and guaranteeing the correctness of the output. Both goals should be obtained even if some parties stop participating or some malevolent coalition of the parties start deviating arbitrarily from the agreed protocol. The task is further complicated by the fact that besides their mutual distrust, nor do the parties trust the channels by which they communicate. A general solution to the secure multiparty computation problem is a compiler which given any feasible function describes an efficient protocol which allows the parties to compute the function securely on their local inputs over an open network.

Over the past twenty years the secure multiparty computation problem has been the subject of a large body of research, both research into the models of multiparty computation and research aimed at realizing general secure multiparty computation. The main approach to realizing secure multiparty computation has been based on verifiable secret sharing as computation, where each party holds a secret share of each input and during the execution computes secret shares of all intermediate values. This approach allows the parties to keep all inputs and intermediate values secret and to pool the shares of the output values to learn exactly these values.

More recently an approach based on joint homomorphic encryption was introduced, allowing for an efficient realization of general multiparty computation secure against an eavesdropper. A joint encryption scheme is an encryption scheme where all parties can encrypt, but where it takes the participation of all parties to open an encryption. The computation then starts by all parties broadcasting encryptions of their inputs and progresses through computing encryptions of the intermediary values using the homomorphic properties of the encryption scheme. The encryptions of the outputs can then be jointly decrypted.

In this dissertation we extend this approach by using threshold homomorphic encryption to provide full-fledged security against an active attacker which might control some of the

parties and which might take over parties during the course of the protocol execution. As opposed to a joint encryption scheme a threshold encryption scheme only requires that a given number of the parties are operating correctly for decryption to be possible. In this dissertation we show that threshold homomorphic encryption allows to solve the secure general multiparty computation problem more efficiently than previous approaches to the problem.

Starting from an open point-to-point network there is a long way to general secure multiparty computation. The dissertation contains contributions at several points along the way. In particular we investigate how to realize *secure channels*. We also show how threshold signature schemes can be used to efficiently realize a *broadcast channel* and how threshold cryptography can be used to provide the parties of the network with a source of *shared randomness*. All three tools are well-known to be extremely powerful resources in a network, and our principle contribution is to provide efficient realizations.

The dissertation also contains contributions to the definitional part of the field of multiparty computation. Most significantly we augment the recent model of *universally composable security* with a formal notion of what it means for a protocol to only realize a given problem under a given *restricted input-output behavior* of the environment. This e.g. allows us to give the first specification of a *universally composable zero-knowledge proof of membership* which does not at the same time require the proof system to be a proof of knowledge.

Acknowledgments

To Ida Buus and Erik Nielsen

with special thanks

to Ran Canetti, Mogens Nielsen and Rafail Ostrovsky for serving on my Ph.D. committee. Especially Ran Canetti and Rafail Ostrovsky, their many valuable comments and suggestions helped improve the dissertation considerably

to the crypto group at BRICS for a great time and a learning time; Ronald Cramer for his competent ACTS dictatorship, Louis Salvail for always laughing and Serge Fehr and Peter Høyer for tolerating me as an office mate, and to Jesus Almansa, Kasper Dupont, Stefand Dziembowski, Jens Groth, Mads Jurik, Eike Kiltz, Maciej Koprowski, Kirill Morozov and Martijn Stam

to everybody how helps making BRICS an absolute amazing place to do basic research in computer science

to the crypto group at IBM T.J. Watson for making my stay there such an enjoyable one. Crypto would not be same without you

to Ran Canetti for sharing so much of his knowledge, about crypto and about New York restaurants. I still miss the commutes between Hawthorn and Manhattan

to Tal Rabin for the drives and talks to and from Hawthorn — I learned at lot, also about crypto — and for the Tiramisu

to the crypto group at Bell Labs for a great day, in particular to Juan Garay for the Danish beverage

to Angelos Keromytis for hosting me as a visiting researcher at Columbia University

to Kim Jersin for discussing many of the ideas in this dissertation with me when we should have been doing completely different things

to those many others who helped me, by discussing an idea, giving me an idea or obliviously letting me steal one, for explaining their work or buying me lunch: Jan Camenisch, David Chaum, Claude Crépeau, Yevgeniy Dodis, Matias Fitzie, Rosario Gennaro, Shai Halevi, Martin Hirt, Yuval Ishai, Jonathan Katz, Aggelos Kiayias, Hugo Krawczyk, Peter Landrock, Yehuda Lindell, Anna Lysyanskaya, Philip MacKenzie, Tal Malkin, Ueli Maurer, Rafael Pass, Omer Reingold, Ove Scavenius, Berry Schoenmakers, Berit Skjernaa, Rebecca Wright, Shouhuai Xu and Moti Yung

and with very special and grateful thanks to Ivan Damgård for teaching me Crypto, for his invaluable guidance, for always having a solution when one is needed and for always having time, for all those half-hour-long *har du fem minutter*-sessions. Thank you for an amazing four years.

Jesper Buus Nielsen,
Århus, June 21, 2003.

c9084d2948be6ca8595a24aba4546cbe

Contents

CHAPTER 1

Introduction

From his cradle to his grave a man never does a single thing which has any first and foremost object but one — to secure peace of mind, spiritual comfort, for himself.
— Mark Twain

The *Merriam-Webster Online Dictionary* has the following to say on dissertations:

```
Main Entry: dis?ser?ta?tion
Pronunciation: "di-s&r-'tA-sh&n
Function: noun
: an extended usually written treatment of a subject;
specifically : one submitted for a doctorate
```

Under this definition, you are reading a dissertation. It is extended, it is written, and its subject is how to build efficient, secure multiparty protocols using threshold cryptography. It was submitted for a doctorate at the Faculty of Science of the University of Aarhus on June 21, 2003.

The dissertation is based on some of the research done during my Ph.D study at BRICS. Parts of the presented material were already published in [DN00, CDN01, Nie02a, Nie02b, DN02b, DN03] together with Damgård and Cramer and some parts are unpublished. Some of the research done during my Ph.D. study did not make it to this dissertation, because it falls outside the topic. This includes some research on cryptographic primitives, in particular on the security of encryption schemes. Parts were published in [DJN01, DN02a, CKN03] together with Damgård, Canetti, Jurik and Krawczyk.

The multiparty computation (MPC) problem is to devise protocols which allow n parties connected by a complete network of point-to-point channels to securely compute any function f on their local inputs and to study under which assumptions and how efficiently such protocols can be devised. *Securely compute* means preserving the correctness of the outputs and the privacy of the inputs in the presence of faults. Faults are modeled by a single entity called the adversary. The adversary controls a set of so-called corrupted parties. If the adversary is passive it sees all inputs and outputs of the corrupted parties and can view their internal state. An active adversary is furthermore allowed to send arbitrary messages

on behalf of the corrupted parties. An adversary has two goals, gathering information about uncorrupted parties' inputs and outputs and, if it is active, exerting influence by sending faulty messages to make the uncorrupted parties terminate with faulty outputs. A protocol which is only secure against passive adversaries is often called privacy preserving.

There are many proposals on how to model the security of a protocol, but common to most is that the security of a protocol is defined by requiring that an adversary attacking the protocol should only be allowed to achieve inevitable goals. This meaning that the adversary should not be able to achieve more than it would from attacking an ideal evaluation of f. In the ideal evaluation we imagine that the parties send their inputs to an incorruptible trusted party over perfectly secure lines. This trusted party then computes the function f on the inputs and returns the result over the perfectly secure lines. Here certainly the adversary archives only inevitable goals: Of information it only learns the corrupted parties' inputs and outputs and if it is active the only influence it can exert is changing the corrupted parties' inputs to the function.

The comparison of the protocol execution to the ideal evaluation is made by requiring that the complete view of an adversary attacking the protocol execution can be simulated given only the view of a correspondingly powerful adversary attacking the ideal evaluation. This captures exactly the idea that the information gathering and the influencing capabilities of the adversary include nothing extra to that of which the adversary is entitled. This approach to comparing the protocol execution to the ideal evaluation originates in the definition of zero-knowledge proof by Goldwasser, Micali and Rackoff [GMR85], where the same approach is used to define that the view of a dishonest verifier does not contain information about the witness, by showing that the view can be simulated without the witness. We call this kind of model of security a simulation model. For the MPC setting the simulation model approach is introduced by Goldreich, Micali and Wigderson [GMW87].

The models used to study MPC vary considerably in the assumptions on the computing power of the parties, the nature of the communication channels, and the classes of faults which are considered. Some common choices for these parameters give rise to the cryptographic model and the information theoretical model. In the cryptographic model, the parties and the adversary are assumed to be probabilistic polynomial time, interactive Turing machines and the channels are assumed to be authenticated but public, i.e. the communication over the channels is publicly readable. In [GMW87] Goldreich, Micali, and Wigderson and independently in [CDG87] Chaum, Damgård, and van de Graaf proved that if less than $n/2$ of the parties deviate from the protocol, then any probabilistic polynomial time n-party function can be computed securely in the cryptographic model. In the information theoretical model, the parties and the adversary are assumed to be computationally unbounded and the channels are assumed to be authenticated and private. In [CCD88] Chaum, Crépeau, and Damgård and independently in [BGW88] Ben-Or, Goldwasser, and Wigderson proved that if less than $n/3$ of the parties are corrupted, then any n-party function can be computed securely. In [RB89] Rabin and Ben-Or and independently in [Bea91] Beaver proved that in the information theoretical model with a broadcast channel any n-party function can be computed securely as long as less than $n/2$ parties are actively corrupted.

We formalize our model of computation and our model of security in Chapter 3. Here we describe three assumptions that we make about the network throughout our investigation

and try to justify why. The assumptions are: *the computation is synchronous*, *the network is complete*, and *the channels are authenticated*.

The reason for considering a synchronous model is that it simplifies the study without (hopefully) abstracting away the problems that are essential to secure MPC. The hope is that tools and methodologies developed in the synchronous model for secure computation can be adapted to the asynchronous model. There is good evidence that this is the case. If we do not consider active faults, then any privacy preserving protocol for the synchronous model can be translated into a privacy preserving protocol for the asynchronous model using a synchronizer, see Awerbuch [Awe85]. Even though no such construction is known for the active case, many of the tools developed for the synchronous model have been adopted to protocol design in the asynchronous setting. One prominent example it the concept of verifiable secret sharing (VSS), which was defined for the synchronous model in [CGMA85] by Chor, Goldwasser, Micali and Wigderson, where it was used to construct a so-called simultaneous broadcast protocol. VSS is an essential part of most MPC protocols. In [CR93] Canetti and Rabin adapted VSS to the asynchronous model. They build an asynchronous VSS (AVSS) protocol and used it to construct asynchronous Byzantine agreement. Also Ben-Or, Canetti and Goldreich [BCG93] and Ben-Or, Kelmer and Rabin [BKR94] used AVSS, as a sub-protocol in building general MPC protocols for the asynchronous information theoretic model. The model introduced in [BCG93] builds directly on the definitional work for the synchronous model begun in [GMW86] and both protocols use the tools and methodologies from the protocols for synchronous MPC protocols in [CCD88, BGW88, RB89]. It should be noted that even though most techniques developed for the synchronous model have been adopted for the asynchronous model, doing so is by far trivial. In particular, the above mentioned adaptations enjoy considerable extra complications in the asynchronous model.

The motivation for considering complete and authenticated networks is partly the same as for studying synchronous computation. For these assumptions, the hope that results in the idealized model apply to more realistic models is even better justified. In [BCK98] Bellare, Canetti and Krawczyk showed how any protocol for the authenticated channel model can be translated into a protocol for the unauthenticated channel model in a modular way using known (and widely applied) techniques for authentication, like authenticated Diffie-Hellman for key exchange and messages authentication codes (MAC's) and digital signatures for data transport. A further discussion of this issue and references to related work can be found in [BCK98]. A study of which general networks allow for simulating a complete network has been performed in a number of papers. E.g., in the information theoretical model, if the network consists of point-to-point channels and t parties might be corrupted, but do not actively cheat, then $t + 1$ disjoint paths between each pair of parties are necessary and sufficient to simulate a complete point-to-point network. If the t corrupted parties may actively cheat, then $2t + 1$ such paths are necessary and sufficient. For more on this subject, see Franklin and Wright [FW00], which also contains similar result for other communication models with multicast and broadcast.

It is the hope that studying secure computation in an idealized model will develop efficient tools and methodologies which have wider applications to protocol design and that it will clarify what is essential aspects of (efficient) secure computation. Furthermore, secure MPC is a place for benchmarking existing tools and methodologies for protocol design, which it is

here our goal to do for threshold cryptography.

The dissertation is structured into three parts. In the first part we set the stages by introducing some well-known cryptographic tools and developing our model of security.

The tools are presented in Chapter 2. The reason why we include definitions of a number of more or less standard cryptographic tools is that the literature differ on the definitions and that it is convenient to phrase the definitions in one common framework, as they will all need to play together in later chapters.

Chapter 3 contains the formalization of our model of security. We will use the model of universally composable (UC) security by Canetti [Can01a]. One of the salient properties of this model is that it guarantees that if a protocol is deemed secure by the model, then the protocol is secure when run in any context, within the model of course. We say that the security is preserved under universal composition. In Chapter 3 we present an extension to the UC framework which allows to specify formally what it means for a protocol to realize a task under a given input-output behavior from the environment in which it is run. This is done in such a way that if the model deem a protocol secure for a given input-output behavior, then it is secure when run in any context exhibiting this input-output behavior. Our main contribution is that we are able to prove this security preservation for input-output behaviors which cannot necessarily be checked in polynomial time. This means that the existing proof that security is preserved under composition in the cryptographic model, where all entities are restricted to polynomial time, does not carry over directly. The extension will among other things allow us to define for the first time a notion of universally composable zero-knowledge proof of membership. The material in Chapter 3 is previously unpublished.

The second part of the dissertation presents a number of basic cryptographic protocols.

In Chapter 4 we investigate how we can add secure channels to our model in a compositional way. The results in this chapter were previously published in [DN00, Nie02a].

In Chapter 5 we present a notion of UC zero-knowledge proof of membership along with an efficiently realization. Having a notion of universally composable zero-knowledge proof of membership will greatly simplify proofs in later chapters as it allows us to use the compositional properties of our security notion. The material in Chapter 5 is previously unpublished.

In Chapter 6 we will take some well-known techniques from threshold cryptography and capture them in the UC framework to allow for a modular use in subsequent chapters. Furthermore, we will show how threshold cryptography can be used to provide the parties in the network with an efficient source of shared randomness. This technique was previously published in [Nie02b].

In Chapter 7 we will investigate the problem of efficiently adding a broadcast channel to our model. This chapter contains several contributions to this area. In particular we will show how one can efficiently deal with the fact that if one uses a round-efficient broadcast protocol, then the parties will necessarily lose synchronization. This gives some problems for protocols in the synchronous model, some of which do not even look as if have been addressed before in full generality. Our primary contribution is a technique which allows to deal with the loss of synchronization, a technique which compared to existing techniques is both more efficient and much simpler. The chapter will also contain a minor optimization of a recent result about efficient deterministic broadcast for the information theoretic model with

amortized constant round complexity, which serves as the most efficient approach to adding broadcast to the synchronous model. We optimize a sub-protocol known as unknown binary set consensus. We also show how to realize efficient amortized constant round broadcast in the cryptographic model, by adopting the above mentioned protocol. The material in Chapter 7 is previously unpublished.

Having these basic protocols in place we are prepared for the third part of the dissertation, where we consider the general MPC problem.

In Chapter 8 we present a solution to the MPC problem based on threshold homomorphic encryption. The communication complexity of the protocol is lower than the best previous MPC protocols and therefore demonstrates that threshold homomorphic encryption is a viable approach to MPC. The protocol in Chapter 8 was previously published in [CDN01].

The protocol in Chapter 8 is only secure against a static adversary, which corrupts all parties before the execution of the protocol. This is an unrealistic model of the adversary. As a step towards making the protocol in Chapter 8 adaptively secure we will present an adaptively secure commitment scheme in Chapter 9. The commitment scheme was previously published in [DN02b].

Using the commitment scheme of Chapter 9 we then show how to introduce adaptive security. This is done in Chapter 10. The protocol in Chapter 10 was previously published in [DN03].

Part I

Setting the Stages

CHAPTER 2

Tools

Have thy tools ready. God will find thee work.
— Charles Kingsley, 1819–1875

In this chapter we treat some tools that will be used in several of the following chapters. Many of these tools are well-known to the reader and can be skipped. We describe them to have a fixed formal definition; For completeness and because the literature vary on the definitions. We have attempted to organize the chapter according to how standard the introduced notions are, so that the reader confident with most standard cryptographic notions can skip a prefix of the sections constituting this chapter. The chapter is organized as follows: We first introduce some notation that we use throughout the dissertation (Section 2.1) and introduce some definitions for random variables (Section 2.2). In Sections 2.3, 2.4 and 2.5 we define notions of function families, public key encryption respectively digital signature schemes. In Section 2.7 we then define some standard complexity theoretic assumptions. In Section 2.8 we define the notion of invertible sampling, which is used in the subsequent Section 2.9, where we define several notions of Σ-protocols. The notion of invertible sampling is also used in various other places throughout the dissertation, see the index. In Section 2.10 we define the notion of N-invertible homomorphisms. That section uses material from both Section 2.7 and Section 2.9. Finally, in Section 2.6 we define some notions of commitment schemes. The reason for placing the definition of commitment scheme so late in the chapter is that it uses material from Section 2.7 and Section 2.10.

2.1 Conventions

In the following we use these conventions:

- $\mathbf{N} = \{0, 1, 2, \ldots\}$, $\mathbf{Z} = \{\ldots, -2, 1, 0, 1, 2, \ldots\}$ and $\mathbf{Z}_n = \{0, 1, \ldots, n-1\}$ for all $n \in \mathbf{N}$. Furthermore, $\mathbf{R}$ denotes the real numbers and $\mathbf{R}^+$ denotes the non-negative real numbers.
- $k \in \mathbf{N}$ denotes the security parameter. All algorithms receive this value as input and the running time is measured in k.

- By a probabilistic polynomial time (PPT) algorithm A we mean a probabilistic algorithm which for some $c \in \mathbf{N}$ on all inputs ($k \in \mathbf{N}, x \in \{0,1\}^*$) always terminates in less than k^c steps. We use $a \leftarrow A(k, x; r)$ to denote running A on input x and uniformly random bits $r \in \{0,1\}^{k^c}$, producing output a. We will often drop the explicit mentioning of k as input to A and we write $r \in \{0,1\}^*$ to avoid mentioning k^c explicitly.
- We use a probabilistic algorithm A to define the set of possible outputs from A. When we write $x \in A(y)$ it means that there exists random inputs $r \in \{0,1\}^*$ such that $x = A(y; r)$.
- We often use $[n]$ to denote the set $\{1, \ldots, n\}$ and sometimes use $[n]$ to denote the set $\{0, 1, \ldots, n-1\}$. The context will reveal what we mean.
- We sometimes use the term uniformly random slightly liberal, to mean statistically close to uniformly random. This e.g. allows us to say *generate a uniformly random element from* $\mathbf{Z}_3$, instead of *generate an element from* $\mathbf{Z}_3$ *with a distribution statistically close to uniform*, which we would often have to as our probabilistic algorithms were defined to take random bits as their source of randomness, which means that they cannot generate a uniformly random element from e.g. $\mathbf{Z}_3$ in finite time.
- ϵ denotes the empty string.
- We use $(\cdot,\cdot) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ to denote some bijective concatenation which can be computed in PPT in both directions, and we sometimes use the notation $x_1 \| x_2 = (x_1, x_2)$. We require that $\epsilon = (\epsilon, \epsilon)$. This bijection can easily be extended such that any string can be parsed uniquely as a t-tuple for any t, using the conventions that $(x) = x$ and $(x, y, \ldots) = (x, (y, \ldots))$. We use this convention extensively. E.g., if for an algorithm taking inputs from $\{0,1\}^*$ we say "on input (x, y, z), where $x \cdots$, do $\cdots$", we mean "on an input X for which $x \cdots$ when $X = (x, y, z)$ do $\cdots$ using the values (x, y, z)". Notice that $\epsilon = (\epsilon, \ldots, \epsilon)$.
- We often use the notation $\{v_i\}_{i \in I}$ to mean the object $(I, \{(i, v_i)\}_{i \in I})$. To distinguish it from the set $\{v_i\}_{i \in I}$, we will call this a family.
- By an l-bit number we mean an integer from the interval $[2^{l-1}, 2^l - 1]$.
- $\mathrm{PRIMES}(l) = \{x \in [2^{l-1}, 2^l - 1] | x \text{ is a prime}\}$, so $\mathrm{PRIMES}(l)$ is the set of l-bit primes.
- We use the definition of interactive Turing machine (ITM) from [Can00].
- A PPT family of sets $\mathcal{I} = (gen, I)$ is given by a PPT generator gen, taking as input the security parameter and outputting $i \in \{0,1\}^*$, along with a PPT computable function $I : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$. For $i \in gen(k)$ we define a set $\mathcal{I}_i$ by $x \in \mathcal{I}_i$ iff $I(i, x) = 1$.

2.2 Random Variables and Distribution Ensembles

We start by defining a notion of negligible function. Basically a negligible function approaches 0 faster than any polynomial.

Definition 2.1 *We say that $f : \mathbf{N} \to \mathbf{R}^+$ is* negligible *if for all $c \in \mathbf{N}$ there exists $k_c \in \mathbf{N}$ such that for all $k > k_c$ we have that $f(k) \leq k^{-c}$. We say that $f : \mathbf{N} \times \{0,1\}^* \to \mathbf{R}^+$ is* negligible *if for all $c \in \mathbf{N}$ there exists $k_c \in \mathbf{N}$ such that for all $k > k_c$ and for all $z \in \{0,1\}^*$ we have that $f(k,z) \leq k^{-c}$.*

We prove a lemma, which will come in handy later, that a polynomial sum of negligible functions is again a negligible function.

Lemma 2.1 *Let $A = \{a_j\}_{j\in J}$ and $F = \{f_j\}_{j\in J}$ be sets of functions $a_j : \mathbf{N} \to \mathbf{R}^+$ and $f_j : \mathbf{N} \times \{0,1\}^* \to \mathbf{R}^+$. Let $g_{A,F}(k,z) = \sum_{j\in J} a_j(k) f_j(k,z)$. If J is finite, a_j is a polynomial for $j \in J$ and f_j is negligible for $j \in J$, then $g_{A,F}$ is negligible.*

Proof. Since a_j is a polynomial and J is finite we can pick $c_A \in \mathbf{N}$ and $k_A \in \mathbf{N}$ such that $k^{c_A} \geq a_j(k)$ for all $j \in J$ and all $k > k_A$. Consider any $c \in \mathbf{N}$. For each $f \in F$, using that it is negligible, there exists $k_{f,c+c_A+1} \in \mathbf{N}$ such that for all $k > k_{f,c+c_A+1}$ and all $z \in \{0,1\}^*$ we have that $f(k,z) \leq k^{-(c+c_A+1)}$. Let $k_{F,c+c_A+1} = \max\{k_{f,c+c_A+1}\}_{f\in F}$. Since F is finite, and can be assumed to be non-empty without loss of generality, we have that $k_{F,c+c_A+1}$ is well-defined and that $f(k,z) \leq k^{-(c+c_A+1)}$ for all $f \in F$, all $k > k_{F,c+c_A+1}$ and all $z \in \{0,1\}^*$. For $k > \max\{|J|, k_A, k_{F,c+c_A+1}\}$ and $z \in \{0,1\}^*$ we have that

$$\begin{aligned} g_{A,F}(k,z) &= \sum_{j\in J} a_j(k) f_j(k,z) \\ &\leq \sum_{j\in J} k^{c_A} k^{-(c+c_A+1)} \\ &= \sum_{j\in J} k^{-(c+1)} \\ &= |J| k^{-1} k^{-c} \\ &< k^{-c} \ , \end{aligned}$$

which proves the lemma with $k_c = \max\{|J|, k_A, k_{F,c+c_A+1}\}$. □

For two random variables X and Y over some finite domain D we let the statistical difference be

$$SD(X,Y) = \frac{1}{2} \sum_{d\in D} |\Pr[X = d] - \Pr[Y = d]| \ .$$

This is a number between 0 and 1.

A distribution ensemble is a family $X = \{X(k,z)\}_{k\in\mathbf{N}, z\in\{0,1\}^*}$ of random variables. We say that a distribution ensemble has domain D if all $X(k,z)$ have a domain which is included in D. If $D = \{0,1\}$ we call X a Boolean distribution ensemble.

Definition 2.2 *For two distribution ensembles X and Y with finite domain D we say that they are* computationally indistinguishable *(written $X \overset{c}{\approx} Y$) if $(k,z) \mapsto SD(X(k,z), Y(k,z))$ is negligible. We say that they are* statistically close *(written $X \overset{s}{\approx} Y$) if there exists $c \in \mathbf{R}$, where $c > 0$ and for all $k \in \mathbf{N}$ and all $z \in \{0,1\}^*$ we have that $SD(X(k,z), Y(k,z)) < 2^{-ck}$ for all $k > k_c$. We say that they are* identical *(written $X = Y$) if for all $k \in \mathbf{N}$ and all $z \in \{0,1\}^*$ we have that $SD(X(k,z), Y(k,z)) = 0$.*

It is a simply exercise to show that:

Fact 2.1 *If $X \overset{c}{\approx} Y$ and $Y \overset{c}{\approx} Z$, then $X \overset{c}{\approx} Z$. If $X \overset{s}{\approx} Y$ and $Y \overset{s}{\approx} Z$, then $X \overset{s}{\approx} Z$. If $X = Y$ and $Y = Z$, then $X = Z$. If $X = Y$, then $X \overset{s}{\approx} Y$, which in turn implies that $X \overset{c}{\approx} Y$.*

We use a constant like 0 to denote the distribution ensembles $\{0\}_{k \in \mathbf{N}, z \in \{0,1\}^*}$, so that we can write $X \overset{c}{\approx} 0$. For distribution ensembles $X_1, \ldots, X_a$ with domains $D_1, \ldots, D_a$ and a function $f : D_1 \times \cdots \times D_a \to D$ we define a distribution ensemble

$$f(X_1, \ldots, X_a) = \{f(X_1(k,z), \ldots, X_a(k,z))\}_{k \in \mathbf{N}, z \in \{0,1\}^*}$$

with domain D. If each X_i has a finite domain then $f(X_1, \ldots, X_a)$ has a finite domain. This e.g. allows us to write that if $X \overset{c}{\approx} 0$ and $Y \overset{c}{\approx} 0$, then $X + Y \overset{c}{\approx} 0$ (which is indeed true). We occasionally misuse this notation, but the meaning will hopefully be clear. The alternative is having to expand Definition 2.2 on the preceding page too often.

When working with computational indistinguishability we often use the following lemmas:

Lemma 2.2 *Let X and Y be two distribution ensembles with domain $\{0,1\}$. For $k \in \mathbf{N}$ and $z \in \{0,1\}^*$, let $d(k,z) = |\Pr[X(k,z) = 1] - \Pr[Y(k,z) = 1]|$. We have that $X \overset{c}{\approx} Y$ iff d is negligible.*

Proof. For $k \in \mathbf{N}$ and $z \in \{0,1\}^*$ we have that $SD(X(k,z), Y(k,z)) = \frac{1}{2} \sum_{i \in \{0,1\}} |\Pr[X(k,z) = i] - \Pr[Y(k,z) = i]| = \frac{1}{2}(|\Pr[X(k,z) = 0] - \Pr[Y(k,z) = 0]| + |\Pr[X(k,z) = 1] - \Pr[Y(k,z) = 1]|) = \frac{1}{2}(|(1 - \Pr[X(k,z) = 1]) - (1 - \Pr[Y(k,z) = 1])| + |\Pr[X(k,z) = 1] - \Pr[Y(k,z) = 1]|) = |\Pr[X(k,z) = 1] - \Pr[Y(k,z) = 1]| = d(k,z)$. □

Lemma 2.3 *Let X and Y be two distribution ensembles with finite domain D. For $d \in D$ define the function $I^{(d)} : D \to \{0,1\}$ by $I^{(d)}(d') = 1$ iff $d' = d$ and define the distribution ensembles $X^{(d)} = I^{(d)}(X)$ and $Y^{(d)} = I^{(d)}(Y)$. We have that $X \overset{c}{\approx} Y$ iff $\forall d \in D(X^{(d)} \overset{c}{\approx} Y^{(d)})$.*

Proof. We have that $X \overset{c}{\approx} Y$ iff $D(k,z) = \frac{1}{2} \sum_{d \in D} |\Pr[X = d] - \Pr[Y = d]|$ is negligible. By the above lemma and the definition of $X^{(d)}$ and $Y^{(d)}$ we have that $X^{(d)} \overset{c}{\approx} Y^{(d)}$ iff $D_d(x,z) = |\Pr[X = d] - \Pr[Y = d]|$ is negligible for all $d \in D$. Since $D(k,z) = \sum_{d \in D} \frac{1}{2} D_d(k,z)$ we have from Lemma 2.1 on the page before that $D(k,z)$ is negligible if all $D_d(k,z)$ are negligible. Furthermore, $D_d(k,z) \leq 2D(k,z)$, so by the same lemma, $D_d(k,z)$ is negligible if $D(k,z)$ is negligible. □

The notion of computational indistinguishability was introduced by Yao, Goldwasser and Micali in [Yao82b, GM84].

2.3 Function Families, Pseudorandom, One-Way, Trapdoor

The following definition of function families is inspired by similar definitions from [Gol01a, BDJR97].

Definition 2.3 *A function family is a family $F = \{F_k\}_{k\in\mathbf{N}}$ of random variables, so that the random variable F_k assumes values which are functions. We say that a function family is a PPT function family if the following two conditions hold:*

PPT indexing:
There exists a PPT algorithm, gen, called the index generator, and a mapping from strings to functions, ϕ, so that $\phi(gen(k))$ and F_k are identically distributed. By I we denote the possible outputs of gen and by f_i, for $i \in I$, we denote the function $\phi(i)$. For $i \in I$ we use D_i to denote the domain of f_i and we use E_i to denote the codomain.

PPT evaluation:
There exists a PPT algorithm F such that $F(i, x) = f_i(x)$ for $i \in I$ and $x \in D_i$.

We name some additional properties that a PPT function family might have:

PPT domain sampling:
We say that dom is a PPT domain sampler for F if dom is a PPT algorithm and $i \in I$ and $x \in dom(i)$ implies that $x \in D_i$.

PPT domain recognition:
We say that dom is a PPT domain recognizer for F if dom is a PPT algorithm and for $i \in I$ it holds that $dom(i, x) = 1$ iff $x \in D_i$.

Let F and G be two function families. We write $F \subseteq G$ if for all k, all functions in the support of F_k is also in the support of G_k.

Consider two families $A = \{A_k\}_{k\in\mathbf{N}}$ and $B = \{B_k\}_{k\in\mathbf{N}}$ of sets. If all B_k are finite, we use $[A \to B]$ to denote the function family $\{F_k\}_{k\in\mathbf{N}}$ where F_k is uniform over the set of all functions from A_k to B_k. We say that a function family F maps from A to B if $F \subseteq [A \to B]$.

For even moderately growing $|A|$ and $|B|$ a PPT function family F can only be a negligible subset of $[A \to B]$. If however a function drawn from F looks as a function drawn from $[A \to B]$ when one is only given the function in a box and can only compute in PPT, then we say that the function family is a pseudorandom function family from A to B.

Definition 2.4 *A PPT function family $F \subseteq [D \to E]$, where D can be recognized in PPT and E can be uniformly sampled in PPT, is said to be pseudorandom if for all PPT algorithm A it holds that $\mathrm{IND}^{prf,0}_{F,A} \overset{c}{\approx} \mathrm{IND}^{prf,1}_{F,A}$, where the random variable $\mathrm{IND}^{prf,b}_{F,A}(k, z)$ is defined in Fig. 2.1 on the following page and $\mathrm{IND}^{prf,b}_{F,A} = \{\mathrm{IND}^{prf,b}_{F,A}(k, z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$ is the corresponding Boolean distribution ensemble. If F is pseudorandom we call it a pseudorandom function family.*

A PPT function family is said to be one-way for a given distribution on the inputs if given an evaluation $y = f(x)$ on an element of the given distribution one cannot find x again, or any other preimage.

Definition 2.5 *A PPT function family F with PPT domain sampler dom being one-way is defined as follows: Let A be an algorithm, called the inverter, and let $z \in \{0, 1\}^*$ be an auxiliary input. Generate $i \leftarrow gen(k)$ and $x \leftarrow dom_i$ and compute $y = f_i(x)$ and*

Game $\mathrm{IND}^{\mathtt{prf},b}_{F,A}(k,z)$

The participants of the game is an adversary A and a PPT function family $F \subseteq [D \to E]$ where D can be recognized in PPT and E can be uniformly sampled in PPT. The input to the game is the security parameter $k \in \mathbf{N}$, a bit b and an auxiliary input $z \in \{0,1\}^*$. The game proceeds as follows.

1. We define a function $f : D_k \to E_k$ as follows:
   (a) If $b = 0$, then let f be a uniformly random function from D_k to E_k. The function f will be lazily defined to ensure that the experiment is running in PPT.
   (b) If $b = 1$, then generate $i \leftarrow gen(k)$ and let $f = f_i$.
2. Input (k, z) to A.
3. Run A to produce a value c.
   (a) If $c = (\mathtt{evaluate}, x)$, where $x \in D_k$, then compute $y = f(x)$, hand y to A and go to Step 3.
   (b) Otherwise interpret c as a bit d and output d.

Figure 2.1: The indistinguishability game for a family of PPT functions.

$x' \leftarrow A(k, z, i, y)$. *Output* 1 *if* $f_i(x') = f_i(x)$ *and otherwise output* 0. *Let* $\mathrm{INV}_{A,F}(k,z)$ *denote the distribution of the output. This defines a Boolean distribution ensemble* $\mathrm{INV}_{A,F} = \{\mathrm{INV}_{A,F}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$. *We call* (F, dom) *one-way if for all PPT algorithms* A *it hold that* $\mathrm{INV}_{A,F} \overset{\mathrm{c}}{\approx} 0$. *If* (F, dom) *is one-way we call it a* one-way function family.

Definition 2.6 *A* family of trapdoor permutations *is a quadruple* (gen, dom, f, f^{-1}) *of PPT algorithms, where*

- *gen, called the* index/trapdoor generator, *takes as input the security parameter* k *and outputs a pair* (i,t), *where* i *is called the* index *and* t *is called the* trapdoor. *In the following we let* gen^{i} *(respectively* gen^{t}*) denote the algorithm given by running* $(i,t) \leftarrow gen(k)$ *and outputting* i *(respectively* t*).*
- (gen^{i}, f) *is a one-way function family with domain sampler* dom.
- f^{-1}, *called the* inverse function, *takes as input* $t \in gen^{\mathrm{t}}$ *and* $y \in dom_i$ *and outputs a value* x. *We write* $x = f_t^{-1}(y)$.
- *For* $(i,t) \in gen$ *it holds that* $f_i : dom_i \to dom_i$ *and that* $f_t^{-1}(f_i(x)) = x$ *for* $x \in dom_i$.

The following definition and theorem will often come in handy when working with trapdoor permutations in protocols.

Definition 2.7 *Let* $F = (gen, dom, f, f^{-1})$ *be a family of trapdoor permutations. Let* A *be any PPT ITM and consider the following game, which we call* the trapdoor game. *The*

algorithm A is given $k \in \mathbf{N}$ and $z \in \{0,1\}^*$ and can ask for a number of index generations and element generations, and the goal of A is to invert a permutation for which it does not know the trapdoor information, on an element it did not generate itself.

- *On a index generation request, A is given i for a uniformly random key $(i,t) \leftarrow gen(k, r_i)$ (here r_i denotes the random bits used by gen).*
- *On a give up request on i, where i was generated in a index generation request, A is given r_i.*
- *On an element generation request for i, A receives $y = f_i(x)$, where x was generated as $x \leftarrow dom(i; r_y)$.*
- *On a give up request on y, where y was generated in an element generation request, A is given r_y.*
- *The ITM A wins the trapdoor game, if it manages to return an element x such that $y = f_i(x)$, where i is an index from a index generation request on which it has not given up and where y is from an element generation request on which it has not given up. The output of the trapdoor game is 1 when A wins the game and 0 otherwise.*

Let $\mathrm{TRAP}_{F,A}(k,z)$ *denote the distribution of the output of the trapdoor game. This defines a Boolean distribution ensemble* $\mathrm{TRAP}_{F,A} = \{\mathrm{TRAP}_{F,A}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Theorem 2.1 *If the family $F = (gen, dom, f, f^{-1})$ is a family of trapdoor permutations, then for all PPT algorithms A it holds that* $\mathrm{TRAP}_{F,A} \overset{\mathrm{c}}{\approx} 0$.

Proof. Assume that we are given a PPT adversary A for the trapdoor game. It has some running time k^c and will therefore make at most k^c index generation requests and at most k^c element generation requests. Consider the following adversary $B(A)$ for the inverting game. It receives (k, z, i, y). Then it picks two uniformly random numbers a and b from $\mathbf{Z}_{k^c}$. Then it it runs $\mathrm{INV}_{A,F}(k,z)$ with the only difference that on the a'th index generation request it gives i to A and on the b'th element generation request for i it gives y to A. If A does not give up on i and does not give up on y and A outputs x such that $y = f_i(x)$, then $B(A)$ outputs x. Otherwise $B(A)$ outputs ϵ. It is straight forward to verify that $\Pr[\mathrm{TRAP}_{A,F}(k,z) = 1] \geq \left(\frac{1}{k^c}\right)^2 \mathrm{INV}_{B(A),F}(k,z)$, which proves the theorem. □

2.4 Public-Key Encryption Schemes

In this section we formalize the notion of a public-key encryption scheme and the notion of indistinguishability under chosen-plaintext attack (IND-CPA). These notions were first formalized by Goldwasser and Micali in [GM84]. As usual a public-key encryption scheme consist of a key generator *gen*, an encryption function *enc* and a decryption function *dec*.

Game $\mathrm{IND}^{\mathtt{cpa},b}_{\mathcal{E},A}(k,z)$

The input to the game is a public-key encryption scheme $\mathcal{E} = (gen, enc, dec)$, an ITM A, the security parameter k, a bit b and an auxiliary input z. The game proceeds as follows:

1. Run $(e,d) \leftarrow gen(k)$ and input (k,z,e) to A.
2. Receive an output from A and interpret it as a pair (m_0,m_1), where $m_0, m_1 \in \mathcal{M}_e$.
3. Generate $c = enc(e,m_b)$ and input c to A.
4. Receive an output from A and interpret it as a bit d.
5. Output d.

Figure 2.2: The CPA indistinguishability game.

Definition 2.8 *A public-key encryption scheme consists of three PPT algorithms $\mathcal{E} = (gen, enc, dec)$. On input k the key generator gen computes $(e,d) \leftarrow gen(k)$, where e is the encryption key or public key and d is the decryption key or private key. The encryption key e defines a plaintext space $\mathcal{M}_e$, which can be recognized in PPT. The encryption function enc takes as input an encryption key e and a plaintext $m \in \mathcal{M}_e$ and a randomizer $r \in \{0,1\}^*$ and outputs a ciphertext $c \in \{0,1\}^*$; We write $c \leftarrow enc(e,m;r)$. The decryption function dec takes as input a decryption key d and $c \in \{0,1\}^*$ and outputs m, where m is the plaintext of c. We require that for all $(e,d) \in gen$ and $m \in \mathcal{M}_e$ it holds that if $c \in enc(e,m)$ and $m' \in dec(d,c)$, then $m' = m$; We call this perfect correctness.*

Definition 2.9 *Let $\mathcal{E}$ be a public-key encryption scheme. The IND-CPA security of $\mathcal{E}$ is defined via the game in Fig. 2.2, as follows: Let A be any ITM and let $\mathrm{IND}^{\mathtt{cpa},b}_{\mathcal{E},A}(k,z)$ denote the random variable describing the output of the game in Fig. 2.2 when the random bits used by A and the random bits used by gen and enc are chosen uniformly at random. This defines a Boolean distribution ensemble $\mathrm{IND}^{\mathtt{cpa},b}_{\mathcal{E},A} = \{\mathrm{IND}^{\mathtt{cpa},b}_{\mathcal{E},A}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$. We say that $\mathcal{E}$ is IND-CPA secure if $\mathrm{IND}^{\mathtt{cpa},0}_{\mathcal{E},A} \overset{\mathrm{c}}{\approx} \mathrm{IND}^{\mathtt{cpa},1}_{\mathcal{E},A}$ for all PPT A.*

The following theorem sometimes comes in handy when using IND-CPA security.

Theorem 2.2 *Let $\mathcal{E}$ be a public-key encryption scheme and let A be any ITM and let $\mathrm{RIND}^{\mathtt{cpa},b}_{\mathcal{E},A}(k,z)$ denote the random variable describing the output of the game in Fig. 2.3 on the next page when the random bits used by A and the random bits used by gen and enc are chosen uniformly at random. This defines a Boolean distribution ensemble $\mathrm{RIND}^{\mathtt{cpa},b}_{\mathcal{E},A} = \{\mathrm{RIND}^{\mathtt{cpa},b}_{\mathcal{E},A}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$. If $\mathcal{E}$ is IND-CPA secure, then $\mathrm{RIND}^{\mathtt{cpa},0}_{\mathcal{E},A} \overset{\mathrm{c}}{\approx} \mathrm{RIND}^{\mathtt{cpa},1}_{\mathcal{E},A}$ for all PPT A.*

Proof. Assume that we are given an adversary A for the repeated IND-CPA game. For $k \in \mathbf{N}$, $z \in \{0,1\}^*$ and $b \in \{0,1\}$, let $P^b(k,z) = \Pr[\mathrm{RIND}^{\mathtt{cpa},b}_{\mathcal{E},A}(k,z) = 1]$ and let $Q(k,z) = |P^0(k,z) - P^1(k,z)|$. By the definition of repeated IND-CPA security it is enough to prove that Q is negligible. To prove this we construct an adversary $B(A)$ for the IND-CPA game. Since A is

Game RIND-CPA

The input to the game is a public-key encryption scheme $\mathcal{E} = (gen, enc, dec)$, an ITM A, the security parameter k, a bit b and an auxiliary input z. The game proceeds as follows:

1. Input (k, z) to A.
2. Run A to receive an output v and proceed as follows:
   (a) If $v = (\texttt{generate})$, then run $(e, d) \leftarrow gen(k)$, store e and input e to A.
   (b) If $v = (\texttt{test-messages}, e, m_0, m_1)$, where e has been stored and $m_0, m_1 \in \mathcal{M}_e$, then proceed as follows: Compute $c = enc(e, m_b)$ and input c to A.
   (c) If $v = (\texttt{guess}, d)$ for $d \in \{0, 1\}$, then output d and halt.

   Go to Step 2.

Figure 2.3: The repeated CPA indistinguishability game.

PPT is has a running time k^c. This means that it makes at most k^c `generate`-requests and at most k^c `test-messages`-requests. The adversary $B(A)$ runs in $\mathrm{IND}^{\texttt{cpa},b}_{\mathcal{E},B(A)}(k, z)$. Initially it is given (k, z, e). Then it generates two uniformly random integers $r, s \in \mathbf{Z}_{k^c}$ and runs a variant of the repeated IND-CPA game as follows:

1. Input (k, z) to A.
2. Let $R = 0$ and let $S = 0$.
3. Run A to receive an output v and proceed as follows:
   (a) If $v = (\texttt{generate})$, then define a key e_R as follows:
      - If $R \neq r$, then run $(e', d') \leftarrow gen(k)$ and let $e_R = e'$.
      - If $R = r$, then let $e_R = e$, where e is the key initially received by $B(A)$ from $\mathrm{IND}^{\texttt{cpa},b}_{\mathcal{E},B(A)}(k, z)$.

      Store e_R, give e_R to A and let $R \leftarrow R + 1$.
   (b) If $v = (\texttt{test-messages}, e_\rho, m_0, m_1)$, where e_ρ has been stored and $m_0, m_1 \in \mathcal{M}_{e_\rho}$, then a ciphertext c is defined as follows:
      - If $\rho < r$, then compute $c = enc(e_\rho, m_1)$.
      - If $\rho = r$, then
        - If $S < s$, then compute $c = enc(e_\rho, m_1)$.
        - If $S = s$, then output $(\texttt{test-messages}, m_0, m_1)$ to the game $\mathrm{IND}^{\texttt{cpa},b}_{\mathcal{E},B(A)}(k, z)$ and receive $c' = enc(e, m_b) = enc(e_\rho, m_b)$. Then let $c = c'$.
        - If $S > s$, then compute $c = enc(e_\rho, m_0)$.
      - If $\rho > r$, then compute $c = enc(e_\rho, m_0)$.
   (c) If $v = (\texttt{guess}, d)$, then output $(\texttt{guess}, d)$ and halt.

For $b \in \{0,1\}$, let $p^b(k,z) = \Pr[\mathrm{IND}^{\mathtt{cpa},b}_{\mathcal{E},B(A)}(k,z) = 1]$ and let $q(k,z) = |p^0(k,z) - p^1(k,z)|$. By the IND-CPA security of $\mathcal{E}$ we have that q is negligible. To exploit this, we relate $q(k,z)$ to $Q(k,z)$. For $r, s \in \mathbf{Z}_{k^c}$ and $b \in \{0,1\}$, let $p^{r,s,b}(k,z) = \Pr[\mathrm{IND}^{\mathtt{cpa},b}_{\mathcal{E},B(A)}(k,z) = 1]$ when $B(A)$ is using the fixed values r and s. It is straight forward to verify that $p^{r,s,1}(k,z) = p^{r,s+1,0}(k,z)$ for $r \in k^c - 1$ and $s \in k^c - 2$. This gives us that

$$\begin{aligned}
p^{r,0,0}(k,z) - p^{r,k^c-1,1}(k,z) &= \left(\sum_{s=0}^{k^c-2} \left(p^{r,s,0}(k,z) - p^{r,s+1,0}(k,z)\right) + p^{r,k^c-1,0}(k,z)\right) \\
&\qquad - p^{r,k^c-1,1}(k,z) \\
&= \sum_{s=0}^{k^c-2} \left(p^{r,s,0}(k,z) - p^{r,s,1}(k,z)\right) \\
&\qquad + \left(p^{r,k^c-1,0}(k,z) - p^{r,k^c-1,1}(k,z)\right) \\
&= \sum_{s=0}^{k^c-1} \left(p^{r,s,0}(k,z) - p^{r,s,1}(k,z)\right) \ .
\end{aligned} \tag{2.1}$$

It is also clear that $p^{r,k^c-1,1}(k,z) = p^{r+1,0,0}(k,z)$. This gives us that

$$\begin{aligned}
p^{0,0,0}(k,z) - p^{k^c-1,k^c-1,1}(k,z) &= \left(\sum_{r=0}^{k^c-2} \left(p^{r,0,0}(k,z) - p^{r+1,0,0}(k,z)\right) + p^{k^c-1,0,0}(k,z)\right) \\
&\qquad - p^{k^c-1,k^c-1,1}(k,z) \\
&= \sum_{r=0}^{k^c-2} \left(p^{r,0,0}(k,z) - p^{r,k^c-1,1}(k,z)\right) \\
&\qquad + \left(p^{k^c-1,0,0}(k,z) - p^{k^c-1,k^c-1,1}(k,z)\right) \\
&= \sum_{r=0}^{k^c-1} \left(p^{r,0,0}(k,z) - p^{r,k^c-1,1}(k,z)\right) \\
&= \sum_{r=0}^{k^c-1} \sum_{s=0}^{k^c-1} \left(p^{r,s,0}(k,z) - p^{r,s,1}(k,z)\right) \\
&= \sum_{r=0,s=0}^{k^c-1,k^c-1} p^{r,s,0}(k,z) - \sum_{r=0,s=0}^{k^c-1,k^c-1} p^{r,s,1}(k,z) \\
&= k^{2c}\left(p^0(k,z) - p^1(k,z)\right) \ ,
\end{aligned} \tag{2.2}$$

where the third to last equation follows from Eq. 2.1 and the last equation follows from the fact that r and s are uniformly chosen from $\mathbf{Z}_{k^c}$ so that $p^b(k,z) = k^{-2c} \sum_{r=0,s=0}^{k^c-1,k^c-1} p^{r,s,b}(k,z)$, for $b \in \{0,1\}$. Now observe that $P^0(k,z) = p^{0,0,0}(k,z)$ and $P^1(k,z) = p^{k^c-1,k^c-1,1}(k,z)$. By Eq. 2.2 this gives us that $Q(k,z) = k^{2c} q(k,z)$. Since q is negligible it follows from Lemma 2.1 on page 11 that Q is negligible. □

Game $\mathrm{FORGE}_{\mathcal{S},A}(k,z)$

The input to the game is a digital signature scheme $\mathcal{S} = (gen, sig, ver)$, an ITM A, the security parameter k and an auxiliary input z. The game proceeds as follows:

1. Generate a key-pair $(s,v) \leftarrow gen(k)$ and give (k,z,v) to A.
2. Run A to generate a command c.
   (a) If $c = (\texttt{sign}, m)$ for $m \in \{0,1\}^*$, then compute $\sigma \leftarrow sig(s,m)$, return σ to A and go to Step 2.
   (b) If $c = (\texttt{forgery}, m, \sigma)$ for $m, \sigma \in \{0,1\}^*$, then the game ends with output $b \in \{0,1\}$, where $b = 1$ iff m never occurred in a $(\texttt{sign}, m)$ command and $ver(v,\sigma,m) = 1$.

Figure 2.4: The forging game.

2.5 Digital Signatures Schemes

Definition 2.10 *A digital signature scheme is a triple $\mathcal{S} = (gen, sig, ver)$ of PPT algorithms, where:*

- *gen is the key generation algorithm which on input the security parameter k outputs a pair (s,v), where s is called the signing key and v is called the verification key. The verification key v defines a set $\mathcal{M}_v$ of signable messages.*
- *sig is the signing algorithm which takes as input a signing key s and a message $m \in \{0,1\}^*$ and outputs a signature σ. We write $\sigma \leftarrow sig(s,m)$.*
- *ver is the verification algorithm which takes as input a verification key $v \in \{0,1\}^*$, a message $m \in \{0,1\}^*$ and a signature $s \in \{0,1\}^*$ and outputs a judgment of validity $j \in \{0,1\}$. We write $j \leftarrow ver(v,\sigma,m)$.*

To call a triple $\mathcal{S}$ a signature scheme we require that:

completeness:
If $(s,v) \in gen(k)$ and $m \in \mathcal{M}_v$ and $\sigma \in sig(s,m)$, then $\Pr[ver(v,m,\sigma) = 1] = 1$.

universal verifiability:
For all $v, m, \sigma \in \{0,1\}^*$ it holds that $\Pr[ver(v,m,\sigma) = 1] \in \{0,1\}$.

The completeness says that a correctly computed signature on a signable message is always accepted, whereas the universal verifiability say that no matter how v and m was generated, the judgment $ver(v,m,\sigma)$ is consistent for all σ. While universal verifiability is essential for many applications of signature scheme, it is a requirement that is not always stated explicitly in definitions of signature scheme.

In [GMR88] a definition of security for digital signature schemes is given. Intuitively the notion says that attacker which sees signatures on messages of its choice cannot afterward produce a new signature on any message. A slightly relaxed notion can be considered, where

the attacker only wins if it produces a signature on a new message, i.e. a message that was not previously signed. In particular, 'mauling' a signature σ on m into a different signature σ' on m is not consider an attack. This is the notion we adapt.

Definition 2.11 *Let $\mathcal{S} = (gen, sig, ver)$ be a digital signature scheme. Let A be any PPT ITM and consider the forging game in Fig. 2.4 on the page before. This defines a random variable* $\mathrm{FORGE}_{\mathcal{S},A}(k,z)$ *and a Boolean distribution ensemble* $\mathrm{FORGE}_{\mathcal{S},A} = \{\mathrm{FORGE}_{\mathcal{S},A}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$. *If it holds for all PPT ITM's A that* $\mathrm{FORGE}_{\mathcal{S},A} \overset{c}{\approx} 0$, *then we say that $\mathcal{S}$ is existentially unforgeable under chosen-message attack.*

2.6 Commitment Schemes

In a commitment scheme, a committer commits to an element m, from some finite set M, by releasing some information about m through a commit protocol to a receiver. Later, the committer may release more information to the receiver to open his commitment, so that the receiver learns m. Loosely speaking, the basic properties we want are first that the commitment scheme is hiding, i.e. a cheating receiver cannot learn m from the commitment protocol, and second that it is binding, i.e. a cheating committer cannot change his mind about m, the verifier can check in the opening that the value opened was what the committer had in mind originally. Each of the two properties can be satisfied unconditionally or relative to a complexity assumption. In the most basic commitment protocols both committing and opening is non-interactive. To commit the committer sends $c = \mathrm{commit}_K(m;r)$ to the receiver, where K is a commitment key and r is a randomizer used for hiding m. Opening then typically means that the committer sends (m,r) to the receiver who checks that $c = \mathrm{commit}_K(m;r)$. The scheme should then have the property that from c one cannot compute m and that the committer cannot compute two openings (m_0,r_0) and (m_1,r_1) such that $m_0 \neq m_1$ and $\mathrm{commit}_K(m_0;r_0) = \mathrm{commit}_K(m_1;r_1)$. For a special flavor of commitment scheme, called trapdoor commitment scheme, the scheme has the property that for each commitment key K there exists a special piece of information t, called the trapdoor, such that given K and t and any commitment c and any message m, one can efficiently compute r such that $c = \mathrm{commit}_K(m;r)$. I.e. the commitment scheme is only binding for a committer that do not know t. We will primarily be interested in trapdoor commitment schemes. Trapdoor commitment schemes have also been called equivocable commitment schemes [Bea96]. Notice that a trapdoor commitment scheme can never be perfect binding. Therefore the best we can hope for is perfect hiding and computational binding. This is therefore the definition we adapt.

Definition 2.12 *A trapdoor commitment scheme consists of two PPT algorithms* (gen, commit). *On input k the key/trapdoor generator gen outputs a commitment key K and a trapdoor t; We write $(K,t) \leftarrow gen(k)$. We let the key space be the PPT family of sets $\mathcal{K} = \{\mathcal{K}(k)\}_{k\in\mathbf{N}}$, where $\mathcal{K}(k) = \{K | \exists r, t \in \{0,1\}^*((K,t) = gen(k;r)\}$. A key $K \in \mathcal{K}$ defines a set of commitments $\mathcal{C}_K$ and a set of messages $\mathcal{M}_K$, both can be recognized in PPT. The commitment function* commit *takes as input a commitment key $K \in \mathcal{K}$, a message $m \in \mathcal{M}_K$ and a randomizer $r \in \{0,1\}^*$ and outputs a commitment $c \in \mathcal{C}_K$. We write $c \leftarrow \mathrm{commit}_K(m;r)$. We require the following:*

perfect (statistical) hiding:

For all $K \in \mathcal{K}$ and all $m_0, m_1 \in \mathcal{M}_K$ the distributions of $\mathrm{commit}_K(m_0)$ and $\mathrm{commit}_K(m_1)$ are identical (statistically close) when the randomizer is uniformly random.

computational binding:

Let A be any adversary and consider the following game called the double opening game. The input is $k \in \mathbf{N}$ and and an auxiliary string $z \in \{0,1\}^*$. Run $(K,t) \leftarrow gen(k)$ and given (k,z,K) to A. Then run A to produce an output (m_0, r_0, m_1, r_1). If $m_0, m_1 \in \mathcal{M}_K$ and $m_0 \neq m_1$ and $\mathrm{commit}_K(m_0; r_0) = \mathrm{commit}_K(m_1; r_1)$, then the output of the game is 1. Otherwise the output of the game is 0. This defines a random variable $\mathrm{DOUBLE}_{(gen,\mathrm{commit}),A}(k,z)$ and a Boolean distribution ensemble $\mathrm{DOUBLE}_{(gen,\mathrm{commit}),A} = \{\mathrm{DOUBLE}_{(gen,\mathrm{commit}),A}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$. We require that $\mathrm{DOUBLE}_{(gen,\mathrm{commit}),A} \overset{\mathrm{c}}{\approx} 0$ for all PPT adversaries A.

trapdoor opening:

Given $(K,t) \in gen$ and $c \in \mathcal{C}_K$ and $m \in \mathcal{M}_K$ it is possible to compute $r \in \{0,1\}^*$ such that r is uniformly random among the r for which $c = \mathrm{commit}_K(m; r)$.

One can construct efficient[1] trapdoor commitment schemes based on all the complexity theoretic assumptions that we use in the remainder. See e.g. the Ph.D. thesis of Fischlin [Fis01].

If the generator does not output a trapdoor t and the scheme has all the properties from Definition 2.12 on the facing page except trapdoor opening, we call the scheme a (statistically) perfect hiding commitment scheme.

2.7 Some Complexity Theoretic Assumptions

In this section we introduce the complexity theoretic assumptions on which we are going to base the security of our protocols.

2.7.1 Diffie-Hellman

In [DH76] the decisional Diffie-Hellman assumption (DDH assumption) is implicitly made, and not to forget, public-key cryptography introduced. The DDH assumption was first formalized in [Bra93] by Brands. We give a formalization along the lines of Brands. Let $\langle\alpha\rangle$ be a cyclic group. The Diffie-Hellman assumption in $\langle\alpha\rangle$ basically states that elements of the form $(\alpha^x, \alpha^y, \alpha^z)$ for uniformly random $x, y, z \in \mathbf{Z}_{\mathrm{ord}(\alpha)}$ cannot be distinguished from uniformly random elements of the form $(\alpha^x, \alpha^y, \alpha^{xy})$ for uniformly random $x, y \in \mathbf{Z}_{\mathrm{ord}(\alpha)}$.

Assumption 2.1 *Let $(\cdot, \alpha) \leftarrow gen(k)$ be a generator generating a description of a group operation $\cdot$ and an element α of finite order $o = |\langle\alpha\rangle|$. Let A be any algorithm, let $b \in \{0,1\}$, let $z \in \{0,1\}^*$ and consider the following game $\mathrm{DDH}_{A,gen}(k,z)$. Run $(\cdot, \alpha) \leftarrow gen(k)$ and generate uniformly random $X, Y \in \mathbf{Z}_o$, for $o = |\langle\alpha\rangle|$. Let $Z = XY$ if $b = 1$ and generate uniformly random $Z \in \mathbf{Z}_o$ otherwise. Input $(k, z, \cdot, \alpha, \alpha^X, \alpha^Y, \alpha^Z)$ to A and receive a bit c. Output c.*

[1] By which we mean committing to k bits using $O(k)$ bits.

This defines a Boolean distribution ensemble $\mathrm{DDH}^b_{gen,A} = \{DDH^b_{gen,A}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$. *The DDH assumption for gen is that* $\mathrm{DDH}^0_{gen,A} \stackrel{c}{\approx} \mathrm{DDH}^1_{gen,A}$.

We will primarily use the DDH assumption in the following group: Let p be a random k-bit prime p for which $q = (p-1)/2$ is also prime. In the following, we let Q_p denote the quadratic residues modulo p and we let $g = 4$. It can be verified using elementary algebra that $\langle g\rangle_{\mathbf{Z}_p^*} = Q_p$, $|Q_p| = q$ and $\mathbf{Z}_p^* = Q_p \cup (-1)Q_p$.

2.7.2 The Paillier Cryptosystem

In this section we describe the Paillier cryptosystem [Pai99]. Throughout this section, let p and q be random $\lceil k/2\rceil$-bit primes for which $p,q > 2$, $p \neq q$ and $\gcd(pq,(p-1)(q-1)) = 1$. Let $n = pq$ and let $\lambda = \mathrm{lcm}(p-1,q-1)$.

Definition 2.13 *By the Paillier group isomorphism we mean the function*

$$\Psi : \mathbf{Z}_n \times \mathbf{Z}_n^* \to \mathbf{Z}_{n^2}^* \ ,$$

$$\Psi(x,r) = g^x r^n \bmod n^2 \ ,$$

where $g = (n+1)$ *and* $r \in \mathbf{Z}_n^*$ *is considered as an element of* $\mathbf{Z}_{n^2}^*$ *using the trivial observation that for* $r \in \{1,\ldots,n-1\}$ *it holds that if* $\gcd(r,n) = 1$, *then* $\gcd(r,n^2) = 1$.

Lemma 2.4 *The Paillier group isomorphism* Ψ_n *is an isomorphism of groups which can be inverted in PPT given* n *and* λ.

Proof. We first prove that Ψ_n is a homomorphism of groups. Let $c_1 = g^{x_1} r_1^n \bmod n^2$ and let $c_2 = g^{x_2} r_2^n \bmod n^2$, for $x_1, x_2 \in \mathbf{Z}_n$ and $r_1, r_2 \in \mathbf{Z}_n^*$. We want to prove that

$$c_1 c_2 \bmod n^2 = g^{x_1 + x_2 \bmod n}(r_1 r_2 \bmod n)^n \bmod n^2 \ . \tag{2.3}$$

Use a binomial expansion to see that $(r+n)^x \bmod n^2 = r^x + xr^{x-1}n \bmod n^2$, which in particular implies that $g = (1+n)$ has order n in $\mathbf{Z}_{n^2}^*$ and that $(r+n)^n \bmod n^2 = r^n \bmod n^2$ for $r \in \mathbf{Z}_{n^2}^*$, which immediately implies Eq. 2.3. To see that Ψ_n is a bijection, we show how to invert it. Let

$$c = g^x r^n \bmod n^2$$

for $x \in \mathbf{Z}_n$ and $r \in \mathbf{Z}_n^*$. Let $d = \lambda(\lambda^{-1} \bmod n)$ and let $e = n^{-1} \bmod \lambda$;[2] Notice that d and e can be computed from n and λ in polynomial time. Since the order of all elements of $\mathbf{Z}_{n^2}^*$ divides $n\lambda$ it follows that the order of $r^n \bmod n^2$ divides λ. Therefore $(r^n \bmod n^2)^d \bmod n^2 = 1$. Since g has order n in $\mathbf{Z}_{n^2}^*$ it follows that $g^d \bmod n^2 = g$. Therefore $c^d \bmod n = g^x \bmod n^2$. Since $g^x \bmod n^2 = 1 + xn$, by Eq. 2.3, it follows that

$$x = \frac{(c^d \bmod n^2) - 1}{n} \ .$$

[2]Here we used the condition on p and q that $\gcd(pq,(p-1)(q-1)) = 1$.

Since $g \bmod n = 1$ it follows that $c \bmod n = r^n \bmod n$. Since $(r \bmod n) \in \mathbf{Z}_n^*$ it follows that the order of $(r \bmod n)$ in $\mathbf{Z}_n^*$ divides λ. Therefore

$$r = c^e \bmod n \ .$$

□

The so-called decisional composite residuosity assumption (DCRA) is that random elements of the form $\Psi(0, r)$ cannot be distinguished from uniformly random elements from $\mathbf{Z}_{n^2}^*$.

Assumption 2.2 *Let gen denote the generator $(p, q) \leftarrow gen(k)$ described above. Let A be any adversary. Consider the following game* $\text{PAILLIER}_{gen,A}^b(k, z)$. *Generate $(p, q) \leftarrow gen(k)$ and let $n = pq$. If $b = 0$, then generate uniformly random $r \in \mathbf{Z}_{pq}^*$ and let $r_b = r^n \bmod n^2$. If $b = 1$, then generate r_b as a uniformly random element from $\mathbf{Z}_{n^2}^*$. Then input (k, z, n, r_b) to A, and run A to obtain a value $c \in \{0, 1\}$. Then output c. The decisional composite residuosity assumption is that* $\text{PAILLIER}_{gen,A}^0 \stackrel{c}{\approx} \text{PAILLIER}_{gen,A}^1$ *for all PPT algorithms A.*

It is straight-forward to see that under the DCRA one can IND-CPA securely encrypt $m \in \mathbf{Z}_n$ as $c = E_n(m; r) = \Phi_n(m; r)$ for a uniformly random $r \in \mathbf{Z}_n^*$. The private key is e.g. λ. This will also be a corollary of a result in Section 2.10. We will be using this cryptosystem a lot.

2.7.3 The Okamoto-Uchiyama Cryptosystem

In this section we describe the cryptosystem by Okamoto and Uchiyama [OU98]. The presentation will be somewhat different from the one in [OU98] as we exploit that we have just presented the Paillier cryptosystem. Notice that, as opposed our order of presentation, [OU98] is prior to [Pai99].

Throughout this section, let p and q be random $\lceil k/2 \rceil$-bit primes for which $p, q > 2$, $p \neq q$ and $\gcd(pq, (p-1)(q-1)) = 1$. Let $n = pq$, let $\lambda = \text{lcm}(p-1, q-1)$ and $N = pn$.

Definition 2.14 *By the Okamoto-Uchiyama group homomorphism we mean the function*

$$\Omega : \mathbf{Z}_p \times \mathbf{Z}_N^* \to \mathbf{Z}_N^* \ ,$$

$$\Omega(x, r) = g^x r^N \bmod N \ ,$$

where $g = (n + 1)$ and $r \in \mathbf{Z}_N^*$.

Lemma 2.5 *The Okamoto-Uchiyama group homomorphism Ω is a homomorphism of groups and a pre-image from $\mathbf{Z}_q \times \mathbf{Z}_n^*$ can be found in PPT given N and λ.*

Proof. We first prove that Ω is a homomorphism of groups. Let $c_1 = g^{x_1} r_1^N \bmod N$ and let $c_2 = g^{x_2} r_2^N \bmod N$, for $x_1, x_2 \in \mathbf{Z}_p$ and $r_1, r_2 \in \mathbf{Z}_N^*$. We want to prove that

$$c_1 c_2 \bmod N = g^{x_1 + x_2 \bmod p} (r_1 r_2 \bmod N)^N \bmod N \ . \tag{2.4}$$

Use a binomial expansion and the equation $n^2 \bmod N = 0$ to see that $(r+n)^x \bmod N = r^x + xr^{x-1}n \bmod N$, which in particular implies that $g = (1+n)$ has order p in $\mathbf{Z}_N^*$ and that $(r+n)^N \bmod N = r^N \bmod N$ for $r \in \mathbf{Z}_N^*$, which immediately implies Eq. 2.4 on the page before. We show how to find a preimage. Let

$$c = g^x r^N \bmod N$$

for $x \in \mathbf{Z}_N$ and $r \in \mathbf{Z}_N^*$. Let $d = \lambda(\lambda^{-1} \bmod p)$ and let $e = N^{-1} \bmod \lambda$; Notice that d and e can be computed from n and λ in polynomial time. Since the orders of all elements of $\mathbf{Z}_N^*$ divides $p\lambda$ it follows that the order of $r^N \bmod N$ divides λ. Therefore $(r^N \bmod N)^d \bmod pn = 1$. Since g has order p in $\mathbf{Z}_N^*$ it follows that $g^d \bmod N = g$. Therefore $c^d \bmod N = g^x \bmod N$. Since $g^x \bmod N = 1 + xn \bmod N$ it follows that

$$x = \frac{(c^d \bmod N) - 1}{n} \ .$$

Since $g \bmod n = 1$ it follows that $c \bmod n = r^N \bmod n$. Since $(r \bmod n) \in \mathbf{Z}_n^*$ it follows that the order of $(r \bmod n)$ in $\mathbf{Z}_n^*$ divides λ. Therefore

$$r \bmod n = c^e \bmod n \ .$$

Let $r' = c^e \bmod n$. Above we established that $(r+n)^N \bmod N = r^N \bmod N$, which implies that $r'^N \bmod N = (r \bmod n)^N \bmod N = r^N \bmod N$. All in all $x' \in \mathbf{Z}_p$, $r' \in \mathbf{Z}_n^*$ and $c = g^{x'} r'^N \bmod N$. □

In [OU98] the p-subgroup assumption is made, which is that random elements of the form $\Omega(0, r)$ cannot be distinguished from random elements from $\mathbf{Z}_N^*$. This assumption corresponds to the DCRA in $\mathbf{Z}_{(pq)^2}$.

Assumption 2.3 *Let gen denote the generator $(p,q) \leftarrow gen(k)$ described above. Let A be any adversary. Consider the following game* $\text{OKAMUCHI}^b_{gen,A}(k,z)$. *Generate $(p,q) \leftarrow gen(k)$ and let $N = p^2 q$. Then generate uniformly random $r_0, r_2 \in \mathbf{Z}_N^*$ and compute $r_1 \leftarrow r_2^N \bmod N$. Then input (k, z, N, r_b) to A, and run A to obtain a value $c \in \{0,1\}$. Then output c. The p-subgroup assumption is that* $\text{OKAMUCHI}^0_{gen,A} \stackrel{c}{\approx} \text{OKAMUCHI}^1_{gen,A}$ *for all PPT algorithms A.*

In Section 2.10 we show how to build an IND-CPA secure encryption scheme from the Okamoto-Uchiyama homomorphism and the p-subgroup assumption.

2.7.4 RSA

In this section we formalize the RSA, the strong RSA assumptions and the RSA-sub-group assumption. The RSA assumption was first suggested in [RSA78] by Rivest, Shamir and Adleman. The RSA-sub-group assumption was first suggested in [DN00] by Damgård and Nielsen. Let an RSA generator be a PPT generator *gen* which on input k generates (p,q) such that p and q are primes.

Assumption 2.4 *Let gen be an RSA generator, which in addition generates a value* $e \in \mathbf{Z}^*_{(p-1)(q-1)}$*, and let A be any adversary. Consider the following game* $\mathrm{RSA}_{gen,A}(k,z)$*. We generate* $(p,q,e) \leftarrow gen(k)$ *and generate uniformly random* $y \in \mathbf{Z}^*_{pq}$*. Then input* (k,z,pq,e,y) *to A, and run A to obtain a value x. If* $y = x^e \bmod pq$*, then output* 1*; Otherwise, output* 0*. We say that the RSA assumption holds for gen if* $\mathrm{RSA}_{gen,A} \overset{c}{\approx} 0$ *for all PPT algorithms A.*

Assumption 2.5 *Let gen be an RSA generator and let A be any algorithm. Consider the following game* $\mathrm{SRSA}_{gen,A}(k,z)$*. We generate* $(p,q) \leftarrow gen(k)$ *and generate uniformly random* $x \in \mathbf{Z}^*_{pq}$*. Then input* (k,z,pq,y) *to A. Then run A to obtain a value* (x,e)*. If* $e > 1$ *and* $y = x^e \bmod pq$*, then output* 1*; Otherwise output* 0*. We say that the strong RSA assumption holds for gen if* $\mathrm{SRSA}_{gen,A} \overset{c}{\approx} 0$ *for all PPT algorithms A.*

We then formalize the RSA-sub-group assumption. Consider the following RSA generator: It generates random numbers N until it finds n which has two large primefactors[3] and then it outputs (p,q). Furthermore, it picks a large random prime $e > N = pq$. Since e is prime and $e > n$ it follows that $e \in \mathbf{Z}^*_{(p-1)(q-1)}$ as in Assumption 2.4. The RSA-sub-group assumption is basically that the RSA assumption hold for this generator, though we will work in the group $\mathbf{Z}^*_n$, which has the RSA group $\mathbf{Z}^*_N$ as a sub-group.

Definition 2.15 *We define a family which we call the RSA-sub-group collection:*

index/trapdoor generation:

An index will be a uniformly random $k\log(k)$*-bit number for which* $p,q > 2^{k-1}$ *for the two largest primefactors p and q of n. The trapdoor will be* $\phi(n)$*.*

Furthermore a random element $e \in \mathrm{PRIMES}(2k\log(k))$ *is chosen. The encryption key is* (n,e) *and the decryption key is* $(n, d = e^{-1} \bmod \phi(n))$*.*

In [Bac88] Bach shows how to sample in PPT a random number with known factorization, which immediately yields a PPT algorithm for the key generation: Sample random $k\log(k)$*-bit integers with known factorization until one is found with* $p,q > 2^{k-1}$*. To make the algorithm PPT, give up after k attempts. In [KT76] Knuth and Trabb investigate the probability that the i'th largest primefactor of a random number n is larger than* n^c *for a given constant c. It is shown to approach a constant as n approaches infinity. In particular, the probability that the second largest primefactor is smaller than* n^c *is approximately linear for small c, in fact it is about* $2c$ *for* $c \leq 0.4$*. It follows that the probability that a uniformly random* $2k\log(k)$*-bit integer has its second largest primefactor shorter than k bits is* $O(1/\log(k))$*. Therefore such a number is found after k trials except with negligible probability.*

domain sampling:

The domain of a public key (n,e) *will be* $\mathbf{Z}^*_n$*. It is described in the proof of Theorem 2.3 on page 28 how to sample this domain in PPT.*

function, inverse function:

The function is the usual RSA function $f_{(n,e)}(x) = x^e \bmod n$*, and it is well-known that the inverse of* $f_{(n,e)}(x)$ *can be computed in PPT on* $\mathbf{Z}^*_n$ *given just d.*

[3] Large is formalized below where it is also explained how learn the primefactors n efficiently.

Assumption 2.6 *The RSA-sub-group assumption is that the RSA-sub-group family is a family of trapdoor permutations.*

Remark 2.1 *We try to justify that Assumption 2.6 holds if the RSA assumption holds for the more tradition generator, where p and q are uniformly random from* $\mathrm{PRIMES}(2^k)$ *and e is uniformly random from $\mathbf{Z}^*_{(p-1)(q-1)}$. Consider an RSA modulus $n = pq$, where the distribution on p and q is defined as follows: We pick a random $k\log(k)$-bit number for which the two largest prime numbers are larger than 2^{k-1}, and we let p and q be these prime numbers. Even though we cannot prove it formally, it seems implausible that the RSA assumption would hold for the generator where p and q are random k-bit numbers and would not hold when p and q are chosen as above. Consider then $e \in \mathrm{PRIMES}(2k\log(k))$. Since $n = pq < 2^{2k}$ we have that $\gcd(n, e) = 1$. In the usual definition of the RSA assumption e is chosen uniformly at random from $\mathbf{Z}^*_{\phi(n)}$. To relate our way of picking e to this we could consider the distribution of $e \bmod \phi(n)$ when $e \in \mathrm{PRIMES}(2k\log(k))$. There are about $2^{2k\log(k)}/(2k\log(k)) = 2^{2k}/(2\log(k))$ primes in $\mathrm{PRIMES}(2k\log(k))$ and the size of $\mathbf{Z}^*_{\phi(n)}$ is less than 2^{2k}. If therefore a significant fraction of the exponents $e \in \mathrm{PRIMES}(2k\log(k))$ are weak and assuming that the primes are randomly distributed, then $\{e \bmod \phi(n)\}$, for the weak exponents e, would be a significant fraction of $\mathbf{Z}^*_{\phi(n)}$. Again it is not clear how to do a reduction, but it seems implausible that the RSA assumption would hold for one distribution of the exponents and not the other — note that we* can *do a reduction to the* strong *RSA assumption. We think this justifies the assumption that RSA is one-way with the modulus picked as described above and $e \in \mathrm{PRIMES}(2k\log(k))$, at least relative to Assumption 2.4 on the page before. It is easy to see then that this assumption implies Assumption 2.6, using the Chinese remainder theorem. Therefore Assumption 2.6 is very closely related to the RSA assumption for the more standard key generator described above.*

The reason why we are at all interested in the RSA-sub-group family is that with high probability a random number n is a public key for this family. We return to this issue in Chapter 4.

2.7.5 The Quadratic Residuosity Assumption

The quadratic residuosity assumption for an RSA group $\mathbf{Z}^*_n$ states that a random element of the form $x^2 \bmod n$ cannot be distinguished from a random element from $\mathbf{Z}^*_N$.

Assumption 2.7 *Let gen be an RSA generator and let A be any adversary. Consider the following game $\mathrm{QR}_{gen,A}(k, z)$. Generate $(p, q) \leftarrow gen(k)$. Then generate uniformly random $r_0, r_2 \in \mathbf{Z}^*_{pq}$ and compute $r_1 \leftarrow r_2^2 \bmod pq$. Then input (k, z, pq, r_b) to A, and run A to obtain a value $c \in \{0, 1\}$. Then output c. The quadratic residuosity assumption (QRA) for gen is that $\mathrm{QR}^0_{gen,A} \stackrel{c}{\approx} \mathrm{QR}^1_{gen,A}$ for all PPT algorithms A.*

We only defined the quadratic residuosity assumption for the group $\mathbf{Z}^*_N$. We can defined the quadratic residuosity for $\mathbf{Z}^*_{N^2}$ in the same way. We will however only use this assumption

for N's of the form in Section 2.7.2, for which it is straight forward to verify that the quadratic residuosity assumption holds for $\mathbf{Z}_{N^2}^*$ iff it holds for $\mathbf{Z}_N^*$.[4]

2.8 Invertible Sampling

In this section we introduce the notion of invertible sampling, which is closely connected to adaptive security in simulation models where erasures are not allowed.

Assume that we want to define a flavor of trapdoor permutation where one can generate a public key without getting to know the matching trapdoor. As it turns out we will indeed want to do this in a subsequent section, for now we ask the reader to accept the challenge. So, in addition to the normal key generation algorithm gen, which outputs a public key pk and a trapdoor t, we assume that there exists another algorithm which we call the oblivious generator $g\tilde{e}n$ which outputs only a public key pk with a distribution similar to public keys produced by gen. However, this condition is not sufficient to capture what we want. The oblivious generator $g\tilde{e}n$ could satisfy it by just running gen and then outputting pk. We therefore also ask that there is an efficient algorithm $g\tilde{e}n^{-1}$ that, based only on a public key pk, comes up with a set of random bits r' such that (pk, r') cannot be distinguished from a normal set of random bits and resulting output from $g\tilde{e}n$, in particular $pk = g\tilde{e}n(r')$. This ensures that whatever side information you get from producing pk using $g\tilde{e}n$, you could also compute efficiently from pk itself. In particular, you do not learn t. The property of being able to reconstruct the random bits of a sampler we call invertible sampling after [DN00].

In [Gol01b] Goldreich defines an enhanced trapdoor permutation to be a trapdoor permutation, where one can sample an element in the domain of the trapdoor permutation in a way such that even given the random bits used by the sampler it is hard to invert. We could have used a similar approach to model oblivious public-key generation. We could require that if we generate $pk = g\tilde{e}n(r)$ and give $(y = f_{pk}, r)$ to the adversary, then it is still hard to compute x. It turns out that the approaches are equivalent. Given a trapdoor permutation which is 'enhanced' in the above sense, we can simply let the public key be the random bits used for sampling. Clearly this would be an invertible sampling. On the other hand, if a trapdoor permutation has invertible sampling, then any adversary A which works given the random bits r can be changed into an adversary A' which works given just the public key: Given $(pk \leftarrow g\tilde{e}n(r), y)$ the adversary A' computes $r' \leftarrow g\tilde{e}n^{-1}(pk)$ and runs A on (y, r'). If A has a significantly lower advantage when run on (y, r') than when run on (y, r), then it can distinguish r and r', contradicting the invertible sampling of $g\tilde{e}n$. In the same way we can see that the notion of enhanced trapdoor permutation in [Gol01b] could equivalently be defined by saying that it is a trapdoor permutation where the domain sampler has invertible sampling. We prefer this approach to the definition as it makes minimal changes to the

[4] Consider the group isomorphism $\Psi : \mathbf{Z}_N \times \mathbf{Z}_N^* \to \mathbf{Z}_{N^2}^*, (m, r) \mapsto (N+1)^m r^N \bmod N^2$ from Section 2.7.2. The quadratic residues in $\mathbf{Z}_N \times \mathbf{Z}_N^*$ is the elements $(m, r)^2 = (2m, r^2)$. Since $m \mapsto 2m \bmod N$ is a bijection, the quadratic residues are $\mathbf{Z}_N \times Q_N^*$ and the quadratic non-residues are $\mathbf{Z}_N \times (-1)Q_N^*$. Being a group isomorphism Ψ maps $\mathbf{Z}_N \times (-1)^b Q_N^*$ bijectively to $(-1)^b Q_{N^2}^*$. So, given a random $u \in (-1)^b Q_N^*$, we can produce a random $u' \in (-1)^b Q_{N^2}^*$ as $u' = \Psi(m \leftarrow_R \mathbf{Z}_N, u)$. On the other hand, given a random $u' \in (-1)Q_{N^2}^*$ we know that $u' = (-1)^b (N+1)^m u^N$ for $u \in Q_N^*$, so we can compute $u' \bmod N = (-1)^b u^N \bmod N$, which is a random element from $(-1)^b Q_N^*$ as $u \mapsto u^N \bmod N$ is bijective.

underlying definition. It makes a 'local' requirement on the sampler, instead of changing the definition in which the algorithm is part. Furthermore, invertible sampling is a property which is in general essential to adaptive security of protocols, see e.g. Section 4.4.1 for an example. Having the notion of invertible sampling will allow us to redefine a number of primitives with minimal changes.

Definition 2.16 *Let $A : X \times \{0,1\}^* \to Y$ be a PPT algorithm. We say that A has invertible sampling and that A is a PPTIS algorithm, if there exists a PPT random-bits-faking algorithm $A^{-1} : Y \times X \to \{0,1\}^*$ such that the following holds: For all input $x \in X$, uniformly random bits $r \leftarrow \{0,1\}^*$, output value $y \leftarrow A(x;r)$ and fake random bits $r' \leftarrow A^{-1}(y,x)$ the random variables (x,y,r') and (x,y,r) are statistically close. We say that G is an invertible sampler for a distribution ensemble $X = \{X(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$ if G is a PPTIS algorithm and $X \overset{s}{\approx} \{G(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$. We say that a family of sets $\{S\}_{k\in\mathbf{N},i\in\{0,1\}^*}$ has invertible uniform sampling if there exists a PPTIS algorithm G such that $G(k,i;r)$ is uniform over $S(k,i)$ for uniformly random r.*

Theorem 2.3 *The following domains have invertible uniform sampling:*

1. $\{0,1\}^{k^c}$ *for* $c \in \mathbf{N}$.
2. $\mathbf{Z}_n$ *for an integer* n, *where* $\log(n) = poly(k)$
3. $\mathbf{Z}_n^*$ *for an integer* n, *where* $\log(n) = poly(k)$.
4. *The set* $\mathrm{PRIMES}(k)$.

Proof. $\{0,1\}^{k^c}$ trivially has invertible uniform sampling: Simply take the next k^c random bits on the random tape and output them. The random-bits-faking algorithm on x simply outputs x.

As for $\mathbf{Z}_n$, one sampler could be the following: Let $l = \lceil \log_2(n) \rceil + \epsilon$, where $\frac{1}{2^\epsilon}$ is negligible. Pick $x \in \{0,1\}^l$ uniformly at random and output $x' = x \bmod n$. The random bits faking algorithm runs as follows: Given an element $x' \in \mathbf{Z}_n$, compute $L = 2^l \bmod n$ and pick $y \in \mathbf{Z}_L$ uniformly random. Let $x = yn + x'$ and output x. This gives statistically good fake random bits.

As for $\mathbf{Z}_n^*$, if n only has exponentially large primefactors one can use the algorithms for $\mathbf{Z}_n$. Otherwise, proceed as follows: Pick $x \in \mathbf{Z}_n$ using an invertible uniform sampling as that above. Then test whether $\gcd(x,n) = 1$. If so, output x. Otherwise, try again. After $k\log(k)$ failed trials, output 1. Since $\phi(n) = \Theta(\frac{n}{\log\log(n)})$, see e.g. [BS96, Theorem 8.8.7], it follows that for uniformly random $x \in \mathbf{Z}_n$ we have that $\gcd(x,n) = 1$ with probability at least $\Theta(\frac{1}{\log(k)})$, so $\gcd(x,n) = 1$ occurs with probability as least $\frac{1}{2}$ after $\log(k)$ trials, and so $\gcd(x,n) = 1$ occurs with probability at least $1 - (\frac{1}{2})^k$ after $k\log(k)$ trials. The random-bits-faking algorithm on input x runs as follows: Generate $x' \in \mathbf{Z}_n^*$ as described above. Let $r_1, x_1, r_2, x_2, \ldots, r_l, x_l, r', x'$ be the sequence of elements x_i tested during the generation of x', along with the randomness used to generate them. Use the random bits faking algorithm above on x to compute r. Output $r_1, r_2, \ldots, r_l, r$.

Now for the set of k-bit primes. We fix some way of sampling the primes. For simplicity, say that we pick p by drawing random numbers in $[2^{k-1}, 2^k - 1]$ until we get a number that tests to primality by some probabilistic primality test. We assume that the test recognizes a composite as a prime with exponentially small probability. For the random-bits-faking, we have to reconstruct, from a given output p, a distribution similar to the prefix of numbers that were not primes. This can be done by drawing random numbers in $[2^{k-1}, 2^k - 1]$ until a number is found testing prime. Set r to be the prefix of numbers testing non-prime and the random bits used to test them non-prime. Then run the primality test on p and let r_p be the bits used in the test. Output (r, p, r_p). If p tests prime in this last test, then the output is trivially distributed identically to the bits originally used to pick p. So, by the assumption that p is composite with exponentially small probability it follows that PRIMES(k) has invertible sampling. □

2.9 Σ-Protocols

In this section we define several flavors of Σ-protocols and give some examples. The examples are given to exemplify the introduces notions, but are also used in subsequent sections.

2.9.1 Σ-Protocols, Basic Definition

Consider an instance language $S \subseteq \{0,1\}^*$ and consider a binary relation $R \subseteq S \times \{0,1\}^*$ where $(x, w) \in R$ can be determined in polynomial time for $(x, w) \in S \times \{0,1\}^*$. We call such a relation a PPT binary relation. By the language of R, $L(R) \subseteq S$, we mean the set $\{x \in S | \exists w \in \{0,1\}^*((x, w) \in R)\}$. In the following we call $x \in S$ an instance and for $x \in \{0,1\}^*$ we call $W(x) = \{w \in \{0,1\}^* | (x, w) \in R\}$ the witness set for x and we call $w \in W(x)$ a witness for x. Since $L(R) = \{x \in \{0,1\}^* | W(x) \neq \emptyset\}$, a witness for x is a witness to the fact that $x \in L(R)$.

Recall that we have the convention that the security parameter k is given as input to all TMs and that they must terminate in time k^c for some $c \in \mathbf{N}$. This means that we can only require a PPT TM to work correctly for inputs (x, w) of a given length. When we define that R is PPT we therefore assume that S and R are given as families $S = \{S_k\}_{k \in \mathbf{N}}$ and $R = \{R_k\}_{k \in \mathbf{N}}$ and we require that there exists a polynomial time TM which on security parameter k computes R_k.

Definition 2.17 *A polynomially sized family of sets is a family $S = \{S_k\}_{k \in \mathbf{N}}$ for which there exist $c \in \mathbf{N}$ such that $S_k \subseteq \{0,1\}^{\leq k^c}$ for all $k \in \mathbf{N}$. A polynomial time family of binary relations for S is $R = \{R_k\}_{k \in \mathbf{N}}$ where there exists $c \in \mathbf{N}$ such that $R_k \subseteq S_k \times \{0,1\}^{\leq k^c}$ and where there exists a TM A and $d \in \mathbf{N}$ such that A terminates in less that k^d steps on all inputs $(k, x, w) \in \mathbf{N} \times \{0,1\}^* \times \{0,1\}^*$ and such that on input $(k, x, w) \in \mathbf{N} \times \{0,1\}^* \times \{0,1\}^*$, where $x \in S_k$, the output of A is 1 iff $(x, w) \in R_k$.*

Notice that with the above definition of polynomial time relation we consider a promise problem; We require that there exists a PPT algorithm A which given $(x, w) \in S_k \times \{0,1\}^*$ determines whether $(x, w) \in R_k$. We have no requirements on the output of A when $x \notin S_k$.

3MPR-Protocol (A, l, Z, B)

prover inputs:
The prover has input $(x, w) \in R$ and random bits $r_a \in \{0,1\}^*$.

verifier inputs:
The verifier has input x and randomness $e \in \{0,1\}^l$.

commit message:
The prover computes $a \leftarrow A(x, w; r_a)$ and sends a to the verifier.

challenge message:
The verifier sends e to the prover.

response message:
The prover computes $z \leftarrow Z(x, w, r_a, e)$ and sends z to the verifier.

verification:
The verifier computes a bit $b \leftarrow B(x, a, e, z)$, where 1 signals accept and 0 signals reject.

Figure 2.5: The generic three-move public-randomness protocol for binary relation R.

This is essential as we have not required S_k to by polynomial time. Notice that from $R_k \subseteq S_k \times \{0,1\}^*$ we have that $(x, w) \notin R_k$ whenever $x \notin S_k$.

We will often talk about sets and relations instead of families of sets and families of relations. This is without loss of generality using the following transformation: Given a family of sets S and a family of binary relations we define a corresponding set S and binary relation R as follows: $S = \bigcup_{k \in \mathbf{N}} \{1^k \| 0\} \times S_k$ and $R = \{(1^k \| 0 \| x, w) | \exists k \in \mathbf{N}((x, w) \in R_k)\}$. This transformation is injective. The inverse transformation is given as follows: Given a set S and a binary relation $R \subseteq S \times \{0,1\}^*$ we define a family of sets $\{S_k\}_{k \in \mathbf{N}}$ by $S_k = \{x \in \{0,1\}^* | 1^k \| 0 \| x \in S\}$ and we define a family of binary relations $\{R_k\}_{k \in \mathbf{N}}$ by $R_k = \{(x, w) \in \{0,1\}^* \times \{0,1\}^* | (1^k \| 0 \| x, w) \in R\}$. Notice that if the family of relations is polynomial time, then we can define a polynomial time algorithm which takes inputs from $S \times \{0,1\}^*$ as follows: Given (k, x, w), parse $x = 1^{k'} \| 0 \| x'$ and output 0 if $k' \neq k$. Otherwise, run A on (k, x', w) and output as A.

A non-erasure Σ-protocol for relation R is a protocol for two parties, called the prover P and the verifier V. The prover gets as input $(x, w) \in R$, the verifier gets as input x, and the goal is for the prover to convince the verifier that he knows w such that $(x, w) \in R$, without revealing information about w. We require that it is done using a so-called three-move public-randomness protocol (3MPR-protocol). Such a protocol can always be specified by two algorithms A and Z for computing the first respectively the last message sent by the prover, an integer l specifying the challenge length, i.e. length of the random challenge sent by the verifier, along with an algorithm B, which the verifier applies to its view after the protocol execution to determine whether to accept or reject the proof. The generic form of such a protocol is given in Fig. 2.5. The output b of the verifier indicates whether to believe that the prover knows a valid witness w or not. We first proceed to give a formal definition of the requirements needed from the protocol. After the definition we then discuss how these correspond to the informal requirements that the verifier is convinced that the

prover knows a witness when $b = 1$ and that the prover is guaranteed that the protocol leaks zero information about w.

2.9.1.1 Formally Speaking

A non-erasure Σ-protocol for relation R is a 3MPR-protocol with some extra requirements. Formally it is a tuple $(A, l, Z, B, \texttt{hvs}, \texttt{rbs}, \texttt{xtr})$, where $l \in \mathbf{N}$ and the remaining entries are six PPT algorithms. The parameter l and the first four algorithms were described above. The algorithm `hvs` is called the honest verifier simulator and takes as input $x \in L(R)$, $e \in \{0,1\}^l$ and a uniformly random bit-string $r_{\texttt{hvs}}$ and produces as output (a, z) which is supposed to be distributed as the (a, z) produced by a honest prover with instance x receiving challenge e — this is defined formally below. The algorithm `rbs` is called the random bits simulator. It takes as input $(x, w) \in R$, a challenge $e \in \{0,1\}^l$ and bits $r_{\texttt{hvs}}$, which we think of as the random bits used by `hvs` in a run $(a, z) \leftarrow \texttt{hvs}(x, e; r_{\texttt{hvs}})$, and it produces as output a bit-string r_a such that $a = A(x, w; r_a)$ and $z = Z(x, w, r_a, e)$. I.e. if (a, z) is the messages simulated using `hvs` given just $x \in L(R)$ and $e \in \{0,1\}^l$, if then w such that $(x, w) \in R$ later becomes known it is possible to construct random bits r_a such that it looks as if (a, z) was generated as $a \leftarrow A(x, w; r_a)$ and $z \leftarrow Z(x, w, r_a, e)$ — this is defined formally below. Finally `xtr` is a knowledge extractor, which given two correct conversations with the same first message can compute a witness — again this is defined formally below. Sometimes the extraction process is not able to compute a witness of the exact form of that of the witnesses which are used as input to the protocol. We will see an example that in Lemma 2.8 on page 40. Therefore we introduce the notion of the extraction relation $R^{\texttt{xtr}}$. Typically $R^{\texttt{xtr}} = R$ and we will not mention it, but we allow that $R^{\texttt{xtr}} \supset R$, and then we only require that `xtr` produces a witness w such that $(x, w) \in R^{\texttt{xtr}}$. We do however require that $L(R^{\texttt{xtr}}) = L(R)$, so that the existence of w for which $(x, w) \in R^{\texttt{xtr}}$ proves that $x \in L(R)$. We say that Σ is a non-erasure Σ-protocol for relation R with extraction relation $R^{\texttt{xtr}}$. We now formalize these requirements along with completeness.

Definition 2.18 *A Σ-protocol for relation R is a tuple $(A, l, Z, B, \texttt{hvs}, \texttt{rbs}, \texttt{xtr}, R^{\texttt{xtr}})$, where (A, l, Z, B) is a 3MPR-protocol, `hvs`, `rbs`, `xtr` are PPT algorithms, $R^{\texttt{xtr}}$ is a PPT binary relation, where $L(R^{\texttt{xtr}}) = L(R)$, and where the following requirements hold:*

completeness:
For all $(x, w) \in R$, $r_a \in \{0,1\}^*$ and $e \in \{0,1\}^l$ we have that

$$B(x, A(x, w; r_a), e, Z(x, w, r_a, e)) = 1 \ .$$

(statistical) special non-erasure honest verifier zero-knowledge:
The following two random variables are (statistically close to) identically distributed for all $(x, w) \in R$ and all $e \in \{0,1\}^l$:

$$\mathrm{EXEC}(x, w, e) = [a \leftarrow A(x, w; r_a); z \leftarrow Z(x, w, r_a, e) : (x, w, a, r_a, e, z)]$$
$$\mathrm{SIM}(x, w, e) = [(a, z) \leftarrow \texttt{hvs}(x, e; r_{\texttt{hvs}}); r_a \leftarrow \texttt{rbs}(x, w, e, r_{\texttt{hvs}}) : (x, w, a, r_a, e, z)]$$

special knowledge soundness:

For all $x \in S$ and (a, e, z) and (a, e', z') where $e, e' \in \{0,1\}^l$ and $e \neq e'$, $B(x, a, e, z) = 1$ and $B(x, a, e', z') = 1$, we have that $(x, \mathtt{xtr}(x, a, e, z, e', z')) \in R^{xtr}$.[5]

special membership soundness:

For all $x \in S$, if there exists (a, e, z) and (a, e', z') where $e, e' \in \{0,1\}^l$ and $e \neq e'$, $B(x, a, e, z) = 1$ and $B(x, a, e', z') = 1$, then $x \in L(R)$.[6]

The above definition is in line with the formalization of Σ-protocols in the Ph.D. thesis of Cramer [Cra96], which contains a brilliant treatment of the theory of Σ-protocols. Slight variations of the definition occur in the literature. In particular one could consider a definition where the challenge e is generated by `hvs` (as opposed to being an input to `hvs`). We prefer the adopted definition because it is stronger and can always be obtained (almost for free) from the weaker definition using a trivial transformation of the protocol.[7] Furthermore, the adopted definition is necessary for certain applications, e.g. the protocol in Chapter 5. All Σ-protocols known to the author already meet the stronger definition, so the above transformation has no practical value; It is merely given to show that in choosing between the definitions of `hvs`, nothing is lost in adopting the stronger definition.

Notice that if Σ only has special membership soundness, then given (a, e, z) and (a, e', z') where $e, e' \in \{0,1\}^l$ and $e \neq e'$, $B(x, a, e, z) = 1$ and $B(x, a, e', z') = 1$, one has a proof that $x \in L(R)$, but it is not of the right form to be called a witness. We call (x, a, e, z, e', z') a membership witness.

We call $(A, l, Z, B, \mathtt{hvs}, \mathtt{xtr})$ a Σ-protocol if it meets the definition of a non-erasure Σ-protocol, except that $\mathrm{EXEC}(x, w, e) = [a \leftarrow A(x, w; r_a); z \leftarrow Z(x, w, r_a, e) : (x, w, a, e, z)]$ and $\mathrm{SIM}(x, w, e) = [(a, z) \leftarrow \mathtt{hvs}(x, e; r_{\mathtt{hvs}}) : (x, w, a, e, z)]$. I.e. we do not require that the random bits can be simulated. We say that such a protocol has (statistical) special honest verifier zero-knowledge.

Below we discuss some aspects of the definition.

2.9.1.2 The Meaning of Special Soundness

The knowledge soundness property of a Σ-protocol is supposed to guarantee the verifier that the prover indeed knows a witness. This is guaranteed as follows: Assume that we have run a Σ-protocol in its generic form in Fig. 2.5 on page 30. After this run the verifier knows the values (x, a, e, z), known as a Σ-conversation. If $B(x, a, e, z) = 1$, then we claim that the prover can soundly assume that the prover knows a witness, in the sense that it is written somewhere in the state of the prover or could be computed in PPT from the state of the

[5] For simplicity we assume that `xtr` computes a function.

[6] Note that special knowledge soundness implies special membership soundness.

[7] The transformation works as follows: Assume that we have an algorithm $\mathtt{hvs}'$ which given x generates (a, e, z) such that (x, a, e, z) looks as a real conversation with instance x and uniformly random challenge e for the protocol (A, Z, B). Consider then the protocol where the first message is computed by computing $a \leftarrow A(x, w; r_a)$ and letting $a' = (a, e_P)$ for uniformly random $e_P \in \{0,1\}^l$. Then the verifier chooses a challenge $e_V \in \{0,1\}^l$ and the protocol proceeds as the original protocol, using challenge $e = e_P \oplus e_V$. So, the response message is $z = Z'(x, w, e_V, (r_a, e_P)) = Z(x, w, e_P \oplus e_V, r_a)$. To simulate this protocol (in the stronger sense), on input (x, e_V), compute $(x, e, z) \leftarrow \mathtt{hvs}'(x)$ and let $a' = (a, e_P)$ for $e_P = e \oplus e_V$. Then (a', z) is a uniformly random conversation with challenge e_V.

prover. Assume namely that the prover does not know a witness. This means that the prover can answer at most one challenge correctly, call it e_0.[8] Since the view of the prover is independent of the randomness of the verifier until e is sent, this means that the probability that the prover can answer the challenge is 2^{-l}, namely the probability that $e = e_0$. If e.g. $l = k$, this means that the prover knows w except with exponentially small probability. Notice that in the case $l = 1$, the verifier is only guaranteed that the prover knows w with probability $\frac{1}{2}$. If the prover wants to be better convinced, then the proof can be repeated k times in parallel to get probability $1 - 2^{-k}$ that the prover knows w, see e.g. [Cra96] for details on this.

2.9.1.3 The Meaning Honest Verifier Zero-Knowledge

The notion honest verifier zero-knowledge refers to the fact that a Σ-protocol in its generic form in Fig. 2.5 on page 30 guarantees that no information about w is leaked when the verifier behaves according to the protocol. After the protocol the verifier learned a Σ-conversation (x, a, e, z). One can argue that these values does not reveal anything to the honest verifier, which he could not have computed himself before interacting with the prover. Prior to the protocol execution, the verifier could namely compute $(a, z) \leftarrow \texttt{hvs}(x, e)$ to obtain a simulated Σ-conversation (x, a, e, z). By the honest verifier zero-knowledge the resulting simulated Σ-conversation (x, a, e, z) is distributed statistically close to the real Σ-conversation that the verifier would see. Whatever the real Σ-conversation would allow the verifier to compute he can therefore compute from his own simulated Σ-conversation (x, a, e, z).

2.9.1.4 Tolerating Dishonest Verifiers

The above argument only holds for honest verifiers. The aspect of honesty that is essential is that the prover chooses e *independently* of a. The reason for this is that in producing the simulated Σ-conversation, the value e is used to compute a: $(a, z) \leftarrow \texttt{hvs}(x, e)$. Consider e.g. a dishonest verifier, which in Fig. 2.5 on page 30 computes $e = f(a)$ for a one-way function f. This results in a real Σ-conversation (x, a, e, z), where $e = f(a)$. To simulate this conversation we could try to compute $(a, z) \leftarrow \texttt{hvs}(x, e)$. But, for the simulated Σ-conversation (x, a, e, z) to be of the same form as the real Σ-conversation, we would need that $e = f(a)$. If this is the case, then in particular `hvs` computed $a = f^{-1}(e)$, which we can only assume is hard. Therefore we cannot argue that the verifier could have computed the real Σ-conversation himself. In particular, after the run of the protocol he now holds a piece of data which he could not have computed himself. We cannot in general exclude that this data contains extractable information about w.

[8] To see that this is the case, assume that the prover could also answer another challenge e_1 correctly. This means that after receiving e_0 or e_1 it could compute z_0 respectively z_1 such that $B(x, a, e_0, z_0) = 1$ respectively $B(x, a, e_1, z_1) = 1$. But then the prover could also in PPT compute $w = \texttt{xtr}(x, a, e_0, z_0, e_1, z_1)$ such that $(x, w) \in R^{\texttt{xtr}}$. So, the prover knows a witness. The same line of arguing holds for membership soundness: If $x \notin L(R)$, then by the special membership soundness, only one challenge can be answered. Notice that for the above argument to go through it is essential that a is sent *before* e such that the two Σ-conversations (x, a, e_0, z_0) and (x, a, e_1, z_1) have the same commit message a — this is needed to be able to apply `xtr`.

Therefore we have no guaranteed that a Σ-protocol in its generic form is (dishonest verifier) zero-knowledge. But, fortunately, the aspect that the verifier choses e independently of a is an aspect which we can enforce. There are two generic approaches to this. Both of them forces the verifier to choose e *before* he gets any information about a. Notice that the straight forward approach of sending a after e fails — it was essential for the soundness that a was sent *before* e. It seems that we have a dead-lock. We need to send a before e and we need that e is sent before seeing a.

In come commitment schemes. The notion of commitment scheme is defined exactly to be able to open the above type of dead-locks. Assume that the prover and the verifier agree on a key K for a statistically hiding trapdoor commitment scheme. This allows them to open the dead-lock in one of two ways.

In the first solution the verifier will be the first party to send a message. Before he is given a he chooses e, computes $c \leftarrow \text{commit}_K(e; r)$ and sends c to the prover. Then the prover sends a and the verifier sends (e, r). If $c \neq \text{commit}_K(e; r)$, then the prover stops the protocol. Otherwise the protocol is completed with the challenge e. Now the verifier chooses e *before* a and must always send the chosen challenge e in its second message unless it can break the computational binding of the commitment scheme. Therefore the zero-knowledge property is guaranteed. The soundness is not violated, as the prover has no information about e when he computes a, by the statistical hiding of the commitment scheme.

In the second solution the prover is still the first party to send a message. After computing a he computes $c \leftarrow \text{commit}_K(a; r)$ and sends c to the prover. Then the prover sends e and the verifier sends (a, r, z). If $c \neq \text{commit}_K(a; r)$, then the prover stops the protocol. Otherwise the verifier bases its judgment on the values (x, a, e, z). Now the verifier only receives c before choosing e. Since the commitment scheme is statistically hiding, the value c contains no information about a. Therefore the verifier is forced to pick e before receiving (any information about) e, as desired. Therefore the zero-knowledge property is guaranteed. The soundness is not violated, as the prover cannot open c in two different ways, so if he can answer two different challenges, then it must still be for the same a, which guaranteed the soundness.

Notice that the first solution added a new move to the protocol, whereas the last solution maintains the generic 3MPR-structure. Therefore the second transformation is often preferable. The transformation was first described in [Dam00]. In Chapter 5 we will explore this transformation of a Σ-protocol in detail and give a formal proof that it guarantees soundness and zero-knowledge.

2.9.1.5 The Meaning of Non-Erasure

We discuss the meaning of non-erasure in *special non-erasure honest verifier zero-knowledge*. Recall that we argued that a honest verifier learns nothing from a run of a Σ-protocol in its generic form in Fig. 2.5 on page 30. He could have computed (x, a, e, z) himself as $(a, z) \leftarrow \texttt{hvs}(x, e)$. We claimed that these values are as good as a real Σ-conversation. Anything the verifier can compute from the real Σ-conversation, he can compute from the simulated Σ-conversation.

However, a subtle arises, which demonstrates just how slippery cryptography can be. If

we change *compute from* to *do with*, the above claim turns out to be false. There might actually be something that the verifier can do with the real Σ-conversation, which he cannot do with the simulated one, when *time* is considered.

Consider the following setting: On any given Monday the prover and the verifier runs the protocol in Fig. 2.5 on page 30. As a result the verifier has a Σ-conversation (x, a, e, z). Assume that he is particularly satisfied with this conversations and takes it home with him and shows it to his wife: *Look what a particularly satisfying Σ-conversation I had with Prover today.* The wife agrees that (x, a, e, z) is a particularly satisfying Σ-conversation. Checking his email on Tuesday Verifier sees the following message from Prover: *Hey V, Remember the Σ-conversation we had yesterday? I found it particularly satisfying. Great stuff. At first I couldn't really tell why, but then I had a look at the witness and the randomness that I used. It turns out that we could actually have as many of these conversations as we want. I've attached the witness and the randomness, have a look at it. P.* Prover has a look at w and r_a, but really doesn't get the point. He takes w and r_a home and shows them to his wife: *Look, here is the witness and the randomness from yesterdays particularly satisfying Σ-conversation. Got them from Verifier, and supposedly they should reveal why the Σ-conversation was so satisfying, but I really have no clue. I know it is the right values, look* $a = A(x, w; r_a)$ *and* $z = Z(x, w, r_a, e)$*, but I really don't get the point!*

Now, could Verifier have simulated all this to his wife without having a real Σ-conversation with Prover? Let us assume that all Verifier gets is x on Monday and w on Tuesday, but no Σ-conversation with Prover. On Monday he computes $(a, z) \leftarrow \texttt{hvs}(x, e)$ and takes (x, a, e, z) with him home: *Look what a ....* On Tuesday he receives w. To continue the show he now has to produce r_a such that $a = A(x, w; r_a)$ and $z = Z(x, w, r_a, e)$. Then he can take w and r_a home and talk to his wife. Verifier exactly needs that the Σ-protocol has special non-erasure honest verifier zero-knowledge. If it doesn't he cannot cook up r_a. Now that he has w he could of course do a real conversation with himself: Compute r_a, $a_{Tuesday} \leftarrow A(x, w, r_a)$ and $z_{Tuesday} \leftarrow Z(x, w, a, r_a)$ and take $(x, a_{Tuesday}, e, z_{Tuesday})$ and r_a with him home. But no matter how particularly interesting this conversation might be, the values $a_{Tuesday}$ and $z_{Tuesday}$ might not match the values he brought home with him on Monday. The wife is just bound to notice this. So, exactly what is it that the verifier can do with the real Σ-conversation and not the simulated one? Well, (x, a, e, z) on Monday and r_a on Tuesday. In the simulation he can still produce (x, a, e, z) and r_a of the correct form, but not in two steps, all on Tuesday.

When we reach Chapter 5 the wife will have become the so-called environment in our definition of protocol security and Prover will not give up the random bits so easily. He will be forced to leak them to Verifier because Verifier breaks into his machine on Tuesday and because Prover was not able to *erase* r_a after running the protocol on Monday. As the wife, the environment will care a lot about at which time values are produced, so to simulate the behavior of Verifier to the environment we will need that the our Σ-protocols have the *non-erasure* property.

Σ-Protocol `equality of discrete logarithms`

A:	Given $((i, u_0, v_0, u_1, v_1), r) \in R$, choose $t \leftarrow_R \mathbf{Z}_o$, and for $b \in \{0,1\}$, let $a_b = u_b^t \bmod G_i$. Let $a = (a_0, a_1)$.
I:	The challenge length is $l = \lfloor \log_2(q) \rfloor$.
Z:	Let $z = er + t \bmod o$.
B:	For $b \in \{0,1\}$, check that $u_b, v_b \in G_i$ and $u_b^z \equiv a_b v_b^e \pmod{G_i}$.
`hvs`:	Given (i, u_0, v_0, u_1, v_1) and e, pick $z \leftarrow \mathbf{Z}_o$ and for $b \in \{0,1\}$, let $a_b = u_b^z v_b^{-e} \bmod G_i$.
`rbs`:	Given r such that $v_b \equiv u_b^r \pmod{G_i}$, let $t = z - er \bmod o$.
`xtr`:	Given $((u_0, v_0, u_1, v_1), a_0, a_1, e, z, e', z')$, compute $w = (z - z')(e - e')^{-1} \bmod o$.

Figure 2.6: The Σ-protocol for proving equality of two discrete logarithms in a group with known order.

2.9.2 Example: Discrete Logarithms in Groups with Known Order

To motivate the definition of Σ-protocols we give an example. This is the well-known proof of equality of discrete logarithms from [CP92]. Here the setting is as follows: The prover and the verifier share u_0, u_1, v_0, v_1 from a group G and the prover claims to known r such that $v_0 = u_0^r$ and $v_1 = u_1^r$. The protocol from [CP92] allows the prover to convince the verifier that this is indeed the case.

Definition 2.19 *Assume that we are given a family of groups $G = \{G_i\}$, where given an index i, specifying a group G_i, one can in PPT compute: 1) The order $o \in \mathbf{N}$ of G_i, 2) A lower bound $q \in \mathbf{N}$ on the smallest primefactor of o, 3) Whether $x \in G_i$ for any $x \in \{0,1\}^*$, and 4) Group operations in G_i. The relation* discrete logarithm in G *is defined as follows: The instance language S is the set $\{(i, u, v)\}$, where i specifies a group G_i and $u, v \in G_i$. The relation is given by $((i, u, v), r) \in R$ iff $r \in \mathbf{Z}_o$ and $v \equiv u^r \pmod{G_i}$, where all computations are in G_i with multiplicative notation. The relation* equality of discrete logarithms *is defined as follows: The instance language S is the set $\{(i, u_0, v_0, u_1, v_1)\}$, where i specifies a group G_i and $u_0, v_0, u_1, v_1 \in G_i$. The relation is given by $((i, u_0, v_0, u_1, v_1), r) \in R$ iff $r \in \mathbf{Z}_o$ and $v_b = u_b^r$ for $b \in \{0,1\}$. In Fig. 2.6 the Σ-protocol* `equality of discrete logarithms` *is given. The Σ-protocol* `discrete logarithm` *is given by dropping all values indexed* 1.

Lemma 2.6 *The protocol* `equality of discrete logarithms` *is a non-erasure Σ-protocol for the relation* `equality of discrete logarithms` *with statistical special non-erasure honest verifier zero-knowledge and special knowledge soundness.*

Proof. It is straight forward to check the correctness. We verify the statistical special non-erasure honest verifier zero-knowledge. Consider the values in the simulation. We have that $a_b \equiv u_b^z v_b^{-e} \bmod G_i$, $v_b \equiv u_b^r \pmod{G_i}$ and $t = z - er \bmod o$. This gives us that $a_b \equiv u_b^z (u_b^r)^{-e} \equiv u_b^{z-re} \equiv u_b^t \pmod{G_i}$ and $z \equiv t + er \pmod{o}$, exactly as in the execution. Therefore the relation between all values is the same in the simulation and the execution.

3MPR-Protocol $(\mathcal{I}, A, l, Z, B)$

setup phase:
Generate a random index $i \leftarrow gen(k)$.

prover inputs:
The prover has $(x, w) \in R$, where $x \in \mathcal{I}_i$, and random bits $r_a \in \{0,1\}^*$.

verifier inputs:
The verifier has input x and randomness $e \in \{0,1\}^l$.

commit message:
The prover computes $a \leftarrow A(x, w; r_a)$ and sends a to the verifier.

challenge message:
The verifier sends e to the prover.

response message:
The prover computes $z \leftarrow Z(x, w, r_a, e)$ and sends z to the verifier.

verification:
The verifier computes a bit $b \leftarrow B(x, a, e, z)$, where 1 signals accept and 0 signals reject.

Figure 2.7: The generic three-move public-randomness protocol for binary relation R, with instances drawn from a set $\mathcal{I}_i$.

We check the distribution. In the execution t is uniformly random. In the simulation z is uniformly random and $t = z - er \bmod o$. This gives the same distribution on t.

To verify the special knowledge soundness, assume that we are given

$$(u_0, v_0, u_1, v_1, a_0, a_1, e, z_0, z_1, e', z_0', z_1') \ ,$$

where, for $b \in \{0,1\}$, $u_b^z \equiv a_b v_b^e \bmod G_i$ and $u_b^{z'} \equiv a_b v_b^{e'} \bmod G_i$. Then $u_b^{z-z'} \equiv v_b^{e-e'} \bmod G_i$. Using that $|e - e'| < q$ it follows that $\gcd(e - e', o) = 1$. So, compute $\alpha, \beta \in \mathbf{N}$ such that $(e - e')\alpha = 1 + \beta o$, and it follows that $u_b^{(z-z')\alpha} \equiv v_b^{(e-e')\alpha} \equiv v_b (v_b^{\beta})^o \equiv v_b \bmod G_i$. So, $w = (z - z')\alpha \bmod o \equiv (z - z')(e - e')^{-1} \pmod{o}$ is a correct witness. □

2.9.3 Computational Soundness

In this section we present a weakening of the notion of Σ-protocol. We define a notion of computational special knowledge soundness (respectively computational special membership soundness), which is meant to capture the intuition that the `xtr` algorithm can compute a witness, unless some computational assumption was broken. This computational assumption is associated to the instances considered. The instances are given by some index i. This index is guaranteed to have a specific distribution and specifies the set of instances that the verifier is supposed to work for. Since i, and thus the instances, is guaranteed to have a specific distribution, we can associate computational assumptions to the instances. We will see a use of this in Lemma 2.8 on page 40.

The distribution on the instances is specified as follows: Let R be a binary relation, let Σ be a Σ-protocol for R and let $\mathcal{I} = (gen, I)$ be a PPT family of sets. The Σ-protocol for

this setting is only intended to be run for random indices and instances $x \in \mathcal{I}_i$. The generic form of such a protocol is given in Fig. 2.7 on the page before. We formalize that no PPT adversary can break the special soundness of Σ for instances drawn at random from the family of sets $\mathcal{I}$:

Definition 2.20 *A computationally sound Σ-protocol for relation R is a tuple $(\mathcal{I}, A, l, Z, B, \mathtt{hvs}, \mathtt{rbs}, \mathtt{xtr}, R^{xtr})$, where $(\mathcal{I}, A, l, Z, B)$ is a 3MPR-protocol of the form in Fig. 2.7 on the preceding page and where the following requirements hold:*

completeness:
For all $i \in gen$, $(x, w) \in R$, where $x \in \mathcal{I}_i$, $r_a \in \{0,1\}^*$ and $e \in \{0,1\}^l$ we have that

$$B(x, A(x, w; r_a), e, Z(x, w, r_a, e)) = 1 \ .$$

(statistical) special non-erasure honest verifier zero-knowledge:
The following two random variables are (statistically close to) identically distributed for all $i \in gen$, $(x, w) \in R$, where $x \in \mathcal{I}_i$, and all $e \in \{0,1\}^l$:

$$\mathrm{EXEC}(x, w, e) = [a \leftarrow A(x, w; r_a); z \leftarrow Z(x, w, r_a, e) : (x, w, a, r_a, e, z)]$$
$$\mathrm{SIM}(x, w, e) = [(a, z) \leftarrow \mathtt{hvs}(x, e; r_{\mathtt{hvs}}); r_a \leftarrow \mathtt{rbs}(x, w, e, r_{\mathtt{hvs}}) : (x, w, a, r_a, e, z)]$$

computational special knowledge soundness:
Given an adversary $\mathcal{A}$, define a Boolean distribution ensemble $\mathrm{SKS}_{\Sigma,\mathcal{I},\mathcal{A}} = \{\mathrm{SKS}_{\Sigma,\mathcal{I},\mathcal{A}}(k, z)\}_{k \in \mathbf{N}, z \in \{0,1\}^*}\}$ (the special knowledge soundness game) as follows: Generate $i \leftarrow gen(k)$ and compute $v \leftarrow \mathcal{A}(k, z, i)$. If v is of the form (x, a, e, z, e', z') for $x \in \mathcal{I}_i$ and $e, e' \in \{0,1\}^l$ and $e \neq e'$, $B(x, a, e, z) = 1$, $B(x, a, e', z') = 1$, and $(x, \mathtt{xtr}(x, a, e, z, e', z')) \notin R^{xtr}$, then output 1; Otherwise output 0. We say that Σ has computational special knowledge soundness relative to $\mathcal{I}$ if $\mathrm{SKS}_{\Sigma,\mathcal{I},\mathcal{A}} \overset{c}{\approx} 0$ for all PPT $\mathcal{A}$.

computational special membership soundness:
Given an adversary $\mathcal{A}$, define a Boolean distribution ensemble $\mathrm{SMS}_{\Sigma,\mathcal{I},\mathcal{A}} = \{\mathrm{SMS}_{\Sigma,\mathcal{I},\mathcal{A}}(k, z)\}_{k \in \mathbf{N}, z \in \{0,1\}^*}\}$ (the special membership soundness game) as follows: Generate $i \leftarrow gen(k)$ and compute $v \leftarrow \mathcal{A}(k, z, i)$. If v is of the form (x, a, e, z, e', z') for $x \in \mathcal{I}_i$ and $e, e' \in \{0,1\}^l$ and $e \neq e'$, $B(x, a, e, z) = 1$, $B(x, a, e', z') = 1$, and $x \notin L(R)$, then output 1; Otherwise output 0. We say that Σ has computational special membership soundness relative to $\mathcal{I}$ if $\mathrm{SMS}_{\Sigma,\mathcal{I},\mathcal{A}} \overset{c}{\approx} 0$ for all PPT $\mathcal{A}$.

In the context of computational special knowledge soundness, we call a membership witness (x, a, e, z, e', z') for which $(x, \mathtt{xtr}(x, a, e, z, e', z')) \notin R^{\mathtt{xtr}}$ a break of the extractor. The computational special knowledge soundness property basically says that, for properly distributed instances, it is not possible to compute a break of the extractor in polynomial time. Notice that the game $\mathrm{SMS}_{\Sigma,\mathcal{I},\mathcal{A}}$ used to define the computational special membership soundness is not necessarily PPT, as the test $x \notin L(R)$ is typically not PPT.

Σ-Protocol `equality of RSA discrete logarithms`

A: Given $((N, B, s, u_0, v_0, u_1, v_1), r) \in R$, let $L = \lceil \log_2(B) + 3k/2 \rceil$ and choose a random L-bit number t, for $b \in \{0, 1\}$, let $a_b = u_b^t \bmod N^{s+1}$ and let $a = (a_0, a_1)$.

I: The challenge length is $k/2 - 2$ bits.

Z: Let $z = 2er + t$.

B: Given $((N, B, s, u_0, v_0, u_1, v_1), a, e, z)$, for $b \in \{0, 1\}$, check that $u_b, v_b \in \mathbf{Z}^*_{N^{s+1}}$ and $u_b^z \equiv a_b v_b^{2e} \pmod{N^{s+1}}$.

`hvs`: Given $(N, B, s, u_0, v_0, u_1, v_1) \in L(R)$ and e, pick z as a uniformly random L-bit number and let $a_b = u_b^z v_b^{-2e} \bmod N^{s+1}$ for $b \in \{0, 1\}$.

`rbs`: Given r such that $v_b = u_b^r \bmod N^{s+1}$, let $t = z - 2er$.

`xtr`: Assume then that we are given $((N, B, s, u_0, v_0, u_1, v_1), a_0, a_1, e, z, e', z')$. Compute $d = \gcd(2(e - e'), z - z')$ and output $w = (z - z')/d$.

Figure 2.8: The Σ-protocol for proving equality of two discrete logarithms in an RSA group.

2.9.4 Example: Discrete Logarithms in RSA Groups

To motivate the definition of a computationally sound Σ-protocol, we give an example of a Σ-protocol which only has special knowledge soundness in the computational sense. The prover and the verifier have input $(N, B, s, u_0, u_1, v_0, v_1)$, where N is an RSA modulus, and the prover claims to know $r \in \mathbf{Z}_B$ such that $v_0 = u_0^r \bmod N^{s+1}$ and $v_1 = u_1^r \bmod N^{s+1}$. The Σ-protocol will only have special membership soundness, but will only have *computational* special knowledge soundness. The primary reason for this is that the order of the group $\mathbf{Z}^*_{N^{s+1}}$ is not known — knowing it is equivalent to knowing the factorization of N. Therefore we cannot use the Σ-protocol from Fig. 2.6 on page 36. Since the order of $\mathbf{Z}^*_{N^{s+1}}$ is not known the prover cannot modular reduce its witness $r \in \mathbf{Z}_B$ to $\mathbf{Z}_{\mathrm{ord}(\mathbf{Z}^*_{N^{s+1}})}$. This is why B is given; Its purpose is to give both the prover and the verifier a bound on the size of the witness r. Since the order of $\mathbf{Z}^*_{N^{s+1}}$ has the primefactor 2 we would be forced the use challenges from $\mathbf{Z}_2$ (c.f. Fig. 2.6 on page 36). To avoid this, we set up N so that the sub-group of quadratic residues only has large primefactors and then do the proof only for the component of u and v in the quadratic residues.

Definition 2.21 *The relation RSA discrete logarithm is defined as follows: The instance language S is the set $\{(N, B, s, u, v)\}$, where $N = pq$ such that $p, q \in \mathrm{PRIMES}(\lceil k/2 \rceil)$ and $(p - 1)/2$ and $(q - 1)/2$ are primes, $s \in \mathbf{N}$, $B \in \mathbf{N}$, $B \geq \mathrm{ord}(\mathbf{Z}^*_{N^{s+1}})$, $u, v \in \mathbf{Z}^*_{N^{s+1}}$. The relation is given by $((N, B, s, u, v), r) \in R$ iff $r \in \mathbf{Z}_B$ and $v^2 = u^{2r} \bmod N^{s+1}$. The relation equality of RSA discrete logarithms is defined as follows: The instance language S is the set $\{(N, B, s, u_0, v_0, u_1, v_1)\}$, where N, B and s are as above and $u_0, v_0, u_1, v_1 \in \mathbf{Z}^*_{N^{s+1}}$. The relation is given by $((N, B, s, u_0, v_0, u_1, v_1), r) \in R$ iff and $r \in \mathbf{Z}_B$ and, for $b \in \{0, 1\}$, $v_b^2 = u_b^{2r} \bmod N^{s+1}$. In Fig. 2.8 the Σ-protocol `equality of RSA discrete logarithms` is given. The Σ-protocol `RSA discrete logarithm` is given by dropping all values indexed 1.*

We first prove that the protocol has membership soundness.

Lemma 2.7 *The protocol* `equality of RSA discrete logarithms` *is a non-erasure* Σ*-protocol for the relation* `equality of RSA discrete logarithms` *with statistical special non-erasure honest verifier zero-knowledge and special* membership *soundness.*

Proof. Correctness is straight forward. For the statistical zero-knowledge, the protocol and the simulation are identical except that in the protocol t is a uniformly random number from $[0, 2^L - 1)$ and in the simulation t is a uniformly random number from $[-er, 2^L - er)$. Since $er < 2^{k/2}B$ it follows that the statistical distance is less than 2^{-k}.

To verify the special membership soundness, assume that we are given

$$(N, B, s, u_0, v_0, u_1, v_1, a_0, a_1, e, z_0, z_1, e', z_0', z_1') \ ,$$

where, for $b \in \{0,1\}$, $u_b^z \equiv a_b v_b^{2e} \pmod{N^{s+1}}$ and $u_b^{z'} \equiv a_b v_b^{2e'} \pmod{N^{s+1}}$. Then $u_b^{z-z'} \equiv v_b^{2(e-e')} \pmod{N^{s+1}}$. Using that e and e' are $(k/2-2)$-bit values and that $N = (2p'+1)(2q'+1)$ for $(k/2-1)$-bit primes p' and q', we have that $\gcd(2(e-e'), N^s M) = 1$, where $M = p'q'$ is the order of Q_N.[9] This means that there exists $\alpha \in \mathbf{N}$ such that $2(e-e')\alpha = 1 + \beta N^s M$, and it follows that $u_b^{(z-z')\alpha} \equiv v_b^{2(e-e')\alpha} \equiv v_b((v_b)^\beta)^{N^s M} \pmod{N}$. Let $\omega = ((v_b)^{-\beta})^{N^s M} \bmod N^{s+1}$. Since the orders of all elements of $\mathbf{Z}^*_{N^{s+1}}$ divide $2N^s M$, we have that $\omega^2 \bmod N^{s+1} = 1$. Therefore, $v_b^2 = u_b^{2(z-z')\alpha} \bmod N^{s+1}$. So, $v_b^2 \equiv u_b^{2r} \pmod{N^{s+1}}$ for some $r < \mathrm{ord}(\mathbf{Z}_{N^{s+1}}) \leq B$, as desired. □

Notice that in the proof we used the inverse of $e - e'$ modulo $N^s M$. Since $N^s M$ is as hard to compute from N^{s+1} as factoring N, the proof does not specify a PPT extraction algorithm. If however the distribution of u_0 is such that the strong RSA assumption holds for (N, u_0), then the protocol has computational knowledge soundness.

Lemma 2.8 *Consider the following PPT family of sets. Consider any generator gen which for security parameter k generates a k-bit RSA modulus N as specified in Definition 2.21 on the preceding page and generates $s \in \mathbf{N}$ and uniformly random $u \in Q^*_{N^{s+1}}$. We let (N, s, u) define the set of all instances of the form $(N, B, s, u, v_0, u_1, v_1)$, where $B \in \mathbf{N}$, $v_0, u_1, v_1 \in \{0,1\}^*$. This defines a PPT family of sets.*

If the strong RSA assumption holds for (N, u) generated by the generator gen, then the protocol `equality of RSA discrete logarithms` *is a non-erasure Σ-protocol for the relation* `equality of RSA discrete logarithms` *with statistical special non-erasure honest verifier zero-knowledge and computational special knowledge soundness relative to the above PPT family of sets. The Σ-protocol is for the extraction relation $R^{\mathtt{xtr}}$, where we put no bound on the discrete logarithm r extracted, i.e. we do not require that $r \in \mathbf{Z}_B$.*

Proof. The correctness and the statistical zero-knowledge follows as above.

To verify the computational special knowledge soundness, assume for the sake of contradiction that there exists a PPT algorithm A, which given $(N, s, u_0) \leftarrow gen$, with non-negligible probability p, can output $((N, B, s, u_0, v_0, u_1, v_1), a_0, a_1, e, z, e', z')$ such that, $e, e' \in \{0,1\}^l$, $e \neq e'$ and $u_b^z \equiv a_b v_b^{2e} \pmod{N^{s+1}}$ and $u_b^{z'} \equiv a_b v_b^{2e'} \pmod{N^{s+1}}$ and $v_0^2 \neq u_0^{2(z-z')/d} \bmod N^{s+1}$ (or $v_0^2 = u_0^{2(z-z')/d} \bmod N^{s+1}$ but $v_1^2 \neq u_1^{2(z-z')/d} \bmod N^{s+1}$).

[9] We assume that the security parameter is large enough that $p, p', q, q' > 2$.

We prove that if $v_0 \neq u_0^{(z-z')/d} \bmod N^{s+1}$ happens with non-negligible probability p, then we can compute a non-trivial root of u_0 modulo N with probability p. For notational convenience, let $u = u_0$ and let $v = v_0$. We have that $u^{z-z'} \equiv v^{2(e-e')} \pmod{N^{s+1}}$. We can compute $d = \gcd(z - z', 2(e - e'))$, $\epsilon = 2(e - e')/d$ and $\sigma = (z - z')/d$ and we have that $u^{\sigma d} \equiv v^{\epsilon d} \pmod{N^{s+1}}$. Since $d|2(e - e')$ and e and e' are $(k/2 - 2)$-bit integers it follows that $\gcd(d, 2N^s M) \in \{1, 2\}$. We can therefore compute $a \in \mathbf{N}$ such that $da = 2 + b2MN^s$ and it follows that $u^{\sigma 2} \equiv v^{\epsilon 2} \pmod{N^{s+1}}$. By assumption $v^2 \neq u^{\sigma 2} \bmod N^{s+1}$, so $\epsilon > 1$. Let $\omega = u^{\sigma} v^{-\epsilon} \bmod N^{s+1}$, such that $u^{\sigma} \equiv \omega v^{\epsilon} \pmod{N^{s+1}}$. Notice that $\omega^2 \bmod N^{s+1} = 1$.

We have that $\gcd(\epsilon, \sigma) = 1$, so we can compute $\alpha, \beta \in \mathbf{N}$ such that $\alpha\epsilon + \beta\sigma = 1$. Then $u = u^{\alpha\epsilon+\beta\sigma} \equiv (u^{\alpha})^{\epsilon}(u^{\sigma})^{\beta} \equiv (u^{\alpha})^{\epsilon}(\omega v^{\epsilon})^{\beta} \equiv \omega^{\beta}(u^{\alpha} v^{\beta})^{\epsilon} \pmod{N^{s+1}}$. So, $u = \omega^{\beta} x^{\epsilon} \bmod N^{s+1}$. If ϵ is odd, then $\omega = \omega^{\epsilon} \bmod N^{s+1}$ and $u = (\omega^{\beta} x)^{\epsilon} \bmod N^{s+1}$, and since $\epsilon > 1$ we have contradicted the strong RSA assumption. If ϵ is even, then $u = \omega^{\beta}(x^{\epsilon'})^2 \bmod N^{s+1}$, where $\omega^{\beta} \bmod N^{s+1}$ is a square root of 1. Assume that this case happens with non-negligible probability. We can then run A on N and u where u is sampled as $u = u'^2 \bmod N^{s+1}$, so that u' is uniformly random among the four square roots of u and $u^2 = u'^4 \bmod N^{s+1}$. With non-negligible probability we also have that $u^2 = (x^{\epsilon'})^4 \bmod N^{s+1}$, giving us $1 = (u' x^{-\epsilon'})^4 \bmod N^{s+1}$. Since u' was random among the four possible values, $u' x^{-\epsilon'} \bmod N^{s+1}$ is a non-trivial square root of 1 with non-negligible probability, which allows to derive p and q with non-negligible probability, contradicting, among other things, the strong RSA assumption.

So, assuming the strong RSA assumption we can assume that $v_0^2 = u_0^{2\sigma} \bmod N^{s+1}$ for $\sigma = (z - z')/d$, except with negligible probability. Now, by the special membership soundness we have that there *exists* r for which $v_0^2 = u_0^{2r} \bmod N^{s+1}$ and $v_1^2 = u_1^{2r} \bmod N^{s+1}$. From $v_0^2 \equiv u_0^{2\sigma} \pmod{N^{s+1}}$ and $v_0^2 \equiv u_0^{2r} \pmod{N^{s+1}}$ we get that $u_0^{2r} \equiv u_0^{2r} \pmod{N^{s+1}}$. We have assumed that u_0 is uniformly random from $Q^*_{N^{s+1}}$, so u_0 generates $Q^*_{N^{s+1}}$ except with negligible probability, so $u_0^2 \bmod N^{s+1}$ generates $Q^*_{N^{s+1}}$ except with negligible probability. From $(u_0^2)^r \equiv (u_0^2)^{\sigma} \pmod{N^{s+1}}$ we therefore get that $\sigma \equiv r \pmod{N^s M}$, and so $u_1^{2\sigma} \equiv v_1^2 \pmod{N^s M}$. All in all, $v_0^2 = u_0^{2\sigma} \bmod N^{s+1}$ and $v_1^2 = u_1^{2\sigma} \bmod N^{s+1}$, except with negligible probability. □

Notice that $r = (z - z')/d$ by far is guaranteed to be from $\mathbf{Z}_B$. So, we needed the relaxation with the extraction relation, as we cannot reduce r modular the unknown order $N^s M$.

Corollary 2.1 *Consider any generator gen as defined in Lemma 2.8 on the preceding page. We let (N, s, u) define the set of all instances (N, s, u, v) where $v \in \{0,1\}^*$. This defines a PPT family of sets.*

If the strong RSA assumption holds for the generator gen, then the protocol `RSA discrete logarithm` *is a non-erasure Σ-protocol for the relation* `RSA discrete logarithm` *with statistical special non-erasure honest verifier zero-knowledge and computational special soundness relative to the above family of sets and extraction relation $R^{\mathtt{xtr}}$, where we put no bound on the discrete logarithm r extracted.*

4MPRS-Protocol $(\mathtt{prs}, A, l, Z, B)$

prover inputs:
The prover has input $(x, w) \in R$ and random bits $r_a \in \{0,1\}^*$.

verifier inputs:
The verifier has input x and randomness $e \in \{0,1\}^l$ and $r \in \{0,1\}^*$.

reference string message:
The verifier computes $s \leftarrow \mathtt{prs}(r)$ and sends s to the prover.

commit message:
The prover computes $a \leftarrow A((x,s), w; r_a)$ and sends a to the verifier.

challenge message:
The verifier sends e to the prover.

response message:
The prover computes $z \leftarrow Z((x,s), w, r_a, e)$ and sends z to the verifier.

verification:
The verifier computes a bit $b \leftarrow B((x,s), a, e, z)$, where 1 signals accept and 0 signals reject.

Figure 2.9: The generic four-move private reference string protocol for binary relation R.

2.9.5 Private Reference String Σ-Protocols

We define the notion of private reference string Σ-protocol (PRS Σ-protocol). This is a tuple $(\mathtt{prs}, A, l, Z, B, \mathtt{hvs}, \mathtt{rbs}, \mathtt{xtr})$, where basically $(A, l, Z, B, \mathtt{hvs}, \mathtt{rbs}, \mathtt{xtr})$ is a Σ-protocol, but with the addition that we allow the verifier to chose a private reference string, which is used by both parties in the protocol. Before the protocol is run the verifier should generate and make public a string $s \leftarrow \mathtt{prs}(r)$, where $r \leftarrow_R \{0,1\}^*$ and $\mathtt{prs}$ is a PPT generator. The generic structure of such a protocol is given in Fig. 2.9

Since the verifier is allowed to pick s, and knows r, only the verifier should depend on s being computed correctly and r being unknown. In particular the zero-knowledge should hold no matter how s is chosen. The knowledge soundness (membership soundness) will again only be computational and will depend one s being computed as $s \leftarrow \mathtt{prs}(r)$ for uniformly random r. Since the soundness protects the verifier, which generates s, it makes sense to assume that s is chosen at random when verifying soundness. The only restriction that we put on a dishonest verifier is that it knows r such that $s = \mathtt{prs}(r)$. This will allow us assume that we know r when we argue the zero-knowledge. Remember that the zero-knowledge simulation of the Σ-conversation is done to demonstrate that the verifier could have computed the Σ-conversation on its own. Since we assume that the verifier knows r it can enter into the computation that have the purpose of demonstrating what the verifier could have computed on its own. Below we will discuss how to ensure that the verifier knows r and we will give an example to justify the notion of a PRS Σ-protocol.

Definition 2.22 *A PRS Σ-protocol for relation R is a tuple $(\mathtt{prs}, A, l, Z, B, \mathtt{hvs}, \mathtt{rbs}, \mathtt{xtr}, R^{xtr})$, where $(\mathtt{prs}, A, l, Z, B)$ is a 4MPRS-protocol of the form in Fig. 2.9 and where the following requirements hold:*

completeness:

For all $s \in \mathtt{prs}$, $(x,w) \in R$, $r_a \in \{0,1\}^*$ and $e \in \{0,1\}^l$ we have that

$$B((x,s), A((x,s),w;r_a), e, Z((x,s),w,r_a,e)) = 1 \ .$$

(statistical) special non-erasure honest verifier zero-knowledge:

The following two random variables are (statistically close to) identically distributed for all r and $s = \mathtt{prs}(r)$, $(x,w) \in R$ and all $e \in \{0,1\}^l$:

$$\begin{aligned}
&\mathrm{EXEC}(x,w,e) \\
&\quad = [a \leftarrow A((x,s),w;r_a); z \leftarrow Z((x,s),w,r_a,e) : ((x,s),w,a,r_a,e,z)] \\
&\mathrm{SIM}(x,w,e) \\
&\quad = [(a,z) \leftarrow \mathtt{hvs}((x,r),e;r_{\mathtt{hvs}}); r_a \leftarrow \mathtt{rbs}((x,r),w,e,r_{\mathtt{hvs}}) : ((x,s),w,a,r_a,e,z)]
\end{aligned}$$

computational special knowledge soundness:

Given an adversary $\mathcal{A}$, define a Boolean distribution ensemble $\mathrm{SKS}_{\Sigma,\mathtt{prs},\mathcal{A}} = \{\mathrm{SKS}_{\Sigma,\mathtt{prs},\mathcal{A}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}\}$ (the special knowledge soundness game) as follows: Generate $s \leftarrow \mathtt{prs}(k)$ and compute $v \leftarrow \mathcal{A}(k,z,s)$. If v is of the form (x,a,e,z,e',z') and $e,e' \in \{0,1\}^l$ and $e \neq e'$, $B((x,s),a,e,z) = 1$, $B((x,s),a,e',z') = 1$, and $(x, \mathtt{xtr}(x,a,e,z,e',z')) \notin R^{\mathtt{xtr}}$, then output 1; Otherwise output 0. We say that Σ has computational special knowledge soundness relative to $\mathtt{prs}$ if $\mathrm{SKS}_{\Sigma,\mathtt{prs},\mathcal{A}} \overset{\mathrm{c}}{\approx} 0$ for all PPT $\mathcal{A}$.

computational special membership soundness:

Given an adversary $\mathcal{A}$, define a Boolean distribution ensemble $\mathrm{SMS}_{\Sigma,\mathtt{prs},\mathcal{A}} = \{\mathrm{SMS}_{\Sigma,\mathtt{prs},\mathcal{A}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}\}$ (the special knowledge soundness game) as follows: Generate $s \leftarrow \mathtt{prs}(k)$ and compute $v \leftarrow \mathcal{A}(k,z,s)$. If v is of the form (x,a,e,z,e',z') and $e,e' \in \{0,1\}^l$ and $e \neq e'$, $B((x,s),a,e,z) = 1$, $B((x,s),a,e',z') = 1$, and $x \notin L(R)$, then output 1; Otherwise output 0. We say that Σ has computational special membership soundness relative to $\mathtt{prs}$ if $\mathrm{SMS}_{\Sigma,\mathtt{prs},\mathcal{A}} \overset{\mathrm{c}}{\approx} 0$ for all PPT $\mathcal{A}$.

2.9.5.1 Setting Up the Private Reference String

The notion of Σ-protocol is primarily a technical notion. When Σ-protocols are used for doing actual zero-knowledge proofs, the protocols are transformed in various ways to make them appropriate, e.g. making them zero-knowledge as opposed to special honest verifier zero-knowledge. In particular, when using a private reference string Σ-protocol one needs a mechanism for setting up the private reference string s. To be appropriate, this mechanism must somehow guarantee that $s \in \mathtt{prs}$ and that the verifier knows r such that $s = \mathtt{prs}(r)$. How this is done depends on the context. One possibility is to let s be generated by some trusted party, which distributes s to the prover and the verifier and keeps r hidden! Another possibility is to let the verifier generate $s \leftarrow \mathtt{prs}(r)$ and then prove to the prover in zero-knowledge it knows r such that $s = \mathtt{prs}(r)$.[10] The proof given by the verifier should of

[10] Note that such an approach would be compatible with the definition of private reference string Σ-protocol. When the verifier is honest he will generate s at random and since he gives a *zero-knowledge* proof that he

course be appropriate for the context. If we use the private reference string Σ-protocol for building a protocol which must be secure under concurrent composition, then the proof given by the verifier probably must be secure under concurrent composition also. A third solution, which is just a heuristic realization of the second approach, would have the verifier generate $s \leftarrow \mathtt{prs}(r)$, register s at some certificate authority and at the same time prove that it knows r. When the prover wants to prove to the verifier he will then retrieve s from the verifier along with a certificate saying that s belongs to the verifier and that it was proved that the verifier knows r. Again, the appropriateness of such an approach would depend on the context. Examples of private reference string Σ-protocols are given in the following section and in Definition 2.27 on page 54. Chapter 5 contains our main use of Σ-protocols, to build zero-knowledge protocols, and also covers the use of private reference string Σ-protocols. A formal model for modeling the setup of the private reference string is given in Section 3.8.8.

2.9.6 Example: A PRS Σ-Protocol for All of NP

To motivate the notion of PRS Σ-protocol, in this section we describe a PRS Σ-protocol for all of NP. The first zero-knowledge protocol for all of NP was presented by Goldreich, Micali and Wigderson [GMW86].

Assume that we given any polynomial time relation $R \subseteq S \times \{0,1\}^*$. By definition there exists a polynomial time TM A which on input $(x,w) \in S_k \times \{0,1\}^*$ outputs 1 iff $(x,w) \in R_k$. Since there exists a polynomial k^d bounding the running time of A we can without loss of generality assume that the algorithm for computing R_k is given by a polynomially sized family of circuits. This is a PPT generator C which on input k and $x \in S_k$ outputs a description of a circuit C_k for computing $w \mapsto A(k,x,w)$. The circuit is specified by a value $C = (l, (G_l, \ldots, G_{l+m-1}), L)$, where $l \in \mathbf{N}$ is the number of circuit input gates, $(G_l, \ldots, G_{l+m-1})$ is the circuit structure and $L \in \{0,1\}^{4m}$ is the circuit logic. Each of the gates G_i is of the form (i_0, i_1), where $0 \leq \{i_0, i_1\} < i$ are the indices of the input gates of G_i. The circuit logic L is used to specify the gate logic of each gate. We index L using $i \in \{l, \ldots, l+m-1\}$ and $(b_0, b_1) \in \{0,1\} \times \{0,1\}$ using the scheme $L_{i,b_0,b_1} = L_{4(i-l)+2b_0+b_1}$. In the following we assume that the circuit structure is fixed and that only the circuit logic L and the input w are variables. For $w \in \{0,1\}^l$ the circuit evaluation is a value $L[w] \in \{0,1\}^{l+m}$, defined as follows: $L[w]_i = w_i$ for $i = 0, \ldots, l-1$, $L[w]_i = L_{i,L[w]_{i_0},L[w]_{i_1}}$ for $i = l, \ldots, l+m-1$. The circuit output is $L[w]_{l+m-1}$.

We make an observation about the information needed to evaluate a circuit on a given input. This observation will be essential to our protocol. Assume that you know the structure of a circuit but does not know the input w or the logic L. To verify that a value $E \in \{0,1\}^{l+m}$ is a correct circuit evaluation all you need to know is w and the bit in each gate logic which specifies the output on the specific gate inputs resulting from evaluating C on w. We call the specification of these entries a w-path and we call the values in the entries the w-path logic. For $w \in \{0,1\}^l$, the w-path in L is defined as follows: Let $E = L[w]$ be the circuit

knows r, he will keep r hidden. This guarantees the special soundness of the Σ-protocol — a property in which the verifier is interested. If on the other hand the verifier is corrupted, the proof that r is known is enough to guarantee special honest verifier zero-knowledge of the Σ-protocol — a property in which the prover is interested.

evaluation $E \in \{0,1\}^{l+m}$. The w-path is $P = (P_l, \ldots, P_{l+m-1})$, where for $i = l, \ldots, l+m-1$ we let $G_i = (i_0, i_1)$ be the i'th gate structure and let $P_i = 4(i-l) + 2E_{i_0} + E_{i_1}$. The w-path logic in L is $V = (V_l, \ldots, V_{l+m-1}) \in \{0,1\}^m$, where $V_i = L_{P_i}$ for $i = l, \ldots, l+m-1$ and P being the w-path in L. Assume that you are given an input $w \in \{0,1\}^*$, a w-path P and a w-path logic V. To compute a circuit evaluation, let $E_i = w_i$ for $i = 0, \ldots, l-1$. Then for $i = l, \ldots, l+m-1$, let $G_i = (i_0, i_1)$ be the i'th gate structure and let $E_i = L_{i,E_{i_0},E_{i_0}}$. Notice that this is possible as, by definition, $L_{i,E_{i_0},E_{i_0}} = L_{4(i-l)+2E_0+E_1} = L_{P_i} = V_i$, and V_i is known.

We describe a technique known as circuit scrambling, which given a circuit logic L and an input w generates a new uniformly random circuit logic L' (for the same circuit structure) and a new uniformly random input w' with the only relation to L and w that $L'[w']_{l+m} = L[w]_{l+m}$.

Assume that we are given a circuit C with l input gates and m gates. Let L be the circuit logic. A circuit scrambler is a bit string $S \in \{0,1\}^{l+m}$. For an input $w \in \{0,1\}^l$, we define $w' = S\langle w \rangle \in \{0,1\}^l$ by $S\langle w \rangle_i = S_i \oplus w_i$ for $i = 1, \ldots, l$. For a circuit logic L we define $S\langle L \rangle \in \{0,1\}^{4m}$ by $S\langle L \rangle_{i,b_0,b_1} = L_{i,b_0 \oplus S_{i_0}, b_1 \oplus S_{i_1}} \oplus S_i$ for $i = l, \ldots, l+m-1$, $(b_0, b_1) \in \{0,1\} \times \{0,1\}$ and $G_i = (i_0, i_1)$ being the i'th gate structure. We make an essential observation:

Lemma 2.9 *For all circuit logics $L \in \{0,1\}^{4m}$, all inputs $w \in \{0,1\}^l$ and all circuit scrambler $S \in \{0,1\}^{l+m}$ we have that $S\langle L \rangle[S\langle w \rangle] = L[w] \oplus S$.*

Proof. For $i = 0, \ldots, l-1$ we have that $S\langle L \rangle[S\langle w \rangle]_i = S\langle w \rangle_i = S_i \oplus w_i = w_i \oplus S_i = L[w]_i \oplus S_i = (L[w] \oplus S)_i$. For $i = l, \ldots, l+m-1$ we have that

$$\begin{aligned}
S\langle L \rangle[S\langle w \rangle]_i &= S\langle L \rangle_{i, S\langle L \rangle[S\langle w \rangle]_{i_0}, S\langle L \rangle[S\langle w \rangle]_{i_1}} \\
&= S\langle L \rangle_{i, (L[w] \oplus S)_{i_0}, (L[w] \oplus S)_{i_1}} \\
&= S\langle L \rangle_{i, L[w]_{i_0} \oplus S_{i_0}, L[w]_{i_1} \oplus S_{i_1}} \\
&= L_{i, (L[w]_{i_0} \oplus S_{i_0}) \oplus S_{i_0}, (L[w]_{i_1} \oplus S_{i_1}) \oplus S_{i_1}} \oplus S_i \\
&= L_{i, L[w]_{i_0}, L[w]_{i_1}} \oplus S_i \\
&= L[w]_i \oplus S_i = (L[w] \oplus S)_i \ .
\end{aligned}$$

□

The above lemma gives us an alternative way to compute a circuit scrambling. Pick $E' \in \{0,1\}^{l+m}$ uniformly at random and let $S = E' \oplus L[w]$. Since E' is uniformly random, this gives a uniformly random scrambler S. Then let $E = S\langle L \rangle[S\langle w \rangle]$. Using the above lemma we have that $E = L[w] \oplus S = L[w] \oplus (E' \oplus L[w]) = E'$. I.e. $E' = S\langle L \rangle[S\langle w \rangle]$. In this process we first picked the value $E' = S\langle L \rangle[S\langle w \rangle]$ (without using w) and then (using w) we isolated the scrambler S. In the normal direction we first pick S (without using w) and then we compute $E' = S\langle L \rangle[S\langle w \rangle]$ (using w). This gives us a Σ-protocol where the prover commits to S and E', receives a challenge $e \in \{0,1\}$ and then based on e reveals either S or E'. The honest verifier simulator and the random bits faking algorithm will use that the simulator can always compute the one value out of S and E' which is to be revealed and that the other value can be computed when w becomes known. The details are given in Fig. 2.10 on the next page.

A PRS Σ-protocol for all of NP

prover inputs:
The prover has input $(x, w) \in R$.

verifier inputs:
The verifier has input x and randomness $e \in \{0, 1\}$.

reference string message:
The verifier computes $K \leftarrow gen(r)$ and sends K to the prover, where gen is the key generator for a statistically hiding commitment scheme.

commit message:
The prover computes:

1. $S \leftarrow_R \{0,1\}^{l+m-1} \times \{0\}$ and $C_S = \text{commit}_K(S; r_S)$.
2. $w' = S\langle w \rangle$ and $C_w = \text{commit}_K(w', r_w)$.
3. $L' = S\langle L \rangle$ and $C_{L,i} = \text{commit}_K(L'_i, r_{L,i})$ for $i = 0, \ldots, 4m-1$.
4. P': the w'-path in L'.

and sends $(C_S, C_w, \{C_{L,i}\}_{i=0}^{4m-1})$ to the verifier.

challenge message:
The verifier sends $e \in \{0, 1\}$ to the prover.

response message:

- If $e = 0$, then the prover sends (S, r_S) and $\{(L'_i, r_{L,i})\}_{i=0}^{4m-1}$ to the verifier.
- If $e = 1$, then the prover sends (w', r_w), P' and $(L'_{P'_i}, r_{L,P'_i})$ for $i = l, \ldots, l+m-1$ to the verifier.

verification:

- If $e = 0$, then the verifier accepts iff $C_S = \text{commit}_K(S, r_S)$, $c_{L,i} = \text{commit}_K(L'_i, r_{L,i})$ for $i = 0, \ldots, 4m-1$, $L' = S\langle L \rangle$ and $S_{l+m-1} = 0$.
- If $e = 1$, then the prover accepts iff $C_w = \text{commit}_K(w', r_w)$, $C_{L,P'_i} = \text{commit}_K(L'_{P'_i}, r_{L,P'_i})$ for $i = l, \ldots, l+m-1$, $E' = (L'_{P'_l}, \ldots, L'_{P'_{l+m-1}})$ is the evaluation for w'-path P' and $L'_{P'_{l+m-1}} = 1$.

honest verifier simulator:

- If $e = 0$, then compute:
  1. $S \leftarrow_R \{0,1\}^{l+m-1} \times \{0\}$ and $C_S = \text{commit}_K(S; r_S)$.
  2. $C_w = \text{commit}_K(?, r'_w)$.
  3. $L' = S\langle L \rangle$ and $C_{L,i} = \text{commit}_K(L'_i, r_{L,i})$ for $i = 0, \ldots, 4m-1$.

  and let $a = (C_S, C_w, \{C_{L,i}\}_{i=0}^{4m-1})$ and $z = ((S, r_S), \{(L'_i, r_{L,i})\}_{i=0}^{4m-1})$.

Figure 2.10(a) (cont. in Fig. 2.10(b) on the facing page): A PRS Σ-protocol for all of NP

A PRS Σ-protocol for all of NP

honest verifier simulator (cont.):

- If $e = 1$, then compute:
  1. $E' \in \{0,1\}^{l+m}$ uniformly at random.
  2. For $i = l, \ldots, l+m-1$, let $P'_i = 4(i-l) + 2E'_{i_0} + E'_{i_1}$, and let $P' = (P'_l, \ldots, P'_{l+m-1})$.
  3. For $i = l, \ldots, l+m-1$, let $L'_{P'_i} = E'_i$ and let $C_{L,P'_i} = \text{commit}_K(L'_{P'_i}, r_{L,P'_i})$.
  4. For $j = 0, \ldots, 4m-1$, where j is not one of the values P'_i for $i = l, \ldots, l+m-1$, let $C_{L,j} = \text{commit}_K(?, r'_{L,j})$.
  5. For $i = 0, \ldots, l-1$, let $w'_i = E'_i$, let $w' = (w'_0, \ldots, w'_{l-1})$ and let $C_w = \text{commit}_K(w', r_w)$.
  6. $C_S = \text{commit}_K(?; r'_S)$.

  and let $a = (C_S, C_w, \{C_{L,i}\}_{i=0}^{4m-1})$ and $z = ((w', r_w), P', \{(L'_{P'_i}, r_{L,P'_i})\}_{i=l}^{l+m-1})$.

random bits faking:

Given w such that $(x,w) \in R$, e, values as generated by the honest verifier above, and r such that $K = gen(r)$, continue as follows:

- If $e = 0$, then compute:
  1. $w' = S\langle w \rangle$ and use r and r'_w to compute r_w such that $C_w = \text{commit}_K(w', r_w)$.
  2. P': the w'-path in L'.

  This defines the remaining internal state of the prover.
- If $e = 1$, then compute:
  1. $S = E' \oplus L[w]$ and use r and r'_S to compute r_S such that $C_S = \text{commit}_K(S; r_S)$.
  2. $L' = S\langle L \rangle$ and for $j = 0, \ldots, 4m-1$, where j is not one of the values P_i for $i = l, \ldots, l+m-1$, use r and $r'_{L,j}$ to compute $r_{L,j}$ such that $C_{L,j} = \text{commit}_K(L'_j, r_{L,j})$.

  This defines the remaining internal state of the prover.

extraction:

Given x, $a = (C_S, C_w, \{C_{L,i}\}_{i=0}^{4m-1})$, $z_0 = ((S, r_S), \{(L'_{i,0}, r_{L,i,0})\}_{i=0}^{4m-1}$ and $z_1 = ((w', r_w), \{(L'_{P'_i,1}, r_{L,P'_i,1})\}_{i=l}^{l+m-1}\})$, where $C_S = \text{commit}_K(S, r_S)$, $c_{L,i} = \text{commit}_K(L'_{i,0}, r_{L,i,0})$ for $i = 0, \ldots, 4m-1$, $L'_0 = S\langle L \rangle$, $S_{l+m-1} = 0$ and $C_w = \text{commit}_K(w', r_w)$, $C_{L,P'_i} = \text{commit}_K(L'_{P'_i,1}, r_{L,P'_i,1})$ for $i = l, \ldots, l+m-1$, $E' = (L'_{P'_l,1}, \ldots, L'_{P'_{l+m-1},1})$ is the evaluation for w'-path P' and $L'_{P'_{l+m-1},1} = 1$, proceed as follows:

- If $L'_{P'_i,1} \neq L'_{P'_i,0}$ for some $i \in \{l, \ldots, l+m-1\}$, then output $w = \perp$.
- Otherwise, let $w = S\langle w' \rangle$.

Figure 2.10(b) (cont. from Fig. 2.10(a) on the facing page): A PRS Σ-protocol for all of NP

Lemma 2.10 *The protocol in Fig. 2.10 on page 46 is a PRS Σ-protocol for R.*

Proof. The correctness is straight forward to verify.

We verify the special non-erasure honest verifier zero-knowledge. When $e = 0$ the verification is straight forward using that the fake openings of the commitments are distributed statistically indistinguishable from real commitments: In the honest verifier simulator S and L' and the commitments C_S and $C_{L,i}$ were computed according to the protocol and when w is given in the random bits faking, w' and P' are computed according to the protocol and the fake commitment C_w is opened to w'. For the case $e = 1$ observe that for $S = E' \oplus L[w]$ and $S\langle L\rangle[S\langle w\rangle] = L[w] \oplus S$ (from Lemma 2.9 on page 45) it follows that S is uniformly random and $E' = S\langle L\rangle[S\langle w\rangle]$, from which it follows that S has the same distribution as in the protocol and all the values P'_i, $L'_{P'_i}$ and w' computed in the honest verifier simulator is computed as in the protocol when the scrambler S is used. Since the random bit faking algorithm simply finishes the protocol using S and opens the fake commitments accordingly, the claim follows.

Finally, for the computational special membership soundness, let $\mathcal{A}$ be any adversary for the game $\mathrm{SKS}_{\Sigma,gen,\mathcal{A}}(k,z)$. It is given (k,z,K) for $K \leftarrow gen(k)$ and, without loss of generality, outputs $(x,a,0,z_0,1,z_1)$. Assume that both $(x,a,0,z_0)$ and $(x,a,1,z_1)$ are accepting and let w denote the value computed in extraction in Fig. 2.10 on page 46. Let $p(k,z)$ denote the probability that $w = \perp$. When $w = \perp$, then $L'_{P'_i,1} \neq L'_{P'_i,0}$ for some $i \in \{l,\ldots,l+m-1\}$. This means that for some $i \in \{l,\ldots,l+m-1\}$ we have that $C_{L,P'_i} = \mathrm{commit}_K(L'_{P'_i,1}, r_{L,P'_i,1})$ and $C_{L,P'_i} = \mathrm{commit}_K(L'_{P'_i,0}, r_{L,P'_i,0})$ for $L'_{P'_i,1} \neq L'_{P'_i,0}$. In particular we have computed a double opening of C_{L,P'_i}. Let $\mathcal{A}'$ be an adversary which given (k,z,K) runs $\mathrm{SKS}_{\Sigma,gen,\mathcal{A}}(k,z)$ using (k,z,K) as input $\mathcal{A}$ and if $w = \perp$, then it outputs the double opening described. Otherwise, it outputs arbitrarily. Clearly $\Pr[\mathrm{DOUBLE}_{(gen,\mathrm{commit}),\mathcal{A}'} = 1] \geq p(k,z)$. Since (gen,commit) is computationally binding it follows that $\mathrm{DOUBLE}_{(gen,\mathrm{commit}),\mathcal{A}'} \overset{\mathrm{c}}{\approx} 0$ and therefore $p(k,z) \overset{\mathrm{c}}{\approx} 0$. We can therefore without loss of generality proceed the analysis under the assumption that $x = \perp$ does not happen. Assume therefore that we are given values as specified in extraction in Fig. 2.10 on page 46 and that $L'_{P'_i,1} = L'_{P'_i,0}$ for all $i \in \{l,\ldots,l+m-1\}$. We then have values $S \in \{0,1\}^{l+m}$, $w' \in \{0,1\}^l$, $L'_0 \in \{0,1\}^{4m}$ and P', where $L'_0 = S\langle L\rangle$, $S_{l+m-1} = 0$ and $E' = (L'_{P'_l,0},\ldots,L'_{P'_{l+m-1},0})$ is the w'-path logic and $L'_{P'_{l+m-1},0} = 1$. Let $w = S\langle w'\rangle$. This means that $w' = S\langle w\rangle$. Therefore $L'_0[w'] = S\langle L\rangle[S\langle w\rangle] = L[w] \oplus S$. Since $S_{l+m-1} = 0$, this in particular implies that $L'_0[w']_{l+m-1} = L[w]_{l+m-1}$. Since $(L'_{P'_l,0},\ldots,L'_{P'_{l+m-1},0})$ is the w'-path logic in L'_0 and $L'_{P'_{l+m-1},0} = 1$ it follows that $L'_0[w']_{l+m-1} = 1$. So, $L[w]_{l+m-1} = 1$. This in turn implies that $(x,w) \in R_k$. All in all we have that $\Pr[\mathrm{SKS}_{\Sigma,gen,\mathcal{A}}(k,z) = 1] \leq p(k,z)$ and $p(k,z) \overset{\mathrm{c}}{\approx} 0$, from which it follows that $\mathrm{SKS}_{\Sigma,gen,\mathcal{A}}(k,z) \overset{\mathrm{c}}{\approx} 0$, as desired. □

Theorem 2.4 *If there exists a statistically hiding trapdoor commitment scheme, then there exists a PRS Σ-protocol for all polynomial time relations.*

2.9.7 Computational Soundness and Private Reference Strings

One can also consider private reference string Σ-protocols with computational soundness relative to some PPT family of sets $\mathcal{I}$. In defining this notion the game $\mathrm{SKS}_{\Sigma,(\mathcal{I},\mathtt{prs}),\mathcal{A}}$ or $\mathrm{SMS}_{\Sigma,(\mathcal{I},\mathtt{prs}),\mathcal{A}}$ is played with $\mathcal{A}$ given (i,s), where $i \leftarrow gen$ and $s \leftarrow \mathtt{prs}$, and $\mathcal{A}$ must produce a break with an instance from $\mathcal{I}_i$. We will give an example of such a protocol in Section 2.9.9, but before we can do this we will need some additional tools presented in the section separating us from Section 2.9.9.

2.9.8 Monotone Logical Composition

In this section we describe how to make monotone logical compositions of PPT binary relations and non-erasure Σ-protocol. The constructions are straight-forward generalizations of constructions from [CDS94]. The constructions do not seem to appear in any published work, but their existence was claimed, and used, in [DN02b].

Definition 2.23 *Let $X = \{x_i\}$ be a set of variables. The set* $\mathrm{MBF}(X)$ *of* monotone Boolean formula *over X is the set of finite Boolean formulas over X for which: If $F \in \mathrm{MBF}(X)$, then either $F[X] = x_i$ for $x_i \in X$ or $F[X] = F_1[X] \vee F_2[X]$ for $F_1[X], F_2[X] \in \mathrm{MBF}[X]$ or $F[X] = F_1[X] \wedge F_2[X]$ for $F_1[X], F_2[X] \in \mathrm{MBF}[X]$. We use $\|F[X]\|$ to denote the* size of a monotone Boolean formula *$F[X]$. The size is defined by $\|x_i\| = 0$ and $\|F_1[X] \vee F_2[X]\| = \|F_1[X] \wedge F_2[X]\| = 1 + \|F_1[X]\| + \|F_2[X]\|$. A* monotone Boolean formula family *over X is a family $F[X] = \{F_k\}_{k\in\mathbf{N}}$, where $F_k[X] \in \mathrm{MBF}[X]$. We use $\|F[X]\|$ to denote the* size of a monotone Boolean formula family *$F[X]$. The size is defined to be the function $\mathbf{N} \to \mathbf{N}, k \mapsto \|F_k[X]\|$. A* polynomial monotone Boolean formula family *is a monotone Boolean formula family $F[X]$, where there exists a PPT machine, which on input k outputs the tree describing F_k according to the recursive definition of* $\mathrm{MBF}[X]$. *In particular $\|F[X]\|$ is upper bounded by a polynomial.*

Definition 2.24 *Let R_0 and R_1 be PPT binary relations. We define the* conjunction *respectively the* disjunction *of R_0 and R_1 by*

$$(R_0 \wedge R_1)((x_0,x_1),(w_0,w_1)) \equiv R_0(x_0,w_0) \wedge R_1(x_1,w_1) \ ,$$
$$(R_0 \vee R_1)((x_0,x_1),w) \equiv R_0(x_0,w) \vee R_1(x_1,w) \ .$$

Let $R = \{R_i\}$ be a set of PPT binary relations. Let X be a set of variables containing a symbol R_i for each $R_i \in R$. Let $F[X]$ be a monotone Boolean formula over X.[11] *By $F[R]$ we mean the PPT binary relation defined by $F[R] = R_i$ when $F[X] = R_i$ and $F[R] = F_1[R] \vee F_2[R]$ (respectively $F[R] = F_1[R] \wedge F_2[R]$) when $F[X] = F_1[X] \vee F_2[X]$ (respectively $F[X] = F_1[X] \wedge F_2[X]$). Let $F[X]$ be a polynomial monotone Boolean function family over the set X. We define the PPT binary relation $F[R]$ by $((k,x),w) \in F[R]$ iff $(x,w) \in F_k[R]$.*

Lemma 2.11 *Let $R = \{R_i\}$ be a finite set of PPT binary relations and let F be a polynomial monotone Boolean formula over R. Then $F[R]$ is a PPT binary relation.*

[11] Below we often call a monotone Boolean formula over X *a monotone Boolean formula over R.*

Proof. We describe a PPT algorithm A for evaluating $F[R]$. Given $((k,x),w)$ we can use the PPT machine for F to write out the tree describing F_k in PPT. We then parse x as $(x_1,\ldots,x_l)$ and parses w as $(w_1,\ldots,w_l)$ according to the description of the tree. For each leaf $F[R] = R_i$ we then compute $v_i = R_i(x,w)$. This can be done in PPT because R_i is PPT and because we consider a finite set of R_i, which allows us to make the code for evaluating each R_i part of A. Then the formula for F_k is evaluated in $(v_1,\ldots,v_l)$ using the tree description of F_k. This can also be done in polynomial time as the size of F is polynomial. □

Definition 2.25 *Let* $\Sigma_0 = (A_0, l_0, Z_0, B_0, \texttt{hvs}_0, \texttt{rbs}_0, \texttt{xtr}_0)$ *and* $\Sigma_1 = (A_1, l_1, Z_1, B_1, \texttt{hvs}_1, \texttt{rbs}_1, \texttt{xtr}_1)$ *be non-erasure* Σ*-protocols. We define the* conjunction *respectively the* disjunction *of* Σ_0 *and* Σ_1 *by Fig. 2.11 on the facing page and Fig. 2.12 on page 52. Let* $\Sigma = \{\Sigma_i\}$ *be a set of* Σ*-protocols and let* F *be a monotone Boolean formula over* Σ*. By* $F[\Sigma]$ *we denote the protocol defined by* $F[\Sigma] = \Sigma_i$ *for* $F = \Sigma_i$*,* $F[\Sigma] = F_1[\Sigma] \vee F_2[\Sigma]$ *(respectively* $F[\Sigma] = F_1[\Sigma] \wedge F_2[\Sigma]$*) for* $F = F_1 \vee F_2$ *(respectively* $F = F_1 \wedge F_2$*). Let* $F[X]$ *be a polynomial monotone Boolean function family over* Σ *and assume that* Σ *is finite. We define the* Σ*-protocol* $F[\Sigma]$ *were all algorithms* $\mathrm{A} \in \{A, Z, B, \texttt{hvs}, \texttt{rbs}, \texttt{xtr}\}$ *on input* $((k,x),w)$ *runs the corresponding algorithm* A_k *from* $F_k[\Sigma]$*.*

Theorem 2.5 *Let* Σ_0 *and* Σ_1 *be a (statistical) non-erasure* Σ*-protocols for relations* R_0 *respectively* R_1*. Then* $\Sigma_0 \vee \Sigma_1$ *is a (statistical) non-erasure* Σ*-protocol for relation* $R_0 \vee R_1$ *and* $\Sigma_0 \wedge \Sigma_1$ *is a (statistical) non-erasure* Σ*-protocol for relation* $R_0 \wedge R_1$*. And, if for* $b \in \{0,1\}$ *it holds that* Σ_b *have computational special soundness relative to* $\mathcal{I}$ *with private reference string* $\texttt{prs}$*, then* $\Sigma_0 \wedge \Sigma_1$ *and* $\Sigma_0 \vee \Sigma_1$ *have computational special soundness relative to* $\mathcal{I}$ *with private reference string* $\texttt{prs}$*.*

Proof. Following the proof of Lemma 2.11 on the preceding page we can prove that all algorithms of $F[\Sigma]$ are indeed PPT. The proof of the first part then follows by a straightforward extension of the proof techniques from [CDS94] to consider also the algorithms `hvs` and `rbs`.

We proceed to prove the second part of the lemma. For this purpose, let A be any adversary against the computational special soundness of $F[\Sigma]$ for relation $F[R]^{\texttt{prs}}$. Consider any value k for the security parameter and let l_k be the number of leafs in the tree for F_k. For each $j \in \{1,\ldots,l_k\}$, let Σ_j be the Σ-protocol for that leaf and let $R_j^{\texttt{prs}}$ be the relation for Σ_j. We describe how to construct from A an adversary A_j against the computational special soundness of Σ_j for relation $R_j^{\texttt{prs}}$. We define the adversary recursively. If $F = \Sigma_j$, then $A = A$. If $F = F_0 \vee F_1$, then proceed as follows: Pick $b \in \{0,1\}$ such that Σ_j occurs in F_b. We show how to construct an adversary $A_{\vee,F_b}$ against the computational soundness of $F_b[\Sigma]$ for relation $F_b[R]^{\texttt{prs}}$. Then an adversary against the computational soundness of Σ_j for the relation $R_j^{\texttt{prs}}$ can be constructed recursively from $A_{\vee,F_b}$. When $A_{\vee,F_b}$ is given (k,z) and $(i,s) \in (gen, \texttt{prs})$ it inputs these values to A and receives an output v. If v is a break of the computational special soundness of $F[\Sigma]$ for $F[R]$, then v is of the form $((x_0,x_1),(a_0,a_1),e,(z_0,z_1,e_0,e_1),e',(z_0',z_1',e_0',e_1'))$, where

Σ-Protocol $\Sigma_0 \vee \Sigma_1$

We define $(A, l, Z, B, \texttt{hvs}, \texttt{rbs}, \texttt{xtr}) = \Sigma_0 \vee \Sigma_1$ as follows:

- Let $l = \min\{l_0, l_1\}$.
- The algorithm $a = A((x_0, x_1), w)$ runs as follows: Pick the smallest $b \in \{0,1\}$ for which $(x_b, w) \in R$. Then compute $a_b \leftarrow A_b(x_b, w; r_b)$ and $(a_{1-b}, z_{1-b}) \leftarrow \texttt{hvs}(x_{1-b}, e_{1-b}; r_{\texttt{hvs},1-b})$ for uniformly random $e_{1-b} \in \{0,1\}^l$. Let $a = (a_0, a_1)$. The random bits used by A are $r_A = (r_{A,b}, e_{1-b}, r_{\texttt{hvs},1-b})$.
- The algorithm $z = Z(x, w, r_A, e)$ runs as follows: Let $x = (x_0, x_1)$ and pick the smallest $b \in \{0,1\}$ for which $(x_b, w) \in R$. Let $r_A = (r_{A,b}, e_{1-b}, r_{\texttt{hvs},1-b})$ and let $e_b = e \oplus e_{1-b}$. Then compute $z_b = Z_b(x_b, w, r_{A,b}, e_b)$ and let $z = (e_0, e_1, z_0, z_1)$.
- The algorithm $b = B(x, a, e, z)$ runs as follows: Let $x = (x_0, x_1)$, let $a = (a_0, a_1)$ and let $z = (e_0, e_1, z_0, z_1)$. Let $b = 1$ if $e = e_0 \oplus e_1$ and $B(x_0, a_0, e_0, z_0) = B(x_1, a_1, e_1, z_1) = 1$; Otherwise, let $b = 0$.
- The algorithm $(a, z) \leftarrow \texttt{hvs}(x, e)$ runs as follows: Pick $e_0 \in \{0,1\}^l$ uniformly at random and let $e_1 = e \oplus e_0$. Then compute $(a_b, z_b) \leftarrow \texttt{hvs}_b(x_b, e_b; r_{\texttt{hvs},b})$ for $b \in \{0,1\}$. Let $(a, z) = ((a_0, a_1), (e_0, e_1, z_0, z_1))$. The random bits used by $\texttt{hvs}$ are $r_{\texttt{hvs}} = (e_0, r_{\texttt{hvs},0}, r_{\texttt{hvs},1})$.
- The algorithm $r_A \leftarrow \texttt{rbs}(x, w, e, r_{\texttt{hvs}})$ runs as follows: Let $x = (x_0, x_1)$, let $r_{\texttt{hvs}} = (e_0, r_{\texttt{hvs},0}, r_{\texttt{hvs},1})$ and let $e_1 = e \oplus e_0$. Pick the smallest $b \in \{0,1\}$ for which $(x_b, w) \in R$ and compute $r_{A,b} \leftarrow \texttt{rbs}_b(x_b, w, e_b, r_{\texttt{hvs},b})$. Let $r_A = (r_{A,b}, e_{1-b}, r_{\texttt{hvs},1-b})$.
- The algorithm $w = \texttt{xtr}(x, a, e, z, e', z')$ runs as follows: Let $x = (x_0, x_1)$, let $a = (a_0, a_1)$, let $z = (e_0, e_1, z_0, z_1)$ and let $z' = (e'_0, e'_1, z'_0, z'_1)$. Since $e_0 \oplus e_1 = e \neq e' = e'_0 \oplus e'_1$ it follows that there exists $b \in \{0,1\}$ such that $e_b \neq e'_b$. Let $w = \texttt{xtr}(x_b, a_b, e_b, z_b, e'_b, z'_b)$.
- If Σ_0 and Σ_1 take a private reference strings, then so does $\Sigma_0 \vee \Sigma_1$, and it inputs *the same private reference string* to both Σ_0 and Σ_1.

Figure 2.11: The disjunction of two Σ-protocols.

$e \neq e'$, and for $b \in \{0,1\}$: $B(x_b, a_b, e_b, z_b) = 1$, $B(x_b, a_b, e'_b, z'_b) = 1$, and $e_0 \oplus e_1 = e$, $e'_0 \oplus e'_1 = e'$ and $(x_b, \texttt{xtr}(x_b, a_b, e_b, z_b, e'_b, z'_b)) \notin F_b[R]$ for $b \in \{0,1\}$. The adversary $A_{\vee,F_b}$ outputs $(x_b, a_b, e_b, z_b, e'_b, z'_b)$. Since $e_b \neq e'_b$ for at least one of the values $b \in \{0,1\}$ this means that $\mathrm{SKS}_{F_0[\Sigma],(\mathcal{I},\texttt{prs}),A_{\vee,F_0}}(k,z) + \mathrm{SKS}_{F_1[\Sigma],(\mathcal{I},\texttt{prs}),A_{\vee,F_1}}(k,z) \geq \mathrm{SKS}_{F[\Sigma],(\mathcal{I},\texttt{prs}),A}(k,z)$. When $F = F_1 \wedge F_1$ we can in the same way define from A adversaries $A_{\wedge,F_b}$ such that $\mathrm{SKS}_{F_0[\Sigma],(\mathcal{I},\texttt{prs}),A_{\wedge,F_0}}(k,z) + \mathrm{SKS}_{F_1[\Sigma],(\mathcal{I},\texttt{prs}),A_{\wedge,F_1}}(k,z) \geq \mathrm{SKS}_{F[\Sigma],(\mathcal{I},\texttt{prs}),A}(k,z)$. Traversing this recursively we get for each Σ_i an adversary A_i such that

$$\mathrm{SKS}_{F[\Sigma],(\mathcal{I},\texttt{prs}),A}(k,z) \leq \sum_{l=1}^{l_k} \mathrm{SKS}_{\Sigma_j,(\mathcal{I},\texttt{prs}),A_j}(k,z) \ . \tag{2.5}$$

Some of the Σ-protocols Σ_j in the leafs might be the same Σ-protocol, Σ_i say. We show how to collapse the adversaries for such Σ-protocols into one adversary for Σ_i. Let J_i be the set of indices j, where $\Sigma_j = \Sigma_i$. When A_{Σ_i} is given (k,z) and $(i,s) \in (gen, \texttt{prs})$ it inputs these

Σ-Protocol $\Sigma_0 \wedge \Sigma_1$

We define $(A, l, Z, B, \texttt{hvs}, \texttt{rbs}, \texttt{xtr}) = \Sigma_0 \wedge \Sigma_1$ as follows:

- Let $l = \min\{l_0, l_1\}$.
- The algorithm $a = A(x, w)$ runs as follows: Let $x = (x_0, x_1)$ and let $w = (w_0, w_1)$. For $b \in \{0, 1\}$, compute $a_b = A_b(x_b, w_b; r_{A,b})$. Let $a = (a_0, a_1)$.
- The algorithm $z = Z(x, w, r_A, e)$ runs as follows: Let $x = (x_0, x_1)$, let $w = (w_0, w_1)$ and let $r_A = (r_{A,0}, r_{A,1})$. For $b \in \{0, 1\}$, compute $z_b = Z(x_b, w_b, r_{A,b}, e)$.
- The algorithm $b = B(x, a, e, z)$ runs as follows: Let $x = (x_0, x_1)$, let $a = (a_0, a_1)$ and let $z = (z_0, z_1)$. Let $b = 1$ if $B(x_0, a_0, e, z_0) = B(x_1, a_1, e, z_1) = 1$; Otherwise, let $b = 0$.
- The algorithm $(a, z) \leftarrow \texttt{hvs}(x, e)$ runs as follows: Let $x = (x_0, x_1)$. For $b \in \{0, 1\}$ compute $(a_b, z_b) \leftarrow \texttt{hvs}_b(x_b, e; r_{\texttt{hvs},b})$. Let $(a, z) = ((a_0, a_1), (z_0, z_1))$. The random bits used by $\texttt{hvs}$ are $r_{\texttt{hvs}} = (r_{\texttt{hvs},0}, r_{\texttt{hvs},1})$.
- The algorithm $r_A \leftarrow \texttt{rbs}(x, w, e, r_{\texttt{hvs}})$ runs as follows: Let $x = (x_0, x_1)$, let $w = (w_0, w_1)$ and let $r_{\texttt{hvs}} = (r_{\texttt{hvs},0}, r_{\texttt{hvs},1})$. For $b \in \{0, 1\}$, compute $r_{A,b} \leftarrow \texttt{rbs}(x_b, w_b, e, r_{\texttt{hvs},b})$. Let $r_A = (r_{A,0}, r_{A,1})$.
- The algorithm $w = \texttt{xtr}(x, a, e, z, e', z')$ runs as follows: Let $x = (x_0, x_1)$, let $a = (a_0, a_1)$, let $z = (z_0, z_1)$ and let $z' = (z'_0, z'_1)$. For $b \in \{0, 1\}$, compute $w_b = \texttt{xtr}(x_b, a_b, e, z_b, e'_b, z'_b)$. Let $w = (w_0, w_1)$.
- If Σ_0 and Σ_1 take a private reference string, then so does $\Sigma_0 \wedge \Sigma_1$, and it inputs *the same private reference string* to both Σ_0 and Σ_1.

Figure 2.12: The conjunction of two Σ-protocols.

values to each of the adversaries A_j, for $j \in J_i$, and receives an output v_j from each. If any of the v_j is a break of the computational special soundness of Σ_i for R_i, then A_{Σ_i} outputs one of these breaks. We have that

$$\mathrm{SKS}_{\Sigma_i,(\mathcal{I},\texttt{prs}),A_{\Sigma_i}}(k, z) \geq \max\{\mathrm{SKS}_{\Sigma_j,(\mathcal{I},\texttt{prs}),A_j}(k, z)\}_{j \in J_i} \ . \tag{2.6}$$

For each Σ_i, let $\alpha_i = \|J_i\|$. By Eq. 2.6 we have that

$$\alpha_i \, \mathrm{SKS}_{\Sigma_i,(\mathcal{I},\texttt{prs}),A_{\Sigma_i}}(k, z) \geq \sum_{j \in J_i} \mathrm{SKS}_{\Sigma_j,(\mathcal{I},\texttt{prs}),A_j}(k, z) \ . \tag{2.7}$$

Using Eq. 2.5 on the preceding page and Eq. 2.7 we have that

$$\begin{aligned} \mathrm{SKS}_{F[\Sigma],(\mathcal{I},\texttt{prs}),A}(k, z) &\leq \sum_{l=1}^{l_k} \mathrm{SKS}_{\Sigma_j,(\mathcal{I},\texttt{prs}),A_j}(k, z) \\ &= \sum_{\Sigma_i \in \Sigma} \sum_{j \in J_i} \mathrm{SKS}_{\Sigma_j,(\mathcal{I},\texttt{prs}),A_j}(k, z) \\ &\leq \sum_{\Sigma_i \in \Sigma} \alpha_i \, \mathrm{SKS}_{\Sigma_i,(\mathcal{I},\texttt{prs}),A_{\Sigma_i}}(k, z) \ . \end{aligned} \tag{2.8}$$

Let $f_i(k,z) = \mathrm{SKS}_{\Sigma_i,(\mathcal{I},\mathtt{prs}),A_{\Sigma_i}}(k,z)$. Using the line of arguing in Lemma 2.11 on page 49 we can argue that A_{Σ_i} is PPT, so from the premise of the lemma we have that $f_i(k,z)$ is negligible. Since F is a polynomial formula we have that α_i is a polynomial. Therefore $\sum_{\Sigma_i \in \Sigma} \alpha_i(k) f_i(k,z)$ is negligible by Lemma 2.1 on page 11. Using Eq. 2.8 on the facing page this gives us that $\mathrm{SKS}_{F[\Sigma],(\mathcal{I},\mathtt{prs}),A}(k,z)$ is negligible, as desired. □

2.9.9 Example: Discrete Logarithms in RSA Groups, Revisited

In this section we describe a new Σ-protocol for proving equality of RSA discrete logarithms. Recall that the Σ-protocol in Section 2.9.4 could prove existence of an r such that $v_0^2 = u_0^{2r}$ and $v_1^2 = u_1^{2r}$; This, however, does not guaranteed that there exists r' such that $v_0 = u_0^{r'}$ and $v_1 = u_1^{r'}$. The goal of this section is to improve the Σ-protocol of Section 2.9.4 such that we can prove existence of r such that $v_0 = u_0^r$ and $v_1 = u_1^r$. We will need such a proof in Chapter 6.

Notice that if $v_0^2 = u_0^{2r}$ and $v_1^2 = u_1^{2r}$ and v_0, u_0, v_1 and u_1 are quadratic residues, then because squaring is a bijection in the group of quadratic residues in the RSA groups we consider, it follows that $v_0 = u_0^r$ and $v_1 = u_1^r$. We will exploit this fact in improving the Σ-protocol from Section 2.9.4. As a tool for improving the Σ-protocol we will therefore need a proof of quadratic residuosity in RSA groups. The very first zero-knowledge proof ever published was for proving RSA quadratic residuosity. It was published by Goldwasser, Micali and Rackoff in [GMR85]. Here we will however construct a new proof. The reason why we insist on constructing a new one is that the proof in [GMR85] communicates k^2 bits per proof and that we need a proof communicating only $O(k)$ bits to obtain some of the efficiency results in later chapters.

2.9.9.1 An Integer Commitment Scheme

We start by describing a statistically hiding commitment scheme by Fujisaki and Okamoto [FO97], which we will use to construct an Σ-protocol for proving quadratic residuosity in RSA groups.

Let $(p,q) \leftarrow gen'$ be an RSA generator as in Definition 2.21 on page 39. Let $N = pq$, generate uniformly $h \in \mathbf{Z}_N^*$ and let $g = h^\alpha \bmod N$ for uniformly random $\alpha \in \mathbf{Z}_{2^{2k}}$. This defines the key generator $(N,h,g) \leftarrow gen$ for the commitment scheme. The message space is $\mathbf{Z}$ and commitment function is given by

$$\mathrm{commit}_{N,h,g}(m;r) = g^m h^r \bmod N \ ,$$

for uniformly random $r \in \mathbf{Z}_{2^{2k}}$. Since $g^m \in \langle h \rangle$ and $h^r \bmod N$ is statistically close to uniformly random in $\langle h \rangle$ it follows that the scheme is statistically hiding. The following theorem is proved in [FO97]:

Theorem 2.6 *If the strong RSA assumption holds for gen', then the Fujisaki-Okamoto commitment scheme is a statistically hiding commitment scheme.*

Σ-Protocol `multiplicative relation between committed values`

A: Let $L = \lceil \log_2(B) + 3k/2 \rceil$. Choose uniformly random $y \in \mathbf{Z}_{2^L}$, $s_2 \in 2^{\lceil 7k/2 \rceil}$ $s_3 \in 2^{L+\lceil 7k/2 \rceil}$. Let $d_2 = g^y h^{s_2} \bmod N$ and $d_3 = c_1^y h^{s_3} \bmod N$ and let $a = (d_2, d_3)$.

I: The challenge length is $k/2 - 2$ bits.

Z: Let $u = y + 2ex_2$, $v_2 = s_2 + 2er_2$ and $v_3 = s_3 + 2e(r_3 - x_2 r_1)$. Then $z = (u, v_2, v_3)$.

B: Check that $c_1, c_2, c_3 \in \mathbf{Z}_N^*$ and $g^u h^{v_2} \equiv d_2 c_2^{2e} \pmod N$ and $c_1^u h^{v_3} = d_3 c_3^{2e} \pmod N$.

Figure 2.13: The Σ-protocol for proving a multiplicative relation between committed values in the Fujisaki-Okamoto scheme.

In [DF02] a Σ-protocol for proving a multiplicative relation between committed values is given. A related protocol was presented in [FO97], but no full proof of security for the scheme in [FO97] is known.

Definition 2.26 *The relation multiplicative relation between committed values is defined as follows: The instance language S is the set $\{(N, g, h)\}$, where $N = pq$ such that $p, q \in \text{PRIMES}(\lceil k/2 \rceil)$ and $(p-1)/2$ and $(q-1)/2$ are primes, $h \in \mathbf{Z}_N^*$ and $g \in \langle h \rangle$. The relation is given by $((N, g, h, c_1, c_2, c_3, B), (x_2, r_2, r_3)) \in R$ iff $c_1, c_2, c_3 \in \mathbf{Z}_N^*$, $x_2 \in [-B..B]$, $r_2 \in 2^{2k}$, $|r_3| \in 2^{2k+\lceil \log_2(B) \rceil}$ and $c_2 = g^{x_2} h^{r_2}$, and $c_3 = c_1^{x_2} h^{r_3}$. The Σ-protocol `multiplicative relation between committed values` is defined in Fig. 2.13.*

The following theorem is proved in [DF02]:

Theorem 2.7 *If the strong RSA assumption holds for gen′, then the Σ-protocol `multiplicative relation between committed values` has computational special knowledge soundness relative to $K = (N, g, h)$ being a random key for the Fujisaki-Okamoto commitment scheme.*

Notice that if an opening $c_1 = g^{x_1} h^{r_1} \bmod N$ is known, then given a witness for the relation in Definition 2.26 we have that $c_3 = g^{x_1 x_2} h^{r_1 x_2 + r_3}$. Notice that if in particular $c_1 = c_2$, then the witness contains an opening of c_1 as $c_1 = c_2 = g^{x_2} h^{r_2} \bmod N$. Therefore $c_3 = g^{x_2^2} h^{r_2 x_2 + r_3}$. We now describe how to use this to obtain a Σ-protocol for proving that an element $y \in \mathbf{Z}$ is a square modulo some integer. Because we need it for the modulus being an RSA modulus, we present the protocol in that setting.

2.9.9.2 A Σ-Protocol for Quadratic Residuosity in RSA Groups

Definition 2.27 *The relation RSA quadratic residuosity is defined as follows: The instance language S is the set $\{(N, y, s)\}$, where $N = pq$ such that $p, q \in \text{PRIMES}(\lceil k/2 \rceil)$ and $(p-1)/2$ and $(q-1)/2$ are primes. The relation is given by $((N, y, s), x) \in R$ iff $x, y \in \mathbf{Z}_{N^{s+1}}^*$ and $y = x^2 \bmod N^{s+1}$. The Σ-protocol `RSA quadratic residuosity` is defined as follows: The protocol is a private reference string Σ-protocol. The private reference will be a random key $K = (N', g, h)$ for the Fujisaki-Okamoto commitment scheme. Given input $((N, y, s), x)$,*

where $y = x^2 \bmod N^{s+1}$ *and a private reference string* $K = (N', g, h)$ *the prover (the algorithm A) computes* $q = (x^2 - y)/N^{s+1}$, $c_1 = g^x h^r \bmod N'$ *for* $r \in \mathbf{Z}_{2^{2k}}$, $c = g^q h^s \bmod N'$ *for* $s \in \mathbf{Z}_{2^{2k}}$ *and lets* $c_3 = g^y c^{N^{s+1}} \bmod N' = g^{y+qN^{s+1}} h^{sN^{s+1}} \bmod N' = g^{x^2} h^{sN^{s+1}} \bmod N'$. *Notice that* $c_1^x = g^{x^2} h^{rx}$. *Therefore* $c_3 = c_1^x h^{sN^{s+1}-rx} \bmod N'$. *So, let* $r_3 = sN^{s+1} - rx$ *and* $(K, c_1, c_1, c_3, 2^{(s+1)k})$ *is an instance for the proof of multiplicative relation between committed values with witness* (x, r, r_3). *The prover sends* (c_1, c) *along with the a message of the proof of multiplicative relation for* $(K, c_1, c_1, c_3, 2^{(s+1)k})$ *and then l, Z, B is just defined to complete this proof of multiplicative relation. Notice that the verifier can compute* $c_3 = g^y c^{N^{s+1}} \bmod N'$ *from c and the instance* (N, y, s) *for RSA quadratic residuosity. Using the* $\wedge$*-construction, at the same time the prover proves knowledge of m and s for which* $c = g^m h^s \bmod N'$ — *a* Σ*-protocol for this can be derived from the* Σ*-protocol* `multiplicative relation between committed values`.

Corollary 2.2 *If the strong RSA assumption holds for gen′, then the* Σ*-protocol* `RSA quadratic residuosity` *is a* Σ*-protocol for the relation* `RSA quadratic residuosity` *with statistical special honest verifier zero-knowledge and computational special knowledge soundness with a private reference string being a random key for the Fujisaki-Okamoto commitment scheme.*

Proof. Correctness is straight-forward to verify. As for the statistical special honest verifier zero-knowledge it is enough to observe that the commitment scheme is statistical hiding, so the proof can be simulated by doing a proof for some dummy witness x'. Notice that the protocol is *not* a non-erasure Σ-protocol. The reason for this is that the Fujisaki-Okamoto commitment scheme is not a trapdoor commitment scheme. For computational special knowledge soundness with a private reference string being a random key for the Fujisaki-Okamoto commitment scheme, observe that for this private reference string we have from Theorem 2.7 on the facing page that except with negligible probability a witnesses (x, r_1, r_3) and (m, s) can be extracted for which $c_1 = g^x h^{r_1} \bmod N'$ and $c_3 = g^{x^2} h^{r_1 x + r_3} \bmod N'$ and $c = g^m h^s \bmod N'$. We furthermore have that $c_3 = g^y c^{N^{s+1}} \bmod N' = g^{y+mN^{s+1}} h^{sN^{s+1}} \bmod N'$. Therefore $y = x^2 - mN^{s+1}$ or a double opening of c_3 was computed. Since our private reference string is a random commitment key, by Theorem 2.6 on page 53 we have that a double opening is computed only with negligible probability. Therefore we can extract $x \in \mathbf{N}$ such that $y = x^2 \bmod N^{s+1}$ except with negligible probability. □

2.9.9.3 An Improved Σ-Protocol for Discrete Logarithms in RSA Groups

In this section we present our improved Σ-protocol for equality of discrete logarithms in RSA groups. We can combine the Σ-protocols `equality of RSA discrete logarithms` and `RSA quadratic residuosity` to obtain a Σ-protocol for proving `equality of even RSA discrete logarithms`, where the 2 in $v^2 = u^{2r} \bmod N^{s+1}$ so to say is moved into the witness.

Definition 2.28 *The relation* even RSA discrete logarithm *is defined as follows: The instance language S is the set* $\{(N, B, s, u, v)\}$, *where* $N = pq$ *such that* $p, q \in \text{PRIMES}(\lceil k/2 \rceil)$ *and*

$(p-1)/2$ and $(q-1)/2$ are primes. The relation is given by $((N,B,s,u,v),r) \in R$ iff $u,v \in \mathbf{Z}^*_{N^{s+1}}$ and $r \in 2\mathbf{Z}_B$ and $v = u^r \bmod N^{s+1}$, where $2B$ is some bound on the size of r.

The relation equality of even RSA discrete logarithms is defined as follows: The instance language S is the set $\{(N,B,s,u_0,v_0,u_1,v_1)\}$, where N is as above. The relation is given by $((N,B,s,u_0,v_0,u_1,v_1),r) \in R$ iff $u_b, v_b \in \mathbf{Z}^*_{N^{s+1}}$ and $r \in 2\mathbf{Z}_B$ and $v_b = u_b^r \bmod N^{s+1}$ for $b \in \{0,1\}$.

The Σ-protocol `equality of even RSA discrete logarithms` is defined as follows: Given an instance witness pair $((N,B,s,u_0,v_0,u_1,v_1),r)$ run a proof `equality of RSA discrete logarithms` for the instance/witness pair $((N,B,s,u_0^2,v_0,u_1^2,v_1),r/2)$ and run two proofs of `RSA quadratic residuosity` with instance/witness pairs $((N,s,v_b),u_b^{r/2})$ for $b \in \{0,1\}$. These are combined with the $\wedge$-construction. A Σ-protocol for the relation `even RSA discrete logarithm` can be obtained by dropping all values indexed $b = 1$.

Notice that given a witness for the three proofs combined with the $\wedge$-construction in Definition 2.28 on the preceding page we get r', x_0, x_1 such that $v_b^2 = (u_b^2)^{2r'} \bmod N^{s+1}$ and $v_b = x_b^2 \bmod N^{s+1}$ for $b \in \{0,1\}$, from which it follows that $(x_b^2)^2 = (u_b^{2r'})^2 \bmod N^{s+1}$ for $b \in \{0,1\}$. Since $x_b^2 \in Q_{N^{s+1}}$ and $u_b^{2r'} \in Q_{N^{s+1}}$ and squaring is a bijection in $Q_{N^{s+1}}$, it follows that $x_b^2 = u_b^{2r'} \bmod N^{s+1}$ for $b \in \{0,1\}$. So, let $r = 2r'$ and use $v_b = x_b^2 \bmod N^{s+1}$ and we have that $v_b = u_b^r \bmod N^{s+1}$ for $r \in 2\mathbf{N}$, as desired. The following is a corollary to Lemma 2.8 on page 40, Corollary 2.2 on the preceding page and Theorem 2.5 on page 50.

Corollary 2.3 *If the strong RSA assumption holds for gen′, then the protocol `equality of even RSA discrete logarithms` (respectively `even RSA discrete logarithms`) is a Σ-protocol for the relation `equality of even RSA discrete logarithms` (respectively `equality of even RSA discrete logarithms`) with statistical special honest verifier zero-knowledge and computational special knowledge soundness relative to the instances being for random N and u_0 being a random quadratic residue (Lemma 2.8 on page 40) with the private reference string being a random key for the Fujisaki-Okamoto commitment scheme. The extraction relation has $B = \infty$.*

2.10 N-Invertible Group Homomorphisms

In this section we introduce the notion of N-invertible group homomorphisms. This will allow us to give a concise treatment of some common theory for some of complexity theoretic assumptions introduced in Section 2.7. The presentation here follows the one by Damgård and Nielsen in [DN02b], where N-invertible homomorphisms are used as a main ingredient in building universally composable commitments. We return to this in Chapter 9.

2.10.1 Definition

The following definition is given by Cramer and Damgård in [CD98].

Definition 2.29 *A group homomorphism generator gen is a PPT algorithm which on input k outputs (a description of) (G,H,f), where G and H are two finite Abelian groups G and H and $f : G \to H$ is a homomorphism of groups. We require that elements in G and H can*

be represented as k-bit strings, and that the group operation and inverses in G and H can be computed in PPT. Finally, f can be computed in PPT and a uniformly chosen element in G can be generated in PPT.

The following definition is a slight modification of definitions 2.2 and 2.3 in [CD98], the differences being that here N is generated by the group homomorphism generator and is not required to be a prime. Instead we require that we know a lower bound b' on the smallest primefactor of N. Also, we do not require the homomorphism to be one-way.

Definition 2.30 *An N-invertible group homomorphism generator is a PPT algorithm gen which on input k outputs (G, H, f, N, b') such that gen restricted to (G, H, f) is a group homomorphism generator and furthermore*

1. *b' is less than or equal to the smallest primefactor of N.*
2. *There exists a PPT algorithm which on input $(G, H, f, N, b') \in gen$, $y \in H$ computes $x' \in G$ such that $f(x') = y^N$.*

Let $F = f(G)$ and consider the factor group H/F. Let $[K]$ be an element from H/F with representative K and let $o = \mathrm{ord}_{H/F}([K])$ denote the order of $[K]$ in H/F. In the following, for $m \in \mathbf{Z}_o$ we let $[m]_K = K^m F$. I.e. $[0]_K, \ldots, [o-1]_K$ are the distinct cosets of the subgroup of H/F generated by $[K]$. Consider the following function $f_K : \mathbf{Z}_o \times G \to H$ given by $f_K(m, r) = K^m f(r)$.

Fact 2.2 *$f_K : \mathbf{Z}_o \times G \to H, (m, r) \to K^m f(r)$ is a homomorphism of Abelian groups.*

We are going to use these homomorphisms extensively for a variety of homomorphisms f. But, notice that we cannot necessarily compute efficiently in the group $\mathbf{Z}_o \times G$, as o, the order of K in the factor group, not necessarily is efficiently computable. We therefore look at another homomorphism. Consider the element $K \in H$. By N-invertibility we can compute $x' \in G$ such that $f(x') = K^N$. This means for all $K \in H$ we have that $K^N \in F$. In other words, the order of all elements in H/F divides N. So, consider the following function $f_K : \mathbf{Z}_N \times G \to H$ given by $f_K(m, r) = K^m f(r)$. Again we have that

Fact 2.3 *$f_K : \mathbf{Z}_N \times G \to H, (m, r) \to K^m f(r)$ is a homomorphism of Abelian groups.*

We know N from the description of the group homomorphism and so are guaranteed that we can compute efficiently in $\mathbf{Z}_N \times G$. By definition we have that $f_K(m, r) \in [m]_K$. If $m \in \mathbf{Z}_N$ and $r \in G$ and $c = f_K(m, r)$ we call (m, r) a K representation of c. Notice that all elements of the coset $[m]$ have $N/o \cdot |f^{-1}(1)|$ K representations, but all with $m \equiv m' \pmod{o}$. One of the reasons why we are so interested in N-invertible homomorphism is that we can construct efficient Σ-protocols for proving knowledge of K representations and proving linear and multiplicative relations between K representations, these Σ-protocols will be non-erasure if the domain of the homomorphism has invertible sampling. We investigate that in Section 2.10.4, but first we want some examples.

2.10.2 Examples of N-Invertible Group Homomorphisms

We give four examples of N invertible group homomorphisms.

Definition 2.31 *We define four group homomorphisms as follows:*

exponentiation:
Let $H = \langle \alpha \rangle$ be a group of prime order q and let $G = \{0\}$. Let $f(0) = \alpha$ and let $N = b' = q$.

Paillier:
Let $n = pq$ be generated as in Section 2.7.2. Let $G = \mathbf{Z}_n^*$, let $H = \mathbf{Z}_{n^2}^*$ and let $f(r) = r^n \bmod n^2$. Let $b' = 2^{\lfloor k/2 \rfloor - 1}$ and let $N = n$.

Okamoto-Uchiyama:
Let $N = p^2 q$ be generated as in Section 2.7.3. Let $G = \mathbf{Z}_N^*$, let $H = \mathbf{Z}_N^*$ and let $f(r) = r^N \bmod N$. Let $b' = 2^{\lfloor k/2 \rfloor - 1}$.

Diffie-Hellman:
Let $\langle \alpha \rangle$ be a group of prime order q. Let $\beta = \alpha^x$ for uniformly random $x \in \mathbf{Z}_q^*$. Let $G = \mathbf{Z}_q$, let $H = \langle \alpha \rangle \times \langle \alpha \rangle$, let $f(r) = (\alpha^r, \beta^r)$ and let $N = b' = q$.

Theorem 2.8 *The examples in Definition 2.31 are N-invertible group homomorphisms where the domains have invertible sampling.*

Proof. Consider first the Paillier group homomorphism. That f is a homomorphism of groups follows from Lemma 2.4 on page 22. Since p and q are $\lfloor k/2 \rfloor$-bit integers it follows that $b' < p, q$, which proves property 1 of Definition 2.30 on the page before. Property 2 of said definition follows by setting $x' = y$. That $G = \mathbf{Z}_N^*$ has invertible sampling follows from Theorem 2.3 on page 28.

Consider the Okamoto-Uchiyama group homomorphism. That f is a homomorphism of groups follows from Lemma 2.5 on page 23. Since p and q are $\lfloor k/2 \rfloor$-bit integers it follows that $b' < p, q$, which proves property 1 of Definition 2.30 on the preceding page. Property 2 of said definition follows by setting $x' = y$. That $G = \mathbf{Z}_N^*$ has invertible sampling follows from Theorem 2.3 on page 28.

Consider then the Diffie-Hellman group homomorphism. That f is a homomorphism of groups follows from $\mathrm{ord}(\alpha) = q$. Furthermore, for $(x, y) \in H$ we have that $(x, y)^q = (1, 1)$. So, for $x' = 0$ we have that $f(x') = y^N$, proving property 2 of Definition 2.30 on the preceding page. Property 1 of said definition follows from q being a prime. That $G = \mathbf{Z}_q$ has invertible sampling follows from Theorem 2.3 on page 28. The exponentiation based example follows using a proper subset of this line of argument. □

The reason for the slightly artificial exponentiation example is that a K representation of $c \in \langle \alpha \rangle$ is an element $d \in \mathbf{Z}_N = \mathbf{Z}_q$ such that $c = K^d f(1) = K^d$. This means that a K representation is a discrete logarithm and the Σ-protocols from Section 2.10.4 for proving linear and multiplicative relations between K representations therefore gives us Σ-protocols for proving linear and multiplicative relations between discrete logarithms.

2.10.3 Encrypting with Cosets

We use our last three examples of N-invertible homomorphisms in later chapters. One of the reasons that we are particularly interested in these examples is that they can encrypt with cosets, by which we mean the following: Let $K \in H$ and consider an encryption function of the form $E_K(m;r) = K^m f(r)$ for $m \in \mathbf{Z}_o$ and $o = \mathrm{ord}_{H/F}([K])$, and uniformly random $r \in G$. Clearly this is at least a unique encoding of m. We will also need decryption. For this purpose we assume that the generator outputs a trapdoor which allows to compute a K representations of elements $c \in H$. We require the generator to output an element $g \in H$ to use as key. Again we have to keep in mind that the order of g is not necessarily efficiently computable given just the description of f. Furthermore, if we publish $o = \mathrm{ord}_{H/F}(g)$ we might not be able to encrypt securely with cosets, as $|H/F|$ might be part of the trapdoor. An example of this is the Uchiyama-Okamoto homomorphism, where $|H/F| = p$. Therefore we assume that the generator publishes a bound b such that $b \leq |H/F|$. We can then encrypt as $E_g(m;r) = g^m f(r)$ for $m \in \mathbf{Z}_b$ and uniformly random $r \in G$. To be able to set up the encryption function such that a K representation always exists we will require that the generator gives us an element $g \in H$ such that $\langle g \rangle_{H/F} = H/F$. If G is not cyclic one could just work with a cyclic subgroup of G.

Definition 2.32 *We say that $(G, H, f, N, b', g, b, t) \leftarrow gen(k)$ is an N-invertible group homomorphism which can encrypt with cosets, if it is always the case that $\langle g \rangle = H/F$ and the below requirements hold. In the description we let $pk = (G, H, f, N, b', g, b)$ and we let $sk = (G, H, f, N, b', g, b, t)$.*

N-invertible:
(G, H, f, N, b') *is an N-invertible homomorphism.*

image indistinguishability:
Random elements from $F = f(G)$ cannot be distinguished from random elements from H given pk. We formalize this as follows: Let A be a TM, and consider the following game $\text{IND-IMG}^c_{gen,A}(k,z)$*, where $k \in \mathbf{N}$, $z \in \{0,1\}^*$ and $c \in \{0,1\}$: Run $(G, H, f, N, b', g, b, t) \leftarrow gen(k)$ and generate $y_0 \in H$ uniformly at random and compute $y_1 \leftarrow f(r)$ for uniformly random $r \in G$. Input $(k, z, (G, H, f, N, b', g, b), y_c)$ to A to receive a bit d. Output d. We require that* $\text{IND-IMG}^0_{gen,A} \overset{c}{\approx} \text{IND-IMG}^1_{gen,A}$ *for all PPT A.*

trapdoor property:
The trapdoor property comes in two flavors:

trapdoor image distinguishability:
Given sk and $c \in H$ one can compute in PPT whether $c \in F$.

trapdoor coset determination:
Given sk and $c \in H$, where $c = g^m f(r)$ for $m \in \mathbf{Z}_{\mathrm{ord}_{H/F}(g)}$ and $r \in G$, one can compute m in PPT.

If the homomorphism has trapdoor coset determination, then we can encrypt as $c = E_{pk}(m;r) = g^m f(r)$ for $m \in \mathbf{Z}_b$. If the homomorphism only has trapdoor image distinguishability, then we can only encrypt $m \in M$ for some small subset $M \subset \mathbf{Z}_b$: To decrypt

$c = g^m f(r)$ we can test whether $cg^{-m'} \in F$ for all $m' \in M$ until we hit $m' = m$. We now prove that these encryption functions are IND-CPA secure under the image indistinguishability assumption.

Lemma 2.12 *If $(G, H, f, N, b', g, b, t) \leftarrow gen(k)$ is an N-invertible group homomorphisms which can encrypt with cosets, then the function $E_{pk}(m; r) = g^m f(r)$, for $m \in \mathbf{Z}_N$ and uniformly random $r \in G$, is IND-CPA secure.*

Proof. Let A be any IND-CPA adversary against $E_{pk}(m; r) = g^m f(r)$. We describe an adversary $B(A)$ for the IND-IMG game: It receives (k, z, pk, y_c) from $\text{IND-IMG}^1_{gen,B}(k, z)$ and has to output a guess $w \in \{0, 1\}$. This is done as follows: It inputs (k, z, pk) to A to receive two values $m_0, m_1 \in \mathbf{Z}_b$ from A. Then it generates a uniformly random bit $e \in \{0, 1\}$ and gives $C = g^{m_e} y_c$ to A to receive a bit d. Then it outputs the guess $w = e \oplus d$. Observe that if $c = 0$, then y_c is a uniformly random element in H, so in the view of A the value C is independent of e and so is therefore d. Since e is uniformly random it therefore follows that $w = d \oplus e$ is uniformly random when $c = 0$, so $\Pr[\text{IND-IMG}^0_{gen,B} = 1] = \frac{1}{2}$. From the image indistinguishability of *gen* it then follows that $\Pr[\text{IND-IMG}^1_{gen,B} = 1] \overset{c}{\approx} \frac{1}{2}$. Consider then $c = 1$, i.e. consider $\text{IND-IMG}^1_{gen,B}$. Here $C = g^{m_e} y_c$ is a correctly distributed encryption of m_e, so $\Pr[\text{IND-IMG}^1_{gen,B}(k, z) = 1] = \frac{1}{2}\Pr[\text{IND}^{\mathsf{cpa},0}_{E,A}(k, z) = 1] + \frac{1}{2}(1 \oplus \Pr[\text{IND}^{\mathsf{cpa},1}_{E,A}(k, z) = 1]) = \frac{1}{2} + \frac{1}{2}(\Pr[\text{IND}^{\mathsf{cpa},0}_{E,A}(k, z) = 1] - \Pr[\text{IND}^{\mathsf{cpa},1}_{E,A}(k, z) = 1])$. Since $\Pr[\text{IND-IMG}^1_{gen,B}(k, z) = 1] \overset{c}{\approx} \frac{1}{2}$ it follows that $\frac{1}{2}(\Pr[\text{IND}^{\mathsf{cpa},0}_{E,A}(k, z) = 1] - \Pr[\text{IND}^{\mathsf{cpa},1}_{E,A}(k, z) = 1]) \overset{c}{\approx} 0$, which proves the lemma. □

Definition 2.33 *We define three candidates for being N-invertible homomorphisms which can encrypt with cosets by the following extensions of the N-invertible homomorphisms from Definition 2.31 on page 58:*

Paillier:
Let $b = N$, $g = (N + 1)$ and let $t = (p, q)$.

Okamoto-Uchiyama:
Let $b = 2^{\lfloor k/2 \rfloor - 1}$, let $g = (n + 1)r^N$ for uniformly random $r \in \mathbf{Z}^*_n$ and let $t = (p, q)$.

Diffie-Hellman:
Let $b = q$, let $g = (1, \beta)$ and let $t = x$.

Theorem 2.9 *The Paillier, the Okamoto-Uchiyama and the Diffie-Hellman group homomorphism generators, from Definition 2.33, are N-invertible group homomorphisms which can encrypt with cosets under the DCRA (Assumption 2.2 on page 23), the p-subgroup assumption (Assumption 2.3 on page 24) respectively the DDH assumption (Assumption 2.1 on page 21) in $\langle \alpha \rangle$. The Diffie-Hellman group homomorphism has trapdoor image distinguishability and the two other examples have trapdoor coset determination.*

Proof. The encryption function $g^m f(r) = (N + 1)^m r^N \bmod N^2$ is the Paillier homomorphism, so trapdoor coset determination and the fact that $N + 1$ generates H/F follows from

Lemma 2.4 on page 22. The image indistinguishability is by definition the DCRA (Assumption 2.2 on page 23).

For the Okamoto-Uchiyama example, the encryption function $g^m f(r) = (N+1)^m r^N \bmod N$ for uniformly random $r \in \mathbf{Z}_N^*$ is the Okamoto-Uchiyama homomorphism, so trapdoor coset determination follows from Lemma 2.5 on page 23. The image indistinguishability is by definition the p-subgroup assumption (Assumption 2.3 on page 24).

Consider then the Diffie-Hellman example. Since $x \neq 0$ and the order α is prime we have that the order of $\beta = \alpha^x$ is the same as the order of α. So, the order of $(1, \beta)$ is q. Since $|f(\mathbf{Z}_q)| = q$ it follows that $|(\langle\alpha\rangle \times \langle\alpha\rangle)/f(\mathbf{Z}_q)| = q$. So, $(1, \beta)$ generates $\langle\alpha\rangle \times \langle\alpha\rangle)/f(\mathbf{Z}_q)$ as required. The image indistinguishability is almost by definition the DDH assumption. For the trapdoor image distinguishability observe that $(A, B) \in f(G)$ iff $B = A^x$. □

2.10.4 Some Σ-Protocols for N-Invertible Homomorphisms

Let $F = f(G)$ and consider the factor group H/F. Let $[K]$ be an element from H/F with representative K and let $o = \mathrm{ord}_{H/F}([K])$ denote the order of $[K]$ in H/F. In the following, for $m \in \mathbf{Z}_o$ we let $[m]_K = K^o F$. I.e. $[0]_K, \ldots, [o-1]_K$ are the distinct cosets of H/F generated by $[K]$. Consider the following function $f_K : \mathbf{Z}_o \times G \to H$ given by $f_K(m, r) = K^m f(r)$. We clearly have that $f_K(m, r) \in [m]_K$. If $c = f_K(m, r)$ we called (m, r) a K representation of c. Below we construct a non-erasure Σ-protocol for proving knowledge of a K representation. An instance will be of the form $((G, H, f, N, b'), K, c)$ and a witness will be m and r such that $c = f_K(m, r)$. Since $\mathrm{ord}_{H/F}([K])$ is not necessarily computable from K we cannot in general ask the prover to also prove that $m \in \mathbf{Z}_{\mathrm{ord}_{H/F}([K])}$. Instead the prover will prove that $m \in \mathbf{Z}_N$.

Definition 2.34 *The relation known K representation for an N group homomorphism generator gen is defined as follows: The instance language S is the set $\{(G, H, f, N, b'), K, c\}$, where (G, H, f, N, b') was generated by gen and $K, c \in H$. The relation is given by $(((G, H, f, N, b'), K, c), (m, r)) \in R$ iff $m \in \mathbf{Z}_N$, $r \in G$ and $c = K^m f(r)$. In Fig. 2.14 on the next page the Σ-protocol* `known K representation` *is given.*

Lemma 2.13 *The protocol* `known K representation` *is a non-erasure Σ-protocol for the relation* `known K representation`*.*

Proof. For the correctness, do a direct computation:

$$\begin{aligned} K^{\tilde{m}} f(\tilde{r}) &= f(x')^{\tilde{i}} f(r)^e f(\overline{r}) \\ &= K^{\tilde{m}+\tilde{i}N} f(r)^e f(\overline{r}) \\ &= K^{em+\overline{m}} f(r)^e f(\overline{r}) \\ &= (K^m f(r))^e K^{\overline{m}} f(\overline{r}) \\ &= c^e \overline{c} \ . \end{aligned}$$

Special non-erasure honest verifier zero-knowledge follows from the observation that in both SIM and EXEC it is the case that $(\tilde{m}, \overline{m}, \tilde{r}, \overline{r})$ is a uniformly random element from $\mathbf{Z}_N \times$

Σ-Protocol `known` K `representation`

A: Pick $\overline{m} \in \mathbf{Z}_N$ and $\overline{r} \in G$ using invertible uniform samplers[a] and let $\overline{c} = K^{\overline{m}} f(\overline{r})$. Output $\overline{c}$.

I: Let $l = \lfloor \log_2(b') \rfloor$, where b' is the public bound on the smallest primefactor in N.

Z: Let $\tilde{m} = em + \overline{m} \bmod N$ and let $\tilde{i} = em + \overline{m} \operatorname{div} N$ (such that $em + \overline{m} = \tilde{m} + \tilde{i}N$). Using property 2 from Definition 2.30 on page 57, compute x' such that $f(x') = K^N$. Let $\tilde{r} = (x')^{\tilde{i}} r^e \overline{r}$. Output $(\tilde{m}, \tilde{r})$.

B: Output 1 iff $K^{\tilde{m}} f(\tilde{r}) = c^e \overline{c}$.

`hvs`: Given $c \in G$ and $e \in \{0,1\}^l$ pick $\tilde{m} \in \mathbf{Z}_N$ and $\tilde{r} \in G$ uniformly at random and let $\overline{c} = K^{\tilde{m}} f(\tilde{r}) c^{-e}$.

`rbs`: Given $m \in \mathbf{Z}_N$, $r \in G$, $c = K^m f(r)$, $e \in \{0,1\}^l$, $\tilde{m} \in \mathbf{Z}_N$, $\tilde{r} \in G$ and $\overline{c} = K^{\tilde{m}} f(\tilde{r}) c^{-e}$, let $\overline{m} = \tilde{m} - em \bmod N$, $\tilde{i} = em + \overline{m} \operatorname{div} N$, $\overline{r} = \tilde{r}(x')^{-\tilde{i}} r^{-e}$, where x' is given as in the description of Z. Then use the random-bits-faking algorithms to compute fake random bits for $\overline{m}$ and $\overline{r}$. Output these bits.

`xtr`: Given $(c, K, \overline{m}, \overline{c}, e, \tilde{m}, \tilde{r}, e', \tilde{m}', \tilde{r}')$, let $i = e - e'$. Using that $e, e' \in \mathbf{Z}_{b'}$ and $e \neq e'$ and property 1 from Definition 2.30 on page 57 we have that $\gcd(e - e', N) = 1$, so we can in PPT compute α and β such that $1 = \alpha N + \beta(e - e')$. Then using property 2 from Definition 2.30 on page 57 compute $r_c, r_K \in G$ such $f(r_c) = c^N$ and $f(r_K) = K^N$. Then compute $n = (\tilde{m} - \tilde{m}')\beta$, $s = (\tilde{r}(\tilde{r}')^{-1})^\beta r_c^\alpha$, $m = n \bmod N$, $r = s r_K^{n \operatorname{div} N}$. Output (m, r).

[a] See Theorem 2.3 on page 28 and Definition 2.29 on page 56 for the existence of these samplers.

Figure 2.14: A Σ-protocol for proving knowledge of a K representation.

$\mathbf{Z}_N \times H \times H$ for which $\tilde{m} = em + \overline{m} \bmod N$, $\tilde{i} = em + \overline{m} \operatorname{div} N$ and $\tilde{r} = \overline{r}(x')^{\tilde{i}} r^e$. Special knowledge soundness follows from a straight-forward verification of the description of `xtr`. □

Definition 2.35 *The relation linear relation between K representation for an N-invertible group homomorphism generator gen is defined as follows: The instance language S is the set*

$$\{(G, H, f, N, b'), K_1, c_1, \ldots, K_l, c_l, a_0, a_1, \ldots, a_l\} \ ,$$

where (G, H, f, N, b') was generated by gen and $K_i, c_i \in H$ for $i = 1, \ldots, l$ and $a_i \in \mathbf{Z}_N$ for $i = 0, 1, \ldots, l$. The relation is given by

$$(((G, H, f, N, b'), K_1, c_1, \ldots, K_l, c_l, a_0, a_1, \ldots, a_l), (m_1, r_1, \ldots, m_l, r_l)) \in R$$

iff $m_i \in \mathbf{Z}_N$, $r_i \in G$ and

$$c_i = K_i^{m_i} f(r_i)$$

for $i = 1, \ldots, l$ and

$$\sum_{i=1}^{l} a_i m_i \equiv a_0 \pmod{N} \ .$$

Σ-Protocol `linear relation between` K `representations`

An instance for the protocol is

$$((G, H, f, N, b'), K_1, c_1, \ldots, K_l, c_l, a_0, a_1, \ldots, a_l)$$

and a witness is of the form

$$(m_1, r_1, \ldots, m_l, r_l) \ .$$

The protocol proceeds by running in parallel l proofs of known K_i representation for c_i, with some additions to relate these proofs. For the proofs of known K_i representation, let Σ denote the Σ-protocol `known` K `representation`. We run the Σ-protocol $\wedge_{i=1}^{l}\Sigma$ with instance

$$(((G, H, f, N, b'), K_1, c_1), \ldots, ((G, H, f, N, b'), K_l, c_l))$$

and witness

$$((m_1, r_1), \ldots, (m_l, r_l)) \ .$$

In the following we index the values in the proof $\wedge_{i=1}^{l}\Sigma$ which naturally corresponds to instance (K_i, c_i) by i. E.g. $(\overline{m}_i, \overline{r}_i)$ is the i'th component of the value generated by A in $\wedge_{i=1}^{l}\Sigma$. We make the following additions to the proof $\wedge_{i=1}^{l}\Sigma$:

A: Send $\Delta = \sum_{i=1}^{l} a_i\overline{m}_i \bmod N$ along with $(\overline{m}_1, \overline{r}_1, \ldots, \overline{m}_l, \overline{r}_l)$.

Z: The algorithm Z is left unchanged from $\wedge_{i=1}^{l}\Sigma$.

B: Also check that $\sum_{i=1}^{l} a_i\tilde{m}_i \equiv ea_0 + \Delta \pmod N$.

`hvs`: After generating the $\tilde{m}_i$ as given in Fig. 2.14 on the facing page, let $\Delta = \sum_{i=1}^{l} a_i\tilde{m}_i - ea_0$.

`rbs`: No more randomness is used by the modifications, so `rbs` is unchanged.

`xtr`: A witness is still of the same form, so `xtr` is unchanged, except that we make sure to use the same β value in all l sub-instances. Note that this is already enforced if β is compute from $e - e'$ and N using a deterministic algorithm.

Figure 2.15: A Σ-protocol for proving a linear relation between K representations.

In Fig. 2.15 the Σ-protocol `linear relation between` K `representations` is given.

Lemma 2.14 *The protocol `linear relation between` K `representations` is a non-erasure Σ-protocol for the relation `linear relation between` K `representations`.*

Proof. Let R denote the relation `known` K `representation`. By Theorem 2.5 on page 50 we have that $\wedge_{i=1}^{l}\Sigma$ is a non-erasure Σ-protocol for relation $\wedge_{i=1}^{l}R$. We prove that the additions to $\wedge_{i=1}^{l}\Sigma$ guarantee correctness, special non-erasure honest-verifier zero-knowledge and special soundness for `linear relation between` K `representation`.

For the correctness, note that $\tilde{m}_i = em_i + \overline{m}_i \bmod N$ and $\sum_{i=1}^{l} a_i m_i \equiv a_0 \pmod N$. Therefore

$$\sum_{i=1}^{l} a_i\tilde{m}_i \equiv e\sum_{i=1}^{l} a_i m_i + \sum_{i=1}^{l} a_i\overline{m_i} \equiv ea_0 + \Delta \pmod N \ ,$$

as desired.

For special non-erasure honest-verifier zero-knowledge we have to check that the simulated value $\Delta = \sum_{i=1}^{l} a_i \tilde{m}_i - e a_0$ gives a Δ of the correct form. This follows from

$$\sum_{i=1}^{l} a_i \overline{m}_i \equiv \sum_{i=1}^{l} a_i (\tilde{m}_i - e m_i) \equiv \sum_{i=1}^{l} a_i \tilde{m}_i - e \sum_{i=1}^{l} a_i m_i \equiv (\Delta + e a_0) - e a_0 \equiv \Delta \ .$$

Notice that $(((G, H, f, N, b'), K_1, c_1, \ldots, K_l, c_l, a_0, a_1, \ldots, a_l), (m_1, r_1, \ldots, m_l, r_l))$ is in the relation `linear relation between` K `representation` iff $(((G, H, f, N, b'), K_1, c_1)(m_i, r_i))$ is in R for $i = 1, \ldots, l$ and $\sum_{i=1}^{l} a_i m_i = a_0 \bmod N$. To prove special soundness it is therefore enough to prove that `xtr` for $\wedge_{i=1}^{l} \Sigma$ produces $m_1, \ldots, m_l$ for which $\sum_{i=1}^{l} a_i m_i = a_0 \bmod N$. Notice that, by Fig. 2.14 on page 62, $m_i = (\tilde{m}_i + \tilde{m}'_i)\beta \bmod N$,[12] where

$$\sum_{i=1}^{l} a_i \tilde{m}_i \equiv e a_0 + \sum_{i=1}^{l} a_i \overline{m}_i \pmod{N}$$

and

$$\sum_{i=1}^{l} a_i \tilde{m}'_i \equiv e' a_0 + \sum_{i=1}^{l} a_i \overline{m}_i \pmod{N} \ .$$

Subtracting these two congruences we get that

$$\sum_{i=1}^{l} a_i (\tilde{m}_i - \tilde{m}'_i) \equiv (e - e') a_0 \pmod{N} \ .$$

I.e.

$$\sum_{i=1}^{l} a_i m_i \equiv (e - e') a_0 \beta \pmod{N} \ .$$

Since $(e - e')\beta = 1 - \alpha N$ it follows that

$$\sum_{i=1}^{l} a_i m_i \equiv a_0 \pmod{N} \ ,$$

as desired. □

Consider the exponentiation example of an N-invertible homomorphism. The Σ-protocol for proving linear relation with $a_0 = 0$, $a_1 = 1$ and $a_2 = -1$ proves knowledge of $m_1, m_2 \in \mathbf{Z}_q$ such that $c_1 = K_1^{m_1}$ and $c_2 = K_2^{m_2}$ and $m_1 \equiv m_2 \pmod{q}$. We denoted this relation by `equality of discrete logarithms` in Section 2.9.2 and gave a Σ-protocol for it by Chaum and Pedersen [CP92]. The Σ-protocol for proving linear relation can be seen as an extension of the protocol from [CP92]. We now present a further extension using ideas by Damgård and Jurik [DJ01].

[12] Here we used that the same β value was used in all sub-instances.

Σ-Protocol `multiplicative relation between` K `representations`

An instance for the protocol is

$$((G, H, f, N, b'), \{K_i\}_{i=1}^{l}, \{M_i\}_{i=1}^{l}, \{N_i\}_{i=1}^{l}, \{L_i\}_{j=1}^{m}, \{O_i\}_{j=1}^{m}, \{a_i\}_{i=1}^{l}, \{b_i\}_{i=1}^{l}, \{c_i\}_{j=1}^{m}) \ ,$$

and a witness is of the form

$$(\{(m_i, r_i)\}_{i=1}^{l}, \{(n_i, s_i)\}_{i=1}^{l}, \{(o_i, t_i)\}_{j=1}^{m}) \ .$$

The protocol proceeds by running in parallel $2l + m$ proofs of known K representation for each of the instances $M_i = K_i^{m_i} f(r_i)$ and $N_i = K_i^{n_i} f(s_i)$ for $i = 1, \ldots, l$ and $O_i = L_i^{o_i} f(t_i)$ for $j = 1, \ldots, m$ with some additional checks to relate these proofs.
In the following we index the values in the proof $\wedge_{i=1}^{2l+m}\Sigma$ which naturally corresponds to a given instance by i. E.g. $(\overline{m}_i, \overline{r}_i)$ is the first message sent for the instance (K_i, M_i).

A: In addition, send $\Delta = \sum_{i=1}^{l}(\overline{m}_i - b_i n_i \sum_{j=1}^{m} c_j \overline{o}_j) \bmod N$.

Z: Instead of $\tilde{m}_i$, send $\Delta_i = e a_i m_i + \overline{m}_i - b_i n_i \sum_{j=1}^{m} c_j \tilde{o}_j \bmod N$.
Let δ_i be such that $\Delta_i + N\delta_i = e a_i m_i + \overline{m_i} - b_i n_i \sum_{j=1}^{m} c_j \tilde{o}_j$.
Compute x_i such that that $f(x_i) = K_i^N$. Instead of $\tilde{r}_i$, send $\sigma_i = o_i^{-b_i \sum_{j=1}^{m}(c_j \tilde{o}_j)} x_i^{\delta_i} r_i^{e a_i} \overline{r}_i$.

B: Also check that $\Delta = \sum_{i=1}^{l} \Delta_i \bmod N$, and for $i = 1, \ldots, l$ that $M_i^{a_i e} \overline{M}_i = K_i^{\Delta_i} N_i^{b_i \sum_{j=1}^{m} c_j \tilde{o}_j} f(\sigma_i)$.

`hvs`: After generating the $\tilde{o}_i$ as given in Fig. 2.14 on page 62, pick $\Delta_1, \ldots, \Delta_l \in \mathbf{Z}_N$ at random at let $\Delta = \sum_{i+1}^{l} \Delta_i \bmod N$. Then let $\overline{M}_i = M_i^{-a_i e} K_i^{\Delta_i} N_i^{b_i \sum_{j=1}^{m} c_j \tilde{o}_j} f(\sigma_i)$ for uniformly random σ_i.

`rbs`: Given a witness compute o_i and s_i as in Fig. 2.14 on page 62. Then let $\overline{m}_i = \Delta_i - e a_i m_i + b_i n_i \sum_{j=1}^{m}(c_j \tilde{o}_j) \bmod N$. As $\overline{m}_i = \Delta_i - e a_i m_i + b_i n_i \sum_{j=1}^{m}(c_j \tilde{o}_j) \bmod N$ we can again compute δ_i such that $\Delta_i + N\delta_i = e a_i m_i + \overline{m_i} - b_i n_i \sum_{j=1}^{m}(c_j \tilde{o}_j)$; Let $\overline{r}_i = o_i^{b_i \sum_{j=1}^{m}(c_j \tilde{o}_i)} x_i^{-\delta_i} r_i^{-e a_i} \sigma_i$.

`xtr`: This algorithm is described in the proof of Lemma 2.15 on the next page.

Figure 2.16: A Σ-protocol for proving a multiplicative relation between K representations.

Definition 2.36 *The relation multiplicative relation between K representations for an N-invertible group homomorphism generator gen is defined as follows: The instance language S is the set of*

$$((G, H, f, N, b'), \{K_i\}_{i=1}^{l}, \{M_i\}_{i=1}^{l}, \{N_i\}_{i=1}^{l}, \{L_j\}_{j=1}^{m}, \{O_j\}_{j=1}^{m}, \{a_i\}_{i=1}^{l}, \{b_i\}_{i=1}^{l}, \{c_j\}_{j=1}^{m}) \ ,$$

where (G, H, f, N, b') was generated by gen and $K_i, M_i, N_i, L_j, O_j \in H$ for $i = 1, \ldots, l$ and $j = 1, \ldots, m$ and $a_i \in \mathbf{Z}_N^*$ and $b_i, c_i \in \mathbf{Z}_N$ for $i = 1, \ldots, l$ and $j = 1, \ldots, m$. A witness is of the form

$$(\{(m_i, r_i)\}_{i=1}^{l}, \{(n_i, s_i)\}_{i=1}^{l}, \{(o_i, t_i)\}_{j=1}^{m}) \ ,$$

and the relation is given by $m_i, n_i, o_j \in \mathbf{Z}_N$ *and* $r_i, s_i, t_j \in G$ *and*

$$M_i = K_i^{m_i} f(r_i), N_i = K_i^{m_i} f(s_i), O_i = L_i^{o_i} f(t_i)$$

for $i = 1, \ldots, l$ *and* $j = 1, \ldots, t$ *and*

$$\sum_{i=1}^{l} a_i m_i \equiv \sum_{i=1}^{l} b_i n_i \sum_{j=1}^{m} c_i o_i \pmod{N} .$$

In Fig. 2.16 on the preceding page the Σ*-protocol* `multiplicative relation between K representations` *is given.*[13]

Lemma 2.15 *The protocol* `multiplicative relation between K representations` *is a non-erasure* Σ*-protocol for the relation* `multiplicative relation between K representations`*.*

Proof. For the correctness we have that

$$\begin{aligned} M_i^{a_i e} \overline{M}_i &= K_i^{a_i(em_i + \overline{m}_i)} f(r_i^{ea_i} \overline{r}_i) \\ &= K_i^{\Delta_i + N\delta_i + b_i n_i \sum_{j=1}^{m} (c_j \tilde{o}_j)} f(r_i^{ea_i} \overline{r}_i) \\ &= K_i^{\Delta_i} (K_i^{n_i})^{b_i \sum_{j=1}^{m} (c_j \tilde{o}_j)} f(s_i^{b_i \sum_{j=1}^{m} (c_j \tilde{o}_j)}) f(o_i^{-b_i \sum_{j=1}^{m} (c_j \tilde{o}_j)} x_i^{\delta_i} r_i^{ea_i} \overline{r}_i) \\ &= K_i^{\Delta_i} N_i^{b_i \sum_{j=1}^{m} (c_j \tilde{o}_j)} f(\sigma_i) . \end{aligned}$$

$$\begin{aligned} \sum_{i=1}^{l} \Delta_i &= e \sum_{i=1}^{l} a_i m_i + \sum_{i=1}^{l} \left(\overline{m}_i - b_i n_i \sum_{j=1}^{m} c_j \tilde{o}_j \right) \bmod N \\ &= e \sum_{i=1}^{l} a_i m_i + \sum_{i=1}^{l} \left(\overline{m}_i - b_i n_i \sum_{j=1}^{m} c_j (e o_j + \overline{o}_j) \right) \bmod N \\ &= e \left(\sum_{i=1}^{l} a_i m_i - \sum_{i=1}^{l} b_i n_i \sum_{j=1}^{m} c_j o_j \right) + \sum_{i=1}^{l} \left(\overline{m}_i - b_i n_i \sum_{j=1}^{m} c_j \overline{o}_j \right) \bmod N \\ &= \Delta . \end{aligned}$$

For the special non-erasure honest verifier zero-knowledge, note that the values $\overline{m}$ and $\overline{r}$ are individually distributed as in the protocol — uniformly random — and $\Delta_i = ea_i m_i + \overline{m}_i - b_i n_i \sum_{j=1}^{m} (c_j \tilde{o}_j) \bmod N$ and $\sigma_i = o_i^{-b_i \sum_{j=1}^{m} (c_j \tilde{o}_j)} x_i^{\delta_i} r_i^{ea_i} \overline{r}_i$, as desired.

For the special knowledge soundness, assume that we have

$$\Delta = \sum_{i=1}^{l} \Delta_i \bmod N ,$$

[13] We note that it is essential that the a_i values are invertible modulo N.

$$\Delta = \sum_{i=1}^{l} \Delta_i' \bmod N \ ,$$

and for $i = 1, \ldots, l$ that

$$M_i^{a_i e} \overline{M}_i = K_i^{\Delta_i} N_i^{b_i \sum_{j=1}^{m} c_j \tilde{o}_j} f(\sigma_i) \ ,$$
$$M_i^{a_i e'} \overline{M}_i = K_i^{\Delta_i'} N_i^{b_i \sum_{j=1}^{m} c_j \tilde{o}_j'} f(\sigma_i) \ .$$

This implies that

$$\sum_{i=1}^{l} (\Delta_i - \Delta_i') \bmod N = 0 \ ,$$

and

$$M_i^{a_i(e-e')} = K_i^{\Delta_i - \Delta_i'} N_i^{b_i \sum_{j=1}^{m} c_j (\tilde{o}_j - \tilde{o}_j')} f(\sigma_i / \sigma_i') \ ,$$

so

$$M_i^{a_i} M_i^{-a_i \alpha N} = M_i^{a_i \beta (e-e')} = K_i^{(\Delta_i - \Delta_i')\beta} N_i^{b_i \sum_{j=1}^{m} c_j (\tilde{o}_j - \tilde{o}_j')\beta} f((\sigma_i / \sigma_i')^\beta) \ .$$

By definition $o_j = (\tilde{o}_j - \tilde{o}_j)\beta \bmod N$ and $a_j \in \mathbf{Z}_N^*$ and $N_i = K_i^{n_i} f(s_i)$. This allows us, using techniques from above, to compute r_i and γ such that

$$M_i = K_i^{\gamma((\Delta_i - \Delta_i')\beta + n_i b_i \sum_{j=1}^{m} c_j o_j) \bmod N} f(r_i) \ ,$$

where $a_i \gamma \bmod N = 1$. We can therefore take $m_i = \gamma((\Delta_i - \Delta_i')\beta + n_i b_i \sum_{j=1}^{m} c_j o_j) \bmod N$ and we have that $M_i = K_i^{m_i} f(r_i)$ as desired. Note that

$$\begin{aligned}
\sum_{i=1}^{l} a_i m_i \bmod N &= \sum_{i=1}^{l} \left((\Delta_i - \Delta_i')\beta + n_i b_i \sum_{j=1}^{m} c_j o_j \right) \bmod N \\
&= \left(\sum_{i=1}^{l} \Delta_i - \sum_{i=1}^{l} \Delta_i' \bmod N \right) \beta + \sum_{i=1}^{l} n_i b_i \sum_{j=1}^{m} c_j o_j \bmod N \\
&= (\Delta - \Delta)\beta + \sum_{i=1}^{l} n_i b_i \sum_{j=1}^{m} c_j o_j \bmod N \\
&= \sum_{i=1}^{l} n_i b_i \sum_{j=1}^{m} c_j o_j \bmod N \ ,
\end{aligned}$$

as desired. □

CHAPTER 3

Universally Composable Security of Synchronous Protocols

A creative artist works on his next composition because he was not satisfied with his previous one.
— Dimitri Shostakovich

3.1 Introduction

In this chapter we formalize our model of synchronous computation. Except for some simplifying differences in the formulation, the basic model presented in Sections 3.2 through 3.4 is the model of universally composable (UC) security by Canetti in [Can01a, Can01b], specialized to the synchronous setting. Following the introduction of the UC framework it has been further developed by Canetti and Krawczyk [CK02] and Canetti and Rabin [CR03]. Our presentation here contains elements from these works too.

There have been made several previous suggestions on how how to define formally the security of cryptographic protocols, see e.g. [GL90, MR91, Bea91, BCG93, HM00, Can00]. Common to them all is the idea that security means that the adversary's view can be *simulated* efficiently by a machine that has access to only those data that the adversary is entitled to know. This is an idea which originates in the seminal work by Goldwasser Micali and Rackoff [GMR85, GMR89] on zero-knowledge proof systems. There a proof system, a protocol where a prover convinces a verifier about the validity of a formal statement, is called zero-knowledge if the verifier only learns from a run of the protocol whether the statement is true or false. In particular the prover should not leak any evidence it has that the claim is true. If e.g. the prover and the verifier agree on a value n, the claim made by the prover could be that n is the product of two primes. In such a setting the evidence held by the prover would be primes p and q such that $n = pq$. The protocol being zero-knowledge will at least guarantee that if the verifier itself cannot factor n, then it cannot factor it either after having seen a proof from the prover that n is indeed the product of two primes. This

is defined by requiring that a simulator given only a statement and the promise that it is true should be able to simulate values which looks exactly like a conversation between a real prover and a verifier. This basically says that whatever the verifier is able to compute from the interaction with the prover, the verifier could have computed on its own without interacting with the prover.

All later definitions mentioned above build on this idea. They say that a protocol is secure if it is zero-knowledge relative to its task, i.e. a protocol is secure if the public view of its run can be reproduced given only the information one is entitled to. This information could e.g. be the length of the secret inputs or some input values which are not required to be kept secret, like the statement in a zero-knowledge proof. The difference between the definitions is then to which extend they guarantee this requirement. Security cannot be viewed in a vacuum. If we say that a protocol is secure or zero-knowledge relative to its task we should also say in which environments. What e.g. happens if several copies of the protocol are run, in sequence, in parallel or arbitrarily composed? Is the combined protocol still zero-knowledge relative to its task? The principle difference between previous definitions is in how complex an environment they guarantee that protocols are secure, and under under which types of composition their security is maintained. The UC framework is the first to guarantee that security is preserved under what is dubbed universal composition, which means that security is maintained no matter how protocols are composed, as long as they interact through message passing. In some sense this is the most general composition we can hope will preserve security. If in particular protocols were allowed to inspect each others internal state, it is clear that no general security preservation under composition could be guaranteed: Compose almost any protocol π with the protocol which reads the internal state of π and broadcasts that state to everyone who is interested, and you will have an insecure protocol.

The fact that security is maintained under universal composition by the UC framework is one of the prime reason why we use it. Furthermore, its model of computation is general enough to capture in one unified framework the security of all the problems that we are going study, primarily because the UC framework allows to capture the security of reactive protocols. This allows for a much easier modularization of the presentation of protocols and the analysis of their security. In [Yao82a], which is generally considered one of the founding papers of cryptographic protocol theory, Yao says that *It would be desirable to have a unified framework where all these applications can be related, and where common proof techniques can be developed for proving the security of protocols.* Arriving at the UC framework some 20 years later the author feels that this desire is fulfilled.

As mentioned, we make some simplifying changes to the framework. The simplifications are made for two reasons. First of all, all protocols presented in later chapters will be synchronous. Therefore we want to aim the description of the model at this setting; In particular there is little reason to first describe the asynchronous model in [Can01b] and then specialize that to the synchronous setting. Second, in Chapter 7 we address the problem of implementing the so-called broadcast model, which will require us to make non-trivial transformations on protocols; In doing this it turns out favorable to have an as simplistic as possible definition of what is a synchronous protocol. The current framework is however derived directly from the framework in [Can01b] and can be cast in that framework.

In Section 3.5 we present a new contribution to the UC framework. We present a formalization of what it means for a protocol to be secure only under some specific input-output pattern. The principle result is that we are able to formalize what it means for a polynomial time protocol to be secure under an input-output behavior which is *not* verifiable in polynomial time, or at all verifiable in finite time. Among other things, this will allow us to give the first formulation of universally composable zero-knowledge proof of *membership*. We discuss these results in greater detail in Section 3.5 and Section 3.8.3 when the formal details of the basic model have been presented.

3.2 Universally Composable Security of Synchronous Protocols

In this section we formalize our model of synchronous computation. Except for some simplifying differences in the formulation it is the model of universally composable (UC) security by Canetti in [Can01a, Can01b], specialized to the synchronous setting.

A protocol π consists of n parties $P_1, \ldots, P_n$, all interactive Turing machines (ITMs). The execution of a protocol takes place in the presence of an environment $\mathcal{Z}$, also an ITM, which supplies inputs to and receives outputs from the parties. The environment $\mathcal{Z}$ also models the adversary of the protocol, and so schedules the activation of the parties, corrupts parties adaptively and controls corrupted parties. Notice that we do not have an explicit adversary in this description. I.e. we use the approach from [Can01b, Definition 4].

In each round r each party P_i sends a message $m_{i,j,r}$ to each party P_j, including itself; The message $m_{i,i,r}$ is the state of P_i after round r, and $m_{i,i,0} = (k, r_i)$ will be the security parameter and the random bits used by P_i in the computation. We model open authenticated channels by showing the messages $\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}$ to $\mathcal{Z}$. This is a reasonable model of communication and can be realized in much weaker networks. We will not pursue this issue further, but refer to the paper by Canetti, Halevi and Herzberg [CHH00], which among other things discusses the issue of realizing a complete and authenticity network with message delivery in an incomplete and unauthenticated network. Furthermore, [CHH00] provides good pointers to related literature. Notice that we do not give $m_{i,i,r}$ to $\mathcal{Z}$. This models that the environment is not able to inspect the internal state of honest parties. In each round $\mathcal{Z}$ inputs a value $x_{i,r}$ to P_i and receives an output $y_{i,r}$ from P_i. We write the r'th activation of P_i as $(\{m_{i,j,r}\}_{j\in[n]}, y_{i,r}) = P_i(\{m_{j,i,r-1}\}_{j\in[n]}, x_{i,r})$. Notice that this simplified model does not allow to specify what is called immediate functionalities in [Can01b], i.e. functionalities which return an answer to the caller in the same round. Each call to the functionality will take at least one round; This will sometimes force us to formulate certain ideal functionalities slightly different than done in [Can01b], e.g. the ideal functionality for signatures in Section 3.8.5.

We model that the parties cannot reliably erase their state by giving r_i to $\mathcal{Z}$ when P_i is corrupted. In the following C will denote the set of corrupted parties and $H = [n] \setminus C$. In detail the real-life execution proceeds as in Fig. 3.1 on the next page. The result of the execution is the bit b output by $\mathcal{Z}$. We denote this bit by $\mathrm{REAL}_{\pi,\mathcal{Z}}(k, \overrightarrow{r}, z)$. This defines a random variable $\mathrm{REAL}_{\pi,\mathcal{Z}}(k, z)$, where we take $\overrightarrow{r}$ to be uniformly random, and in turn defines a Boolean distribution ensemble $\mathrm{REAL}_{\pi,\mathcal{Z}} = \{\mathrm{REAL}_{\pi,\mathcal{Z}}(k, z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Algorithm $\mathrm{REAL}_{\pi,\mathcal{Z}}(k, \overrightarrow{r}, z)$

initialization:

The input is k, random bits $\overrightarrow{r} = (r_1, \ldots, r_n, r_{\mathcal{Z}}) \in (\{0,1\}^*)^{n+1}$ and an auxiliary input $z \in \{0,1\}^*$. Set $r = 1$ and $C = \emptyset$. For $i, j \in [n]$, let $m_{i,j,0} = \epsilon$ and let $m_{i,i,0} = (k, r_i)$. Then input k, n, z and $r_{\mathcal{Z}}$ to $\mathcal{Z}$ and activate $\mathcal{Z}$.[a]

environment activation:

When $\mathcal{Z}$ is activated it outputs one of the following commands:

- $(\texttt{activate}, i, x_{i,r}, \{m_{j,i,r-1}\}_{j\in C})$ for $i \in H$;
- $(\texttt{corrupt}, i)$ for $i \in H$;
- $(\texttt{end round})$;
- $(\texttt{guess}, b)$ for $b \in \{0,1\}$.

Commands are handled as described below and the environment is then activated again. We require that all honest parties are activated between two $(\texttt{end round})$ commands. When a $(\texttt{guess}, b)$ command is given the execution stops.

party activation:

On $(\texttt{activate}, i, x_{i,r}, \{m_{j,i,r-1}\}_{j\in C})$, the values $\{m_{j,i,r-1}\}_{j\in H}$ were defined in the previous round; Add these to $\{m_{j,i,r-1}\}_{j\in C}$ and compute $(\{m_{i,j,r}\}_{j\in[n]}, y_{i,r}) = P_i(\{m_{j,i,r-1}\}_{j\in[n]}, x_{i,r})$. Input $\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}$ to $\mathcal{Z}$.

corrupt:

On $(\texttt{corrupt}, i)$, input r_i to $\mathcal{Z}$. Set $C = C \cup \{i\}$.

end round:

On $(\texttt{end round})$, input $\{y_{i,r}\}_{i\in H}$ to $\mathcal{Z}$.[b] Set $r = r + 1$.

[a] Notice that we give n to $\mathcal{Z}$ to let it know how many parties there are in the protocol. In the follow we often fail to mention the input n explicitly.

[b] The value $y_{i,r}$ was defined in party activation.

Figure 3.1: The real-life execution.

Since Fig. 3.1 hopefully seems rather simple we want to avoid the impression that some details were hidden, by specifying how certain properties are modeled. In particular, notice that in Fig. 3.1 we model a rushing environment, which sees the messages of honest parties before specifying the messages of corrupted parties; This is modeled by the fact that when $(\texttt{activate}, i, x_{i,r}, \{m_{j,i,r-1}\}_{j\in C})$ is output by $\mathcal{Z}$ the values $\{m_{j,i,r-1}\}_{j\in H}$ to be sent by the honest parties were already specified in the previous round and shown to the environment, see party activation. Notice furthermore that $\mathcal{Z}$ can corrupt in the midst of a round, as there is no restriction on the order of commands, except that all honest parties are activated between two `end round` commands, see environment activation.

The security of a protocol is defined using the notion of an ideal functionality $\mathcal{F}$. An ideal functionality is an ITM with n input tapes and n output tapes which we think of as being connected to n parties. The input-output behavior of the ideal functionality defines the desired input-output behavior of the protocol. To be able to specify protocols which are allowed to leak certain information $\mathcal{F}$ has a special output tape (SOT) on which it writes

Algorithm $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, \overrightarrow{r}, z)$

initialization:
The input is k, random bits $\overrightarrow{r} = (r_{\mathcal{F}}, r_{\mathcal{S}}, r_{\mathcal{Z}}) \in (\{0,1\}^*)^3$ and an auxiliary input $z \in \{0,1\}^*$. Set $r = 1$ and $C = \emptyset$. Provide $\mathcal{S}$ with $(k, r_{\mathcal{S}})$, provide $\mathcal{F}$ with $(k, r_{\mathcal{F}})$ and give k, n, z and $r_{\mathcal{Z}}$ to $\mathcal{Z}$ and activate $\mathcal{Z}$.

environment activation:
$\mathcal{Z}$ is defined exactly as in the real-life model, but now commands are handled by $\mathcal{S}$, as described below.

party activation:
On $(\texttt{activate}, i, x_{i,r}, \{m_{j,i,r-1}\}_{j\in C})$, input $(i, x_{i,r})$ to $\mathcal{F}$ and receive $v_{\mathcal{F}}$ on the SOT of $\mathcal{F}$. Input $(\texttt{activate}, i, v_{\mathcal{F}}, \{m_{j,i,r-1}\}_{i\in C})$ to $\mathcal{S}$ to generate a value $\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}$. Input $\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}$ to $\mathcal{Z}$.

corrupt:
On $(\texttt{corrupt}, i)$, input $(\texttt{corrupt}, i)$ on the SIT of $\mathcal{F}$ and receive a value $v_{\mathcal{F}}$ on the SOT of $\mathcal{F}$. Input $(\texttt{corrupt}, i, v_{\mathcal{F}})$ to $\mathcal{S}$ and receive a value r_i. Input r_i to $\mathcal{Z}$. Set $C = C \cup \{i\}$.

end round:
On (`end round`), input (`end round`) to $\mathcal{S}$ to generate $v_{\mathcal{F}}$. Input $(\texttt{activate}, v_{\mathcal{F}})$ to $\mathcal{F}$ to generate $\{y_{i,r}\}_{i\in[n]}$. Input $\{y_{i,r}\}_{i\in C}$ to $\mathcal{S}$ and input $\{y_{i,r}\}_{i\in H}$ to $\mathcal{Z}$. Set $r = r + 1$.

Figure 3.2: The ideal process.

this information. The ideal functionality also has the special input tape (SIT). The ideal functionality is given an input for the party P_i by inputting (i, x_i) to $\mathcal{F}$, where the first component i specifies the input tape and the second component x_i specifies the value written on that tape. Each time $\mathcal{F}$ is given an input for P_i it writes some value on the SOT modeling the part of the input which is not required to be kept secret. The ideal functionality can also receive the input $(\texttt{activate}, v)$ on the SIT in response to which it produces an output value for each party, by outputting $\{(i, y_i)\}_{i\in[n]}$ — the value y_i is written on the i'th output tape. The command $(\texttt{activate}, v)$ signals the end of a round. Among other things, the value v models the inputs from the corrupted parties. We often assume that v is parsed as $(\{(i, x_i)\}_{i\in C}, v')$, where C is the indices of the corrupted parties. This allows us to talk about the input of all parties, honest and corrupted, when describing functionalities. Notice that with the convention that $(\texttt{activate}, v)$ specifies the inputs of the corrupted parties we model a rushing environment, which in a given round sees the values sent be honest parties before it sends messages on behalf of corrupted parties. When a party P_i is corrupted $\mathcal{F}$ receives the input $(\texttt{corrupt}, i)$ on the SIT in response to which it produce some output $v_{\mathcal{F}}$, which is written on the SOT; This models information allowed to leak from P_i when it is corrupted. Unless stated explicitly we assume that $v_{\mathcal{F}}$ contains all the inputs x_i written on the i'th input tape during the execution along with all the outputs y_i written on the i'th output tape.

We say that a protocol π realizes an ideal functionality $\mathcal{F}$ if there exists an interface, also called simulator, $\mathcal{S}$ which given access to $\mathcal{F}$ can simulate (to an environment $\mathcal{Z}$) a run of π with the same input-output behavior. In the simulation $\mathcal{S}$ is given the commands from $\mathcal{Z}$

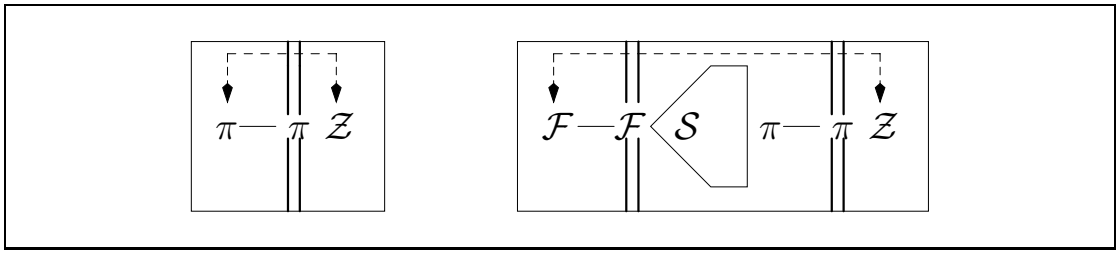

Figure 3.3: In the real-life execution $\mathrm{REAL}_{\pi,\mathcal{Z}}$, depicted to the left, the environment $\mathcal{Z}$ has access to a real copy of π. Its limited access to π is depicted by a wall. The double arrow between π and $\mathcal{Z}$ depicts the ability of $\mathcal{Z}$ to give inputs to the honest parties of π and to see their outputs. The symbol π occurring in the wall depicts the limited access that the environment has to the network of π: It can see all messages sent, it can corrupt a party and see its internal state and it can send messages on behalf of corrupted parties. In the ideal process $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, depicted to the right, the environment instead inputs to $\mathcal{F}$ and sees the outputs from $\mathcal{F}$. Then the interface $\mathcal{S}$ has access to $\mathcal{F}$, limited to inputting on behalf of corrupted parties and seeing the output on the SOT of $\mathcal{F}$. Using this access $\mathcal{S}$ produces a simulated version of π. The environment is then given access to this simulated version of π: It sees all simulated communication, it can corrupt parties and must receive a simulated internal state and it can send messages on behalf of corrupted parties. The interface $\mathcal{S}$ is said to produce π from $\mathcal{F}$ if no environment $\mathcal{Z}$ can determine whether it is in the left or the right setup.

and must then return to $\mathcal{Z}$ values looking as if coming from a real-life execution. In doing this $\mathcal{S}$ is only given the inputs of the corrupted parties, and the information leaked on the SOT of $\mathcal{F}$. The simulator being allowed to see only that information captures secrecy of the protocol: That the protocol is zero-knowledge relative to the information allowed to leaked by the specification of $\mathcal{F}$, we say that it is zero-knowledge relative to its task. Finally, $\mathcal{S}$ can input on the SIT of $\mathcal{F}$, and this is the only way that $\mathcal{S}$ can influence the outputs given by $\mathcal{F}$. This models the correctness of the protocol, that an adversary cannot force an erroneous output except too the extend allowed by the specification of $\mathcal{F}$. In detail the simulation, called the ideal process, proceeds as in Fig. 3.2 on the preceding page. The result of the ideal process is the bit b output by $\mathcal{Z}$. We denote this bit by $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, \overrightarrow{r}, z)$. This defines a random variable $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)$ and in turn defines a Boolean distribution ensemble $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} = \{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Notice that the interaction of $\mathcal{Z}$ with the real-life model and the ideal process has the same syntax. The goal of the interface $\mathcal{S}$ is then to produce the values that it hands to $\mathcal{Z}$ in such a way that $\mathcal{Z}$ cannot distinguish whether it is observing the real-life execution or a simulation of it in the ideal process. The idea is depicted in Fig. 3.3. If it can do this, then we say that it can produce π given $\mathcal{F}$. We write $\mathcal{S} : \mathcal{F} \triangleleft \pi$. We say that a protocol π realizes $\mathcal{F}$ if there exists a PPT interface $\mathcal{S}$ producing π given $\mathcal{F}$.

Definition 3.1 *Let $\mathcal{S}$ be an interface, let π be a protocol and let $\mathcal{F}$ be an ideal functionality. We say that π can produce π given $\mathcal{F}$, written $\mathcal{S} : \mathcal{F} \triangleleft \pi$, if for all PPT environments $\mathcal{Z}$ it holds that $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{\mathrm{c}}{\approx} \mathrm{REAL}_{\pi,\mathcal{Z}}$.*

Definition 3.2 *Let π be a protocol and let $\mathcal{F}$ be an ideal functionality. We say that π realizes*

$\mathcal{F}$ if there exists a PPT interface $\mathcal{S}$ such that $\mathcal{S} : \mathcal{F} \triangleleft \pi$.

3.3 The Hybrid Models

We now describe the synchronous $\mathcal{G}$-hybrid model for an ideal functionality $\mathcal{G}$. Basically the $\mathcal{G}$-hybrid model is the real-life model where in addition the parties have access to an ideal functionality $\mathcal{G}$ to aid them in the computation. In each round r party P_i will receive an output $t_{i,r-1}$ from $\mathcal{G}$ from round $r-1$ round and will produce an input $s_{i,r}$ for $\mathcal{G}$ for round r. This means that the r'th activation of P_i is now given by $(\{m_{i,j,r}\}_{j\in[n]}, y_{i,r}, s_{i,r}) = P_i(\{m_{j,i,r-1}\}_{j\in[n]}, x_{i,r}, t_{i,r-1})$. In the hybrid model, still $\mathcal{Z}$ models the adversary. Therefore, the output from $\mathcal{G}$ on the SOT, which models public information, is given to $\mathcal{Z}$, and the inputs to $\mathcal{G}$ on the SIT, which can be thought of as modeling the inputs from corrupted parties, is provided by $\mathcal{Z}$. In detail the hybrid execution with one ideal-functionality proceeds as in Fig. 3.4 on the following page. The extension to a constant number of functionalities is straight-forward, by simply splitting the s and t values into inputs to and outputs from the individual functionalities. We discuss this in more detail in a subsequent section. Since it is essential for the protocol which ideal-functionality it has access to, we consider $\mathcal{G}$ part of the protocol π; We use $\pi[\mathcal{G}]$ to denote a hybrid protocol with ideal functionality $\mathcal{G}$.

The result of the hybrid execution is the bit b output by $\mathcal{Z}$. We denote this bit by $\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}(k,r,z)$. This defines a random variable $\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}(k,z)$ and in turn defines a Boolean distribution ensemble $\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}$.

As for an interface $\mathcal{S}$ simulating a real-life execution of a protocol π in the ideal process for ideal functionality $\mathcal{F}$ we can define the notion of a hybrid interface $\mathcal{T}$ simulating a hybrid execution of a hybrid protocol $\pi[\mathcal{G}]$ in the ideal process for ideal functionality $\mathcal{F}$. This is defined equivalently. The only difference is that a hybrid interface $\mathcal{T}$ has to return more values to $\mathcal{Z}$ to be successful. For completeness we give the ideal process with a hybrid interface in detail in Fig. 3.5 on page 77. The symbol $\mathcal{G}$ occurring as a superfix in the symbol $\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\mathcal{Z}}(k,r,z)$ indicates that $\mathcal{T}$ should simulate for a hybrid model with $\mathcal{G}$. The exact value of $\mathcal{G}$ does not matter for how the hybrid execution $\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\mathcal{Z}}(k,r,z)$ is defined. The superfix is only used to distinguish the symbol $\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\mathcal{Z}}(k,r,z)$ from the symbol $\mathrm{IDEAL}_{\mathcal{F},\mathcal{T},\mathcal{Z}}(k,r,z)$ denoting a simulation of a real-life protocol.

Notice that the interaction of $\mathcal{Z}$ with the hybrid model and the ideal process has the same syntax. The goal of the hybrid interface $\mathcal{T}$ is then to produce the values that it hands to $\mathcal{Z}$ in such a way that $\mathcal{Z}$ cannot distinguish whether it is observing the hybrid execution or a simulation of it in the ideal process.

Definition 3.3 *Let $\mathcal{T}$ be a hybrid interface, let π be a protocol for the $\mathcal{G}$-hybrid model and let $\mathcal{F}$ be an ideal functionality. We say that $\mathcal{T}$ can produce $\pi[\mathcal{G}]$ given $\mathcal{F}$, written $\mathcal{T} : \mathcal{F} \triangleleft \pi[\mathcal{G}]$, if for all $\mathcal{G}$-hybrid environments $\mathcal{Z}$ it holds that $\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\mathcal{Z}} \overset{\mathrm{c}}{\approx} \mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}$.*

Definition 3.4 *Let π be a protocol for the $\mathcal{G}$-hybrid model and let $\mathcal{F}$ be an ideal functionality. We say that π realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if there exists a PPT $\mathcal{G}$-hybrid interface $\mathcal{T}$ such that $\mathcal{T} : \mathcal{F} \triangleleft \pi[\mathcal{G}]$.*

Algorithm $\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}(k,r,z)$

initialize:

The input is the security parameter k, the random bits $r_1,\ldots,r_n \in \{0,1\}^*$ used by the parties, the random bits $r_{\mathcal{G}}$ for $\mathcal{G}$ and an auxiliary input $z \in \{0,1\}^*$ and random bits $r_{\mathcal{Z}}$ for $\mathcal{Z}$.

Initialize the round counter $r = 1$ and initialize the set of corrupted parties $C = \emptyset$.

Let $m_{i,j,0} = \epsilon$ for $i,j \in [n]$, let $m_{i,i,0} = (k, r_i)$ $t_{i,-1} = \epsilon$.

Provide $\mathcal{G}$ with $r_{\mathcal{G}}$ and input k, n, z and $r_{\mathcal{Z}}$ to $\mathcal{Z}$ and activate $\mathcal{Z}$.

environment activation:

$\mathcal{Z}$ is defined exactly as in the real-life model except that the (`end round`) command has the syntax $(\texttt{end round}, v_{\mathcal{G}})$ for some value $v_{\mathcal{G}}$ and that $\mathcal{Z}$ receives some extra values in response to the commands as described below.

party activation:

On $(\texttt{activate}, i, x_{i,r}, \{m_{j,i,r-1}\}_{j\in C})$, values $\{m_{j,i,r-1}\}_{j\in H}$ and $t_{i,r-1}$ were defined in the previous round. Add these to $\{m_{j,i,r-1}\}_{j\in C}$ and compute $(\{m_{i,j,r}\}_{j\in[n]}, y_{i,r}, s_{i,r}) = P_i(\{m_{j,i,r-1}\}_{j\in[n]}, x_{i,r}, t_{i,r-1})$. Input $(i, s_{i,r})$ to $\mathcal{G}$ and receive $v_{\mathcal{G}}$ on the SOT of $\mathcal{G}$. Input $(\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}, v_{\mathcal{G}})$ to $\mathcal{Z}$.

corrupt:

On $(\texttt{corrupt}, i)$, input $(\texttt{corrupt}, i)$ on the SIT of $\mathcal{G}$ and receive a value $v_{\mathcal{G}}$ on the SOT of $\mathcal{G}$. Input $(r_i, v_{\mathcal{G}})$ to $\mathcal{Z}$. Set $C = C \cup \{i\}$.

end round:

On $(\texttt{end round}, v_{\mathcal{G}})$, input $(\texttt{end round}, v_{\mathcal{G}})$ to $\mathcal{G}$ to generate $\{t_{i,r}\}_{i\in[n]}$. Input $(\{y_{i,r}\}_{i\in H}, \{t_{i,r}\}_{i\in C})$ to $\mathcal{Z}$. The values $\{t_{i,r}\}_{i\in H}$ are used as input for the honest parties in the next round. Set $r = r + 1$.

Figure 3.4: The $\mathcal{G}$-hybrid model.

3.3.1 Multiple Ideal Functionalities

In this section we describe how to deal with hybrid models with multiple functionalities.

Definition 3.5 *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two ideal functionalities. The double ideal functionality $\mathcal{G} = [\mathcal{G}_1, \mathcal{G}_2]$ is defined as follows: Let m be a variable ranging over $\{1,2\}$. When initialized with $r_{\mathcal{G}}$ the ideal functionality $\mathcal{G}$ splits $r_{\mathcal{G}}$ into $(r^{\mathcal{G}_1}, r^{\mathcal{G}_2})$ and initializes $\mathcal{G}_m$ with $r^{\mathcal{G}_m}$. On input x_i to P_i the ideal functionality $\mathcal{G}$ parses x_i as $(x_i^{\mathcal{G}_1}, x_i^{\mathcal{G}_2})$ and inputs $x_i^{\mathcal{G}_m}$ to $\mathcal{G}_m$. Let $v^{\mathcal{G}_m}$ denote the value that was written on the SOT of $\mathcal{G}_m$. Then $\mathcal{G}$ outputs $v = (v^{\mathcal{G}_1}, v^{\mathcal{G}_2})$ on its SOT. On input $(\texttt{corrupt}, i)$ the ideal functionality $\mathcal{G}$ inputs $(\texttt{corrupt}, i)$ to $\mathcal{G}_m$. Let $v^{\mathcal{G}_m}$ denote the value that was written on the SOT of $\mathcal{G}_m$. Then $\mathcal{G}$ outputs $v = (v^{\mathcal{G}_1}, v^{\mathcal{G}_2})$ on its SOT. On input $(\texttt{activate}, v)$ the ideal functionality $\mathcal{G}$ parses v as $(v^{\mathcal{G}_1}, v^{\mathcal{G}_2})$ and inputs $(\texttt{activate}, v^{\mathcal{G}_m})$ to $\mathcal{G}_m$. Let $\{y_i^{\mathcal{G}_m}\}_{i\in[n]}$ denote the output values produced by $\mathcal{G}_m$. Let $y_i = (y_i^{\mathcal{G}_1}, y_i^{\mathcal{G}_2})$. Then $\mathcal{G}$ produces the output values $\{y_i\}_{i\in[n]}$. Let $\mathcal{G}_1,\ldots,\mathcal{G}_l$ be l ideal functionalities. The multi ideal functionality $\mathcal{G} = [\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_l]$ is defined as follows: If $l = 0$, we write $\mathcal{G} = []$, then we define $\mathcal{G}$ to be the trivial ideal functionality $\mathcal{F}_{\mathrm{triv}}$ which ignores all inputs from the parties and all inputs on its SIT, and which always outputs $x_i = \epsilon$ to P_i*

Algorithm $\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\mathcal{Z}}(k,r,z)$

initialize:

The input to an ideal process is the security parameter k, the random bits $r_{\mathcal{F}}$ and $r_{\mathcal{T}}$ used by $\mathcal{F}$ and $\mathcal{T}$ and an auxiliary input $z \in \{0,1\}^*$ and random bits $r_{\mathcal{Z}}$ for $\mathcal{Z}$.

Initialize the round counter $r = 1$ and initialize the set of corrupted parties $C = \emptyset$.

Provide $\mathcal{T}$ with $r_{\mathcal{T}}$, provide $\mathcal{F}$ with $r_{\mathcal{F}}$ and give k, z and $r_{\mathcal{Z}}$ to $\mathcal{Z}$ and activate $\mathcal{Z}$.

environment activation:

$\mathcal{Z}$ is defined exactly as in the hybrid model, but now the commands are handled by $\mathcal{T}$, as described below.

party activation:

On $(\texttt{activate}, i, x_{i,r}, \{m_{j,i,r-1}\}_{j\in C})$, input $(i, x_{i,r})$ to $\mathcal{F}$ and receive $v_{\mathcal{F}}$ on the SOT of $\mathcal{F}$. Input $(\texttt{activate}, i, v_{\mathcal{F}}, \{m_{j,i,r-1}\}_{i\in C})$ to $\mathcal{T}$ to generate $(\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}, v_{\mathcal{G}})$. Input $(\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}, v_{\mathcal{G}})$ to $\mathcal{Z}$.

corrupt:

On $(\texttt{corrupt}, i)$, input $(\texttt{corrupt}, i)$ on the SIT of $\mathcal{F}$ and receive a value $v_{\mathcal{F}}$ on the SOT of $\mathcal{F}$. Input $(\texttt{corrupt}, i, v_{\mathcal{F}})$ to $\mathcal{T}$ and receive a value $(r_i, v_{\mathcal{G}})$. Input $(r_i, v_{\mathcal{G}})$ to $\mathcal{Z}$. Set $C = C \cup \{i\}$.

end round:

On $(\texttt{end round}, v_{\mathcal{G}})$, input $(\texttt{end round}, v_{\mathcal{G}})$ to $\mathcal{T}$ to generate $v_{\mathcal{F}}$. Input $(\texttt{activate}, v_{\mathcal{F}})$ to $\mathcal{F}$ to generate $\{y_{i,r}\}_{i\in[n]}$. Input $\{y_{i,r}\}_{i\in C}$ to $\mathcal{T}$ to generate $\{t^{\mathcal{G}}_{i,r}\}_{i\in C}$. Then input $(\{y_{i,r}\}_{i\in H}, \{t^{\mathcal{G}}_{i,r}\}_{i\in C})$ to $\mathcal{Z}$. Set $r = r + 1$.

Figure 3.5: The ideal process with a hybrid-adversary.

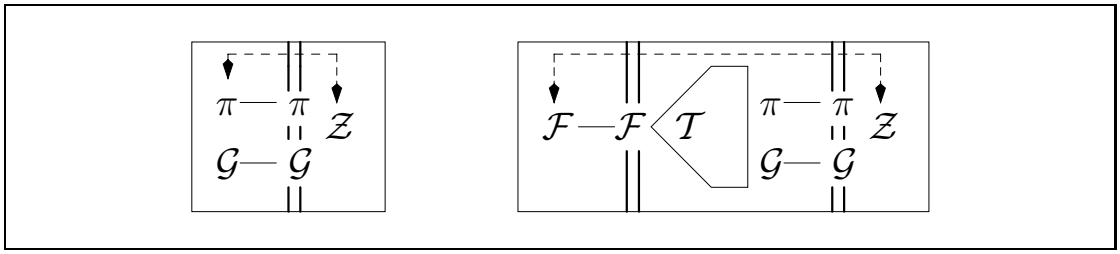

Figure 3.6: In the hybrid model $\text{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}$, depicted to the left, the environment $\mathcal{Z}$ has access to a real copy of π running with $\mathcal{G}$. In addition to the real-life execution $\mathcal{Z}$ can now input to $\mathcal{G}$ on behalf of corrupted parties and sees the output on the SOT of $\mathcal{G}$. In the ideal process $\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\mathcal{Z}}$, depicted to the right, the interface $\mathcal{T}$ has access to $\mathcal{F}$. Using this access $\mathcal{T}$ produces a simulated version of π running with $\mathcal{G}$. The environment is then given access to this simulated version of π running with $\mathcal{G}$: It sees all simulated communication, it can corrupt parties and must receive a simulated internal state, it can send messages on behalf of corrupted parties and it can input on the SIT of the simulated $\mathcal{G}$ and see the simulated output on the SOT. The interface $\mathcal{T}$ is said to produce $\pi[\mathcal{G}]$ from $\mathcal{F}$ if no environment $\mathcal{Z}$ can determine whether it is in the left or the right setup.

and always outputs ϵ on the SOT; If $l = 1$, then we let $\mathcal{G} = \mathcal{G}_1$; If $l \geq 2$, then we let $\mathcal{G} = [\mathcal{G}_1, [\mathcal{G}_2, \ldots, \mathcal{G}_l]]$.

Definition 3.6 *Let $\mathcal{F}$, $\mathcal{G}_1, \ldots, \mathcal{G}_l$ be ideal functionalities and let π be a hybrid protocol. We say that π realizes $\mathcal{F}$ in the $(\mathcal{G}_1, \ldots, \mathcal{G}_l)$-hybrid model if π realizes $\mathcal{F}$ in the $[\mathcal{G}_1, \ldots, \mathcal{G}_l]$-hybrid model.*

Definition 3.7 *Let $\mathcal{T}$ be a hybrid interface, let $\mathcal{F}$, $\mathcal{G}_1, \ldots, \mathcal{G}_l$ be ideal functionalities and let π be a hybrid protocol. We say that $\mathcal{T}$ can produce $\pi[\mathcal{G}_1, \ldots, \mathcal{G}_l]$ given $\mathcal{F}$, written $\mathcal{T} : \mathcal{F} \triangleleft \pi[\mathcal{G}_1, \ldots, \mathcal{G}_l]$ if $\mathcal{T} : \mathcal{F} \triangleleft \pi[([\mathcal{G}_1, \ldots, \mathcal{G}_l])]$.*

The following fact is trivial to prove and will come in handy at some points as it allows us to consider only double ideal functionalities in many cases and allows us to disregard the order in which ideal functionalities are composed.

Lemma 3.1 *Let $\mathcal{F}$, $\mathcal{G}_1, \mathcal{G}_2$ be ideal functionalities and let π be a protocol for the $[\mathcal{G}_1, \mathcal{G}_2]$-hybrid model. We can consider π a protocol for the $[\mathcal{G}_2, \mathcal{G}_1]$-hybrid model by considering the protocol π which simply lets $s_i = (s_i^{\mathcal{G}_2}, s_i^{\mathcal{G}_1})$, when π would have $s_i = (s_i^{\mathcal{G}_1}, s_i^{\mathcal{G}_2})$, and lets $t_i = (t_i^{\mathcal{G}_2}, t_i^{\mathcal{G}_1})$ when π would have $t_i = (t_i^{\mathcal{G}_1}, t_i^{\mathcal{G}_2})$. With this convention we have that π realizes $\mathcal{F}$ in the $[\mathcal{G}_1, \mathcal{G}_2]$-hybrid model iff π realizes $\mathcal{F}$ in the $[\mathcal{G}_2, \mathcal{G}_1]$-hybrid model. Let $\mathcal{F}$, $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ be ideal functionalities and let π be a protocol for the $[\mathcal{G}_1, [\mathcal{G}_2, \mathcal{G}_3]]$-hybrid model. We can consider π as a protocol for the $[[\mathcal{G}_1, \mathcal{G}_2], \mathcal{G}_3]$-hybrid model by considering the protocol π which simply lets $s_i = ((s_i^{\mathcal{G}_1}, s_i^{\mathcal{G}_2}), s_i^{\mathcal{G}_3}))$, when π would have $s_i = (s_i^{\mathcal{G}_1}, (s_i^{\mathcal{G}_2}, s_i^{\mathcal{G}_3}))$, and which lets $t_i = ((t_i^{\mathcal{G}_1}, t_i^{\mathcal{G}_2}), t_i^{\mathcal{G}_3}))$, when π would have $t_i = (t_i^{\mathcal{G}_1}, (t_i^{\mathcal{G}_2}, t_i^{\mathcal{G}_3}))$. With this convention π realizes $\mathcal{F}$ in the $[\mathcal{G}_1, [\mathcal{G}_2, \mathcal{G}_3]]$-hybrid model iff π realizes $\mathcal{F}$ in the $[[\mathcal{G}_1, \mathcal{G}_2], \mathcal{G}_l]$-hybrid model.*

3.3.1.1 Forget the Real-Life Model

By now we have four models of computation: The real-life model, the ideal process, the hybrid model and the hybrid ideal process. To prune the framework we dispense of the two first models, by observing that they are special cases of the last two. It is straight-forward to prove the following theorem.

Theorem 3.1 *Let $\mathcal{F}$ be an ideal functionality and let $\pi = (P_1, \ldots, P_n)$ be a real-life protocol. We can consider π a hybrid protocol by simply letting P_i ignore $t_{i,r-1}$ and always letting $s_{i,r} = \epsilon$. With this convention it holds that π realizes $\mathcal{F}$ (in the real-life model) iff π realizes $\mathcal{F}$ in the $()$-hybrid model.*

Proof. By definition π realizes the protocol π in the $()$-hybrid model iff π realizes $\mathcal{F}$ in the $\mathcal{F}_{\text{triv}}$-hybrid model. Since $\mathcal{F}_{\text{triv}}$ ignores all inputs and have a constant output it is trivial to simulate. Furthermore π will have the exact same input-output behavior in the real-life model and the $\mathcal{F}_{\text{triv}}$-hybrid model. □

This basically allows us to consider the $()$-hybrid model as the real-life model. We can easily extend the result as follows to allow removing trivial ideal functionalities from a hybrid model:

Theorem 3.2 *Let $\mathcal{F}$, $\mathcal{G}_1, \ldots, \mathcal{G}_l$ ideal functionalities and let $\pi = (P_1, \ldots, P_n)$ be a hybrid protocol for the $(\mathcal{G}_1, \ldots, \mathcal{G}_l)$-hybrid model. We can consider π a hybrid protocol for*

the $(\mathcal{G}_1, \ldots, \mathcal{G}_i, \mathcal{F}_{\text{triv}}, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_l)$*-hybrid model by simply letting* P_i *ignore* $t_{i,r-1}^{\mathcal{F}_{\text{triv}}}$ *and always letting* $s_{i,r}^{\mathcal{F}_{\text{triv}}} = \epsilon$*. With this convention it holds that* π *realizes* $\mathcal{F}$ *in the* $(\mathcal{G}_1, \ldots, \mathcal{G}_l)$*-hybrid model iff* π *realizes* $\mathcal{F}$ *in the* $(\mathcal{G}_1, \ldots, \mathcal{G}_i, \mathcal{F}_{\text{triv}}, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_l)$*-hybrid model.*

Proof. For $l = 0$ and $l = 1$, this is Theorem 3.1 on the facing page and $l = 2$ the result follows as for said theorem. □

In particular this allows us to consider any $(\mathcal{F}_{\text{triv}}, \ldots, \mathcal{F}_{\text{triv}})$-hybrid model as the real-life model, which will come in handy in some places.

3.4 The Composition Theorem

In this section we describe how to compose synchronous protocols, and we state and prove a composition theorem saying that security is maintained under this type of composition. Let π be a $(\mathcal{G}_1, \ldots, \mathcal{G}_l)$-hybrid protocol and let γ be a $(\mathcal{H}_1, \ldots, \mathcal{H}_m)$-hybrid protocol realizing $\mathcal{G}_i$ for some $i \in \{1, \ldots, l\}$. The composition operation allows us to construct a protocol $\pi[\gamma/\mathcal{G}_i]$, which runs exactly as π, except that all calls from π to $\mathcal{G}_i$ are handled by invoking γ instead and returning the outputs from γ to π as the replies from $\mathcal{G}_i$. Since π no longer uses $\mathcal{G}_i$ we can therefore remove $\mathcal{G}_i$ from the hybrid model. Since γ uses $(\mathcal{H}_1, \ldots, \mathcal{H}_m)$ we add these instead. This results in a protocol $\pi[\gamma/\mathcal{G}_i]$ for the $(\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}, \mathcal{H}_1, \ldots, \mathcal{H}_m, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_l)$-hybrid model. To simplify the treatment we use that we have Lemma 3.1 on the preceding page. This allows us to move the ideal functionality $\mathcal{G}_i$ to be replaced to the front of the list $(\mathcal{G}_1, \ldots, \mathcal{G}_l)$ and lets us consider the remaining ideal functionalities as one functionality. I.e., we consider π as a $[\mathcal{G}_i, [\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_l]]$-hybrid protocol. Furthermore, we consider the ideal functionalities $(\mathcal{H}_1, \ldots, \mathcal{H}_l)$ as one functionality $\mathcal{H} = [\mathcal{H}_1, \ldots, \mathcal{H}_l]$. All in all this allows us to consider a $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid protocol π and a $\mathcal{H}$-hybrid protocol γ and show how to replace $\mathcal{G}_1$ by γ to obtain a $(\mathcal{H}, \mathcal{G}_2)$-hybrid protocol.

3.4.1 Composing Protocols

Assume that we are given two protocols $\gamma[\mathcal{H}] = (P_1^\gamma, \ldots, P_n^\gamma)$ for the $\mathcal{H}$-hybrid model and $\pi[\mathcal{G}_1, \mathcal{G}_2] = (P_1^\pi, \ldots, P_n^\pi)$ for the $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid model. We describe how to compose such protocols to obtain a protocol $\pi[\gamma[\mathcal{H}], \mathcal{G}_2] = (P_1^\pi[P_1^\gamma/\mathcal{G}_1], \ldots, P_n^\pi[P_n^\gamma/\mathcal{G}_1])$ for the $(\mathcal{H}, \mathcal{G}_2)$-hybrid model. This protocol is intended to be the two protocols run in lock-step while replacing the ideal functionality access of π to $\mathcal{G}_1$ by calls to γ. The protocol γ uses access to $\mathcal{H}$, so the resulting protocol is for the $(\mathcal{H}, \mathcal{G}_2)$-hybrid model. The messages sent by the parties $P_i = P_i^\pi[P_i^\gamma]$ will consist of a message from each of the two protocols. The value sent to $\mathcal{H}$ will be the value sent to $\mathcal{H}$ by P_i^γ and the output from $\mathcal{H}$ will be given to P_i^γ. The value sent to $\mathcal{G}_2$ will be the value sent to $\mathcal{G}_2$ by P_i^π and the output from $\mathcal{G}_2$ will be given to P_i^π. The value sent to $\mathcal{G}_1$ by P_i^π will be used as the input to P_i^γ and the output of P_i^γ will be given to P_i^π in the following round as the output from $\mathcal{G}_1$. For the purpose of concatenating messages, recall that $(\cdot, \cdot) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ is a bijective encoding which can be computed and inverted in PPT.

The running time of P_i is the sum of the running times of P_i^γ and P_i^π plus the running time for some applications of $(\cdot, \cdot)$ as described below. In particular the random bit string

r_i input to P_i can be split into random bit strings sufficiently long for P_i^γ and P_i^π. The activation

$$(\{m_{i,j,r}\}_{j\in[n]}, y_{i,r}, s_{i,r}) \leftarrow P_i(\{m_{j,i,r-1}\}_{j\in[n]}, x_{i,r}, t_{i,r-1})$$

is computed as follows: In round 1, take the input $m_{i,i,0} = (k, r_i)$, split r_i into r_i^π and r_i^γ and let $m_{i,i,0}^\pi = (k, r_i^\pi)$ and let $m_{i,i,0}^\gamma = (k, r_i^\gamma)$ and set all other values to ϵ. In round $r > 1$, let

$$\begin{gathered}(m_{j,i,r-1}^\pi, m_{j,i,r-1}^\gamma) \leftarrow m_{j,i,r-1} \quad \text{for } j \in [n] \setminus \{i\} \\ ((m_{i,i,r-1}^\pi, m_{i,i,r-1}^\gamma), t_{i,r-1}^{\mathcal{G}_1}) \leftarrow m_{i,i,r-1} \\ x_{i,r}^\pi \leftarrow x_{i,r} \\ (t_{i,r-1}^{\mathcal{H}}, t_{i,r-1}^{\mathcal{G}_2}) \leftarrow t_{i,r-1} \\ t_{i,r-1}^\pi \leftarrow (t_{i,r-1}^{\mathcal{G}_1}, t_{i,r-1}^{\mathcal{G}_2}) \ .\end{gathered}$$

Then compute

$$(\{m_{i,j,r}^\pi\}_{j\in[n]}, y_{i,r}^\pi, s_{i,r}^\pi) \leftarrow P_i^\pi(\{m_{j,i,r-1}^\pi\}_{j\in[n]}, x_{i,r}^\pi, t_{i,r-1}^\pi) \ .$$

Then let

$$\begin{gathered}(s_{i,r}^{\mathcal{G}_1}, s_{i,r}^{\mathcal{G}_2}) \leftarrow s_{i,r}^\pi \\ x_{i,r}^\gamma \leftarrow s_{i,r}^{\mathcal{G}_1} \ ,\end{gathered}$$

and compute

$$(\{m_{i,j,r}^\gamma\}_{j\in[n]}, y_{i,r}^\gamma, s_{i,r}^{\mathcal{H}}) \leftarrow P_i^\gamma(\{m_{j,i,r-1}^\gamma\}_{j\in[n]}, x_{i,r}^\gamma, t_{i,r-1}^{\mathcal{H}}) \ .$$

Then let

$$t_{i,r}^{\mathcal{G}_1} \leftarrow y_{i,r}^\gamma \ ,$$

and let

$$\begin{gathered}m_{i,j,r} \leftarrow (m_{i,j,r}^\pi, m_{i,j,r}^\gamma) \quad \text{for } j \in [n] \setminus \{i\} \\ m_{i,i,r} \leftarrow ((m_{i,i,r}^\pi, m_{i,i,r}^\gamma), t_{i,r}^{\mathcal{G}_1}) \\ y_{i,r} \leftarrow y_{i,r}^\pi \\ s_{i,r} \leftarrow (s_{i,r}^{\mathcal{H}}, s_{i,r}^{\mathcal{G}_2}) \ .\end{gathered}$$

Definition 3.8 *Let π be a $(\mathcal{G}_1, \ldots, \mathcal{G}_l)$-hybrid protocol and let γ be a $(\mathcal{H}_1, \ldots, \mathcal{H}_m)$-hybrid protocol. For $i \in \{1, \ldots, l\}$ we define $\pi[\gamma/\mathcal{G}_i]$ by reordering $(\mathcal{G}_1, \ldots, \mathcal{G}_l)$ to consider π a $[\mathcal{G}_i, \mathcal{G}']$-hybrid protocol for $\mathcal{G}' = [\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_l]$ and consider γ a $\mathcal{H}$-hybrid protocol for $\mathcal{H} = (\mathcal{H}_1, \ldots, \mathcal{H}_m)$. Then we let $\pi[\gamma/\mathcal{G}_i]$ be the $[\mathcal{H}, \mathcal{G}']$-hybrid protocol $\pi[\gamma[\mathcal{H}], \mathcal{G}']$ defined above and reorder the messages to and from $[[\mathcal{H}_1, \ldots, \mathcal{H}_m], [\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_l]]$ in $\pi[\gamma/\mathcal{G}_i]$ to $[\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}, \mathcal{H}_1, \ldots, \mathcal{H}_m, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_l]$.*

3.4.2 The Composition Theorem

We can prove that the composition of secure protocols yields secure protocols. I.e., if $\pi[\mathcal{G}_1, \mathcal{G}_2]$ realizes $\mathcal{F}$ and $\gamma[\mathcal{H}]$ realizes $\mathcal{G}_1$, then $\pi[\gamma[\mathcal{H}], \mathcal{G}_2]$ realizes $\mathcal{F}$. The main result needed to establish this is the following: Assume that there exists an $(\mathcal{G}_1, \mathcal{G}_2)$ interface $\mathcal{T}$ such that $\mathcal{T} : \mathcal{F} \triangleleft \pi[\mathcal{G}_1, \mathcal{G}_2]$ and that there exists a $\mathcal{H}$-hybrid interface $\mathcal{S}$ such that $\mathcal{S} : \mathcal{G}_1 \triangleleft \gamma[\mathcal{H}]$. Then we can consider the $(\mathcal{H}, \mathcal{G}_2)$-hybrid interface $\mathcal{T}[\mathcal{S}/\mathcal{G}_1]$, which given access to $\mathcal{F}$ first uses $\mathcal{T}$ to produce $\pi[\mathcal{G}_1, \mathcal{G}_2]$ from $\mathcal{F}$ and then uses $\mathcal{S}$ to produce $\gamma[\mathcal{H}]$ from the copy of $\mathcal{G}_1$ in the produced $\pi[\mathcal{G}_1, \mathcal{G}_2]$. Then it composes the view of $\pi[\mathcal{G}_1, \mathcal{G}_2]$ and $\gamma[\mathcal{H}]$ as in the composition $\pi[\gamma/\mathcal{G}_1]$ and have produced a view of $\pi[\gamma[\mathcal{H}], \mathcal{G}_2]$. This composed interface is depicted in Fig. 3.7 on the following page. We prove in Lemma 3.4 on page 89 that indeed $\mathcal{T}[\mathcal{S}/\mathcal{G}_2] : \mathcal{F} \triangleleft \pi[\gamma[\mathcal{H}], \mathcal{G}_2]$. This clearly implies the claim that the composition of secure protocols yield secure protocol.

Theorem 3.3 *Let $\mathcal{F}$ be an ideal functionality, let π be a $(\mathcal{G}_1, \ldots, \mathcal{G}_l)$-hybrid protocol and let γ by a $(\mathcal{H}_1, \ldots, \mathcal{H}_m)$-hybrid protocol. Assume that γ realizes $\mathcal{G}_i$ in the $(\mathcal{H}_1, \ldots, \mathcal{H}_m)$-hybrid model, for $i \in \{1, \ldots, l\}$, and assume that π realizes $\mathcal{F}$ in the $(\mathcal{G}_1, \ldots, \mathcal{G}_l)$-hybrid model. Then $\pi[\gamma/\mathcal{G}_i]$ realizes $\mathcal{F}$ in the $(\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}, \mathcal{H}_1, \ldots, \mathcal{H}_m, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_l)$-hybrid model. If in particular $m = 0$, then $\pi[\gamma/\mathcal{G}_i]$ realizes $\mathcal{F}$ in the $(\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_l)$-hybrid model.*

Proof. The first claim follows directly from Lemma 3.4 on page 89, using Lemma 3.1 on page 78. The second claim then follows from Theorem 3.2 on page 78. □

The proof of Lemma 3.4 on page 89 is quite technical, so a pictorial prelude is instructive. Consider again Fig. 3.7 on the following page. We have to prove that no environment can determine whether it is in Fig. 3.7 or in Fig. 3.12. We prove this by considering a number of intermediary setting. Consider first the setting in Fig. 3.8 where we run the environment $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ with $\mathcal{T}$. Here $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ just does the job that $\mathcal{T}[\mathcal{S}/\mathcal{G}_1]$ was doing in Fig. 3.7 and shows the resulting view to $\mathcal{Z}$. Then $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ takes the final output of $\mathcal{Z}$ to be its own output. All values flow in the same way in both setting, it follows that $\text{IDEAL}^{\mathcal{H},\mathcal{G}_2}_{\mathcal{F},\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} = \text{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\mathcal{H}]}$. Consider then Fig. 3.9. Under the assumption that $\mathcal{T} : \mathcal{F} \triangleleft \pi[\mathcal{G}_1, \mathcal{G}_2]$ it follows that $\text{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\mathcal{H}]} \overset{\text{c}}{\approx} \text{HYB}^{\mathcal{G}_1,\mathcal{G}_2}_{\pi,\mathcal{Z}[\mathcal{S}/\mathcal{H}]}$. Consider the setting in Fig. 3.10. Compared to Fig. 3.9 we just move π and $\mathcal{G}_2$ into $\mathcal{Z}[\pi, \mathcal{G}_2]$ and move $\mathcal{S}$ out. All values still flow in the same way. In particular, $\text{HYB}^{\mathcal{G}_1,\mathcal{G}_2}_{\pi,\mathcal{Z}[\mathcal{S}/\mathcal{H}]} = \text{IDEAL}^{\mathcal{H}}_{\mathcal{G}_1,\mathcal{S},\mathcal{Z}[\pi,\mathcal{G}_2]}$. Then compare Fig. 3.10 to Fig. 3.11. If $\mathcal{S} : \mathcal{G}_1 \triangleleft \gamma[\mathcal{H}]$, then $\text{IDEAL}^{\mathcal{H}}_{\mathcal{G}_1,\mathcal{S},\mathcal{Z}[\pi,\mathcal{G}_2]} \overset{\text{c}}{\approx} \text{HYB}^{\mathcal{H}}_{\gamma,\mathcal{Z}[\pi,\mathcal{G}_2]}$. Finally compare Fig. 3.11 to Fig. 3.12. We just moved π and $\mathcal{G}_2$ out of $\mathcal{Z}[\pi, \mathcal{G}_2]$ and into the real execution. In particular all values flow in the same way in both and $\text{HYB}^{\mathcal{H}}_{\gamma,\mathcal{Z}[\pi,\mathcal{G}_2]} = \text{HYB}^{\mathcal{H},\mathcal{G}_2}_{\pi[\gamma/\mathcal{G}_1],\mathcal{Z}}$. The claim then follows by Fact 2.1 on page 12.

In the following sections we flesh out this argument. The reader is encouraged to use the discussed depictions as a reference when reading the formal details.

3.4.3 Composing Interfaces

We describe how to compose two interfaces. Assume that we are given a $\mathcal{H}$-hybrid interface $\mathcal{S}$ and a $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid interface $\mathcal{T}$. We describe how to construct a new $(\mathcal{H}, \mathcal{G}_2)$-hybrid interface $\mathcal{T}[\mathcal{S}/\mathcal{G}_1]$. The idea is the following: Assume that $\mathcal{T}$ simulates a protocol $\pi[\mathcal{G}_1, \mathcal{G}_2]$ while

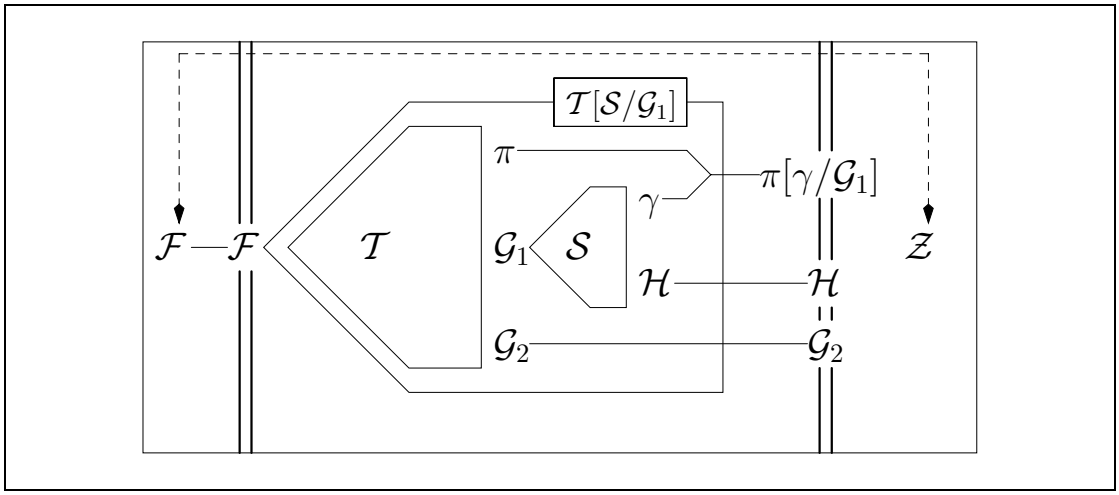

Figure 3.7: The $(\mathcal{H}, \mathcal{G}_2)$-hybrid interface obtained by composing the $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid interface $\mathcal{T}$ and the $\mathcal{H}$-hybrid interface $\mathcal{S}$, running in $\text{IDEAL}^{\mathcal{H},\mathcal{G}_2}_{\mathcal{F},\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}}$.

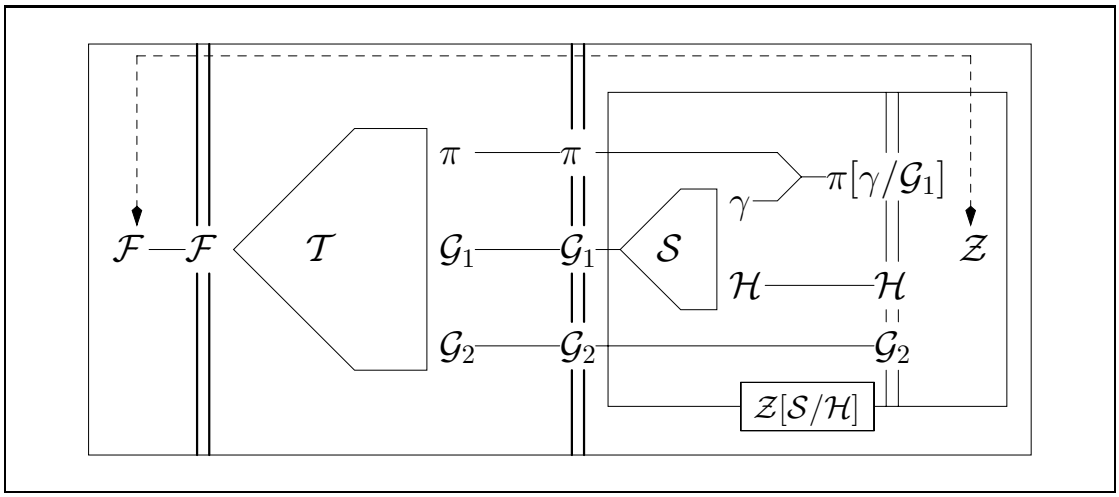

Figure 3.8: The $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid environment $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ obtained by composing the $\mathcal{H}$-hybrid interface $\mathcal{S}$ and the $(\mathcal{H}, \mathcal{G}_2)$-hybrid environment $\mathcal{Z}$, running in $\text{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\mathcal{H}]}$.

having access to the ideal functionality $\mathcal{F}$, and assume that $\mathcal{S}$ simulates a protocol $\gamma[\mathcal{H}]$ while having access to $\mathcal{G}_1$. We then want $\mathcal{U} = \mathcal{T}[\mathcal{S}/\mathcal{G}_1]$ to simulate the protocol $(\pi[\gamma/\mathcal{G}_1])[\mathcal{H}, \mathcal{G}_2]$ while having access to $\mathcal{F}$. This is done as follows: First of all $\mathcal{U}$ runs $\mathcal{T}$ using $\mathcal{U}$'s access to $\mathcal{F}$. This provides $\mathcal{U}$ with a simulated version of $\pi[\mathcal{G}_1, \mathcal{G}_2]$ consistent with $\mathcal{F}$, which in particular provides it with a simulated access to $\mathcal{G}_1$. Using the simulated access to $\mathcal{G}_1$ it then runs $\mathcal{S}$ and gets a simulated version of $\gamma[\mathcal{H}]$ consistent with the $\mathcal{G}_1$ from the simulated $\pi[\mathcal{G}_1, \mathcal{G}_2]$, which was in turn consistent with $\mathcal{F}$. It then merges the values of the simulated version of $\pi[\mathcal{G}_1, \mathcal{G}_2]$ and the simulated version of $\gamma[\mathcal{H}]$ as defined by the composition operation on protocols and obtains a simulated version of $(\pi[\gamma])[\mathcal{H}, \mathcal{G}_2]$ consistent with $\mathcal{F}$. The notation used to describe the composition operation will reflect the above idea. The composed interface works as given

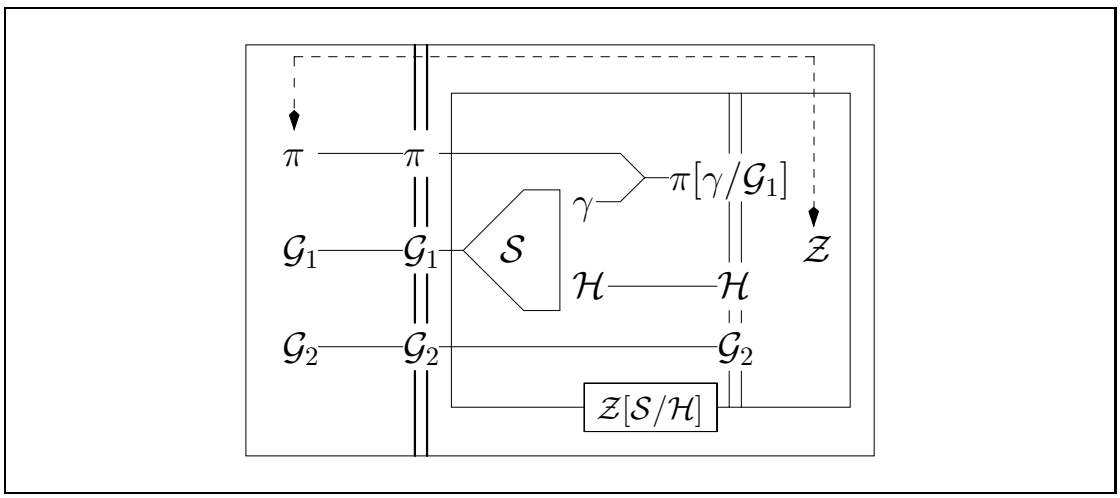

Figure 3.9: The $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid environment $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ obtained by composing the $\mathcal{H}$-hybrid interface $\mathcal{S}$ and the $(\mathcal{H}, \mathcal{G}_2)$-hybrid environment $\mathcal{Z}$, running in $\mathrm{HYB}^{\mathcal{G}_1,\mathcal{G}_2}_{\pi,\mathcal{Z}[\mathcal{S}/\mathcal{H}]}$.

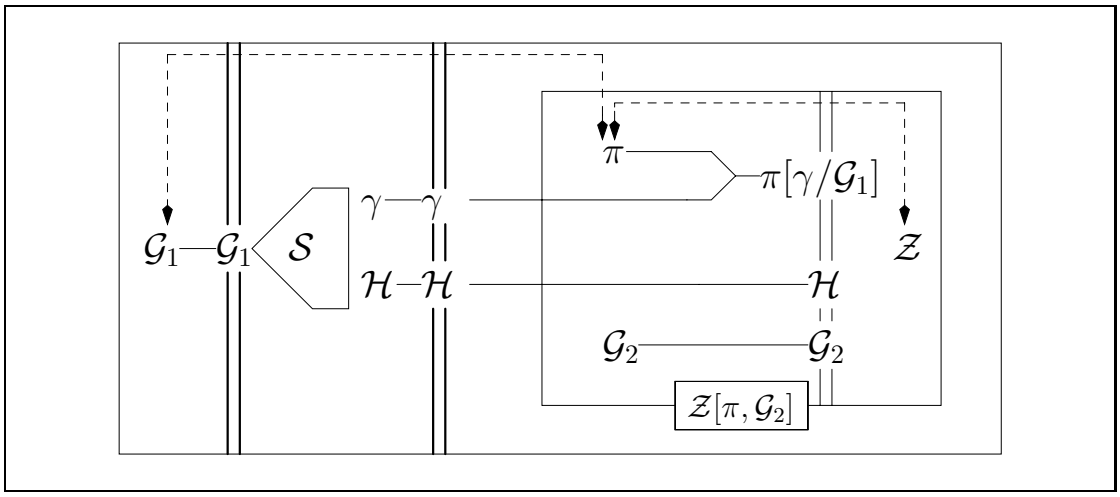

Figure 3.10: The $\mathcal{H}$-hybrid environment $\mathcal{Z}[\pi, \mathcal{G}_2]$ obtained by composing the $(\mathcal{H}, \mathcal{G}_2)$-hybrid environment $\mathcal{Z}$ with the protocol $\pi[\mathcal{G}_1, \mathcal{G}_2]$ and the ideal functionality $\mathcal{G}_2$, running in $\mathrm{IDEAL}^{\mathcal{H}}_{\mathcal{G}_1,\mathcal{S},\mathcal{Z}[\pi,\mathcal{G}_2]}$.

in Fig. 3.13 on page 85.

3.4.4 Composing an Interface with an Environment

We show how to compose a $(\mathcal{H}, \mathcal{G}_2)$-hybrid environment $\mathcal{Z}$ with a $\mathcal{H}$-hybrid interface $\mathcal{S}$ ('expecting' to run in the $\mathcal{G}_1$-hybrid model) to obtain a hybrid environment $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ for the $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid model. We construct $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ such that for all $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid interfaces $\mathcal{T}$ and all $(\mathcal{H}, \mathcal{G}_2)$-hybrid environments $\mathcal{Z}$ it holds that $\mathrm{IDEAL}^{\mathcal{H},\mathcal{G}_2}_{\mathcal{F},\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} = \mathrm{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\mathcal{H}]}$ — we so to say move $\mathcal{S}$ from $\mathcal{T}[\mathcal{S}/\mathcal{G}_1]$ into $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$. That $\mathrm{IDEAL}^{\mathcal{H},\mathcal{G}_2}_{\mathcal{F},\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} = \mathrm{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\mathcal{H}]}$

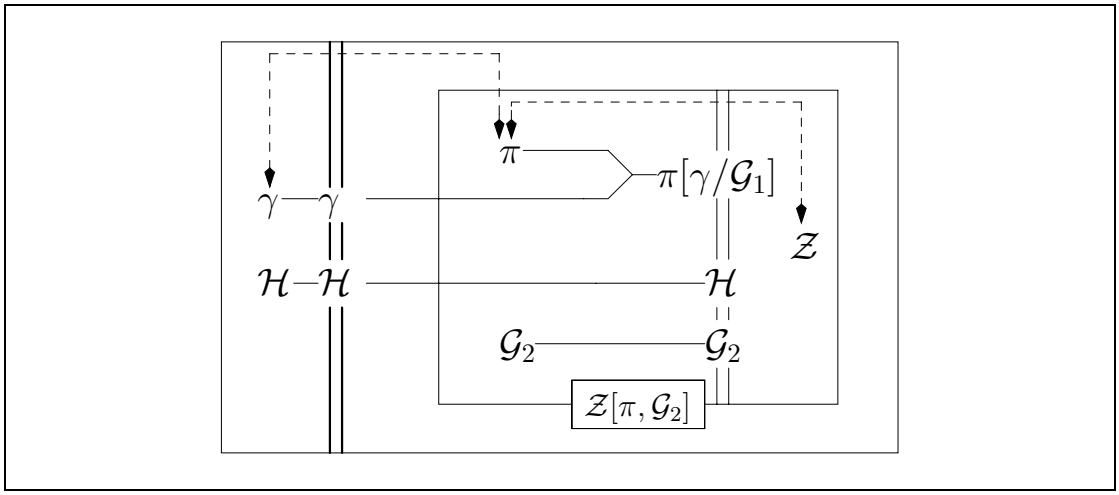

Figure 3.11: The $\mathcal{H}$-hybrid environment $\mathcal{Z}[\pi, \mathcal{G}_2]$ obtained by composing the $(\mathcal{H}, \mathcal{G}_2)$-hybrid environment $\mathcal{Z}$ with the protocol $\pi[\mathcal{G}_1, \mathcal{G}_2]$ and the ideal functionality $\mathcal{G}_2$, running in $\mathrm{HYB}^{\mathcal{H}}_{\gamma, \mathcal{Z}[\pi, \mathcal{G}_2]}$.

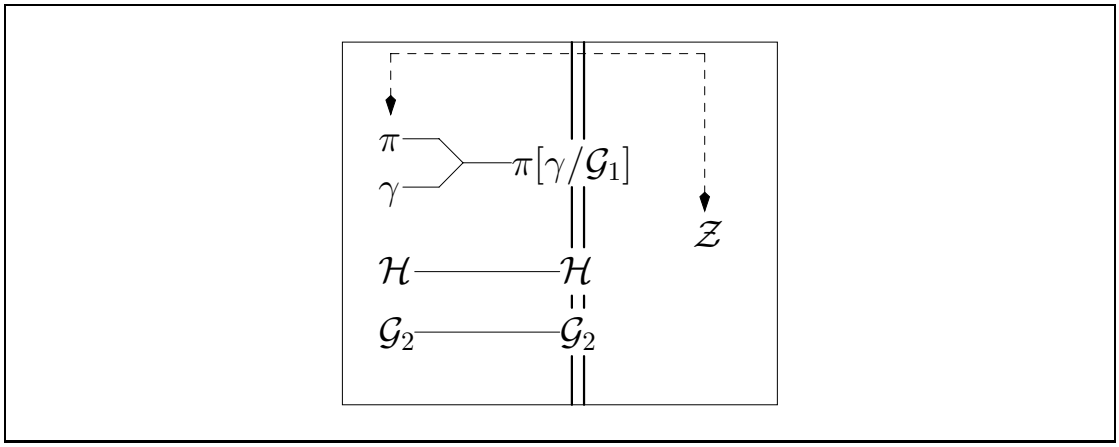

Figure 3.12: The $(\mathcal{H}, \mathcal{G}_2)$-hybrid environment $\mathcal{Z}$ running in the hybrid model with $\pi[\gamma[\mathcal{H}], \mathcal{G}_2]$, running in $\mathrm{HYB}^{\mathcal{H}, \mathcal{G}_2}_{\pi[\gamma/\mathcal{G}_1], \mathcal{Z}}$.

will be straight-forward when the construction of $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ is compared to the construction of $\mathcal{T}[\mathcal{S}/\mathcal{G}_1]$. The idea behind the construction is as follows: Assume that in $\mathrm{IDEAL}^{\mathcal{G}_1, \mathcal{G}_2}_{\mathcal{F}, \mathcal{T}, \mathcal{Z}[\mathcal{S}/\mathcal{H}]}$ the interface $\mathcal{T}$ simulates a protocol $\pi[\mathcal{G}_1, \mathcal{G}_2]$. This view is given to $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$, which shows the simulation of $\mathcal{G}_1$ to $\mathcal{S}$ to make it simulate some protocol $\gamma[\mathcal{H}]$. Then $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ composes the simulation of $\pi[\mathcal{G}_1, \mathcal{G}_2]$ and the simulation of $\gamma[\mathcal{H}]$ according to the composition theorem for protocols and shows this simulated view of $\pi[\gamma/\mathcal{G}_1, \mathcal{G}_2]$ to $\mathcal{Z}$. This is exactly the view that $\mathcal{Z}$ sees in $\mathrm{IDEAL}^{\mathcal{H}, \mathcal{G}_2}_{\mathcal{F}, \mathcal{T}[\mathcal{S}/\mathcal{G}_1], \mathcal{Z}}$; The only difference is that in $\mathrm{IDEAL}^{\mathcal{H}, \mathcal{G}_2}_{\mathcal{F}, \mathcal{T}[\mathcal{S}/\mathcal{G}_1], \mathcal{Z}}$ the simulated versions of $\pi[\mathcal{G}_1, \mathcal{G}_2]$ and $\gamma[\mathcal{H}]$ are composed by $\mathcal{T}[\mathcal{S}/\mathcal{G}_1]$ and then shown to $\mathcal{Z}$. The construction is given in Fig. 3.14 on page 87.

Interface $\mathcal{U} = \mathcal{T}[\mathcal{S}/\mathcal{G}_1]$

initialize:

$\mathcal{U}$ receives k and random bits r. When $\mathcal{S}$ or $\mathcal{T}$ reads a random bit $\mathcal{U}$ gives them the next random bit from r.

party activation:

$\mathcal{U}$ receives $\{m_{i,j,r-1}\}_{i\in C}$ from $\mathcal{Z}$ and $v_{\mathcal{F}}$ from $\mathcal{F}$ and must provide outputs $\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}$ and $(v_{\mathcal{H}}, v_{\mathcal{G}_2})$. This is done as follows:

1. For $i \in C$ compute $(m^{\pi}_{i,j,r-1}, m^{\gamma}_{i,j,r-1}) \leftarrow m_{i,j,r-1}$.
2. Input $\{m^{\pi}_{i,j,r-1}\}_{i\in C}$ and $v_{\mathcal{F}}$ to $\mathcal{T}$ which generates values $\{m^{\gamma}_{i,j,r}\}_{j\in[n]\setminus\{i\}}$ and $(v_{\mathcal{G}_1}, v_{\mathcal{G}_2})$.
3. Input $\{m^{\gamma}_{i,j,r-1}\}_{i\in C}$ and $v_{\mathcal{G}_1}$ to $\mathcal{S}$ which generates values $\{m^{\gamma}_{i,j,r}\}_{j\in[n]\setminus\{i\}}$ and $v_{\mathcal{H}}$.
4. Output $\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}$ and $(v_{\mathcal{H}}, v_{\mathcal{G}_2})$, where $m_{i,j,r} = (m^{\pi}_{i,j,r}, m^{\gamma}_{i,j,r})$.

corrupt:

$\mathcal{U}$ receives $(\texttt{corrupt}, i, v_{\mathcal{F}})$ and must provide an output $(r_i, (v_{\mathcal{H}}, v_{\mathcal{G}_2}))$. This is done as follows:

1. Input $(\texttt{corrupt}, i, v_{\mathcal{F}})$ to $\mathcal{T}$ to generate $(r^{\pi}_i, (v_{\mathcal{G}_1}, v_{\mathcal{G}_2}))$.
2. Input $(\texttt{corrupt}, i, v_{\mathcal{G}_1})$ to $\mathcal{S}$ to generate $(r^{\gamma}_i, v_{\mathcal{H}})$.
3. Output $(r_i = (r^{\pi}_i, r^{\gamma}_i), (v_{\mathcal{H}}, v_{\mathcal{G}_2}))$.

end round:

$\mathcal{U}$ receives $(\texttt{end round}, (v_{\mathcal{H}}, v_{\mathcal{G}_2}))$ and must produce an output $v_{\mathcal{F}}$ for $\mathcal{F}$ in response to which it receives $\{y_{i,r}\}_{i\in C}$. Then it must produce outputs $\{(t^{\mathcal{H}}_{i,r}, t^{\mathcal{G}_2}_{i,r})\}_{i\in C}$. This is done as follows:

1. Activate $\mathcal{S}$ on input $(\texttt{end round}, v_{\mathcal{H}})$ and receive as output a value $v_{\mathcal{G}_1}$.
2. Activate $\mathcal{T}$ on input $(\texttt{end round}, (v_{\mathcal{G}_1}, v_{\mathcal{G}_2}))$ and receive as output a value $v_{\mathcal{F}}$.
3. Output $v_{\mathcal{F}}$ and receive $\{y^{\mathcal{F}}_{i,r}\}_{i\in C}$.
4. Hand $\{y^{\mathcal{F}}_{i,r}\}_{i\in C}$ to $\mathcal{T}$ and get the output $\{(t^{\mathcal{G}_1}_{i,r}, t^{\mathcal{G}_2}_{i,r})\}_{i\in C}$.
5. Hand $\{t^{\mathcal{G}_1}_{i,r}\}_{i\in C}$ to $\mathcal{S}$ and get the output $\{t^{\mathcal{H}}_{i,r}\}_{i\in C}$.
6. Output $\{(t^{\mathcal{H}}_{i,r}, t^{\mathcal{G}_2}_{i,r})\}_{i\in C}$.

Figure 3.13: The $(\mathcal{H}, \mathcal{G}_2)$-hybrid interface $\mathcal{U}$ obtained by composing the $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid interface $\mathcal{T}$ and the $\mathcal{H}$-hybrid interface $\mathcal{S}$.

Lemma 3.2 *For all $(\mathcal{H}, \mathcal{G}_2)$-hybrid environments $\mathcal{Z}$, all $\mathcal{H}$-hybrid interfaces $\mathcal{S}$, all ideal functionalities $\mathcal{F}$ and all $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid interfaces $\mathcal{T}$ it holds that*

$$\mathrm{IDEAL}^{\mathcal{H},\mathcal{G}_1}_{\mathcal{F},\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} = \mathrm{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\mathcal{H}]} \ .$$

Proof. This follows by construction, as we just moved the execution of $\mathcal{S}$ from $\mathcal{T}[\mathcal{S}/\mathcal{G}_1]$ into $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$. The data-flow is the same in both executions, the only difference being that the

entities are placed at different locations in the two. This is straight-forward to verify by following the flow of values in Fig. 3.13 on the preceding page and Fig. 3.14 on the next page. To make this easier Fig. 3.14 has been augmented with comments describing the data-flow outside $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$. □

3.4.5 Composing an Environment with a Protocol

Finally we define a composition of a $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid protocol π with a $(\mathcal{H}, \mathcal{G}_2)$-hybrid environment, resulting in a $\mathcal{H}$-hybrid environment $\mathcal{Z}[\pi, \mathcal{G}_2]$. We think of $\mathcal{Z}$ as an environment which thinks that it runs $\pi[\gamma/\mathcal{G}_1]$ for a $\mathcal{H}$-hybrid protocol γ. We then construct $\mathcal{Z}[\pi, \mathcal{G}_2]$ such that $\text{HYB}^{\mathcal{H}}_{\gamma,\mathcal{Z}[\pi,\mathcal{G}_2]} = \text{HYB}^{\mathcal{H},\mathcal{G}_2}_{\pi[\gamma/\mathcal{G}_1],\mathcal{Z}}$ — we so to say move the execution of π from $\text{HYB}^{\mathcal{H},\mathcal{G}_2}_{\pi[\gamma/\mathcal{G}_1],\mathcal{Z}}$ into $\mathcal{Z}[\pi, \mathcal{G}_2]$. To assure that $\text{HYB}^{\mathcal{H}}_{\gamma,\mathcal{Z}[\pi,\mathcal{G}_2]} = \text{HYB}^{\mathcal{H},\mathcal{G}_2}_{\pi[\gamma/\mathcal{G}_1],\mathcal{Z}}$, in the run $\text{HYB}^{\mathcal{H}}_{\gamma,\mathcal{Z}[\pi,\mathcal{G}_2]}$ the environment $\mathcal{Z}[\pi, \mathcal{G}_2]$ uses the messages from π to $\mathcal{G}_1$ (the $s^{\mathcal{G}_1}_{i,r}$ value in $(\{m^{\pi}_{i,j,r}\}_{j\in[n]}, y^{\pi}_{i,r}, (s^{\mathcal{G}_1}_{i,r}, s^{\mathcal{G}_2}_{i,r})) = P^{\pi}_i(\{m^{\pi}_{j,i,r-1}\}_{j\in[n]}, x^{\pi}_{i,r}, (t^{\mathcal{G}_1}_{i,r-1}, t^{\mathcal{G}_2}_{i,r-1}))$) to activate γ, and it returns the output from γ to π as the $t^{\mathcal{G}_1}_{i,r}$ value in the next round — exactly as in the composition of protocols. Furthermore it runs an internal copy of $\mathcal{G}_2$ on the $s^{\mathcal{G}_2}_{i,r}$ values and returns the outputs from $\mathcal{G}_2$ to P^{π}_i as the $t^{\mathcal{G}_2}_{i,r}$ value. It then composes its internal view of π with the view of γ returned to $\mathcal{Z}[\pi, \mathcal{G}_2]$ in $\text{HYB}^{\mathcal{H}}_{\gamma,\mathcal{Z}[\pi,\mathcal{G}_2]}$, according to the composition operation on protocols. Then $\mathcal{Z}[\pi, \mathcal{G}_2]$ shows $\mathcal{Z}$ this view of $\pi[\gamma/\mathcal{G}_1]$. The details are given in Fig. 3.15 on page 88.

Lemma 3.3 *For all $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid protocols π, all $\mathcal{H}$-hybrid protocols γ and all $(\mathcal{H}, \mathcal{G}_2)$-hybrid environments $\mathcal{Z}$, it holds that*

$$\text{HYB}^{\mathcal{H},\mathcal{G}_2}_{\pi[\gamma/\mathcal{G}_1],\mathcal{Z}} = \text{HYB}^{\mathcal{H}}_{\gamma,\mathcal{Z}[\pi,\mathcal{G}_2]} \ .$$

For all $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid protocols π, all ideal functionalities $(\mathcal{G}_1, \mathcal{G}_2)$, all $(\mathcal{H}, \mathcal{G}_2)$-hybrid environments $\mathcal{Z}$ and all $\mathcal{H}$-hybrid interfaces $\mathcal{S}$ it holds that

$$\text{HYB}^{\mathcal{G}_1,\mathcal{G}_2}_{\pi,\mathcal{Z}[\mathcal{S}/\mathcal{H}]} = \text{IDEAL}^{\mathcal{H}}_{\mathcal{G}_1,\mathcal{S},\mathcal{Z}[\pi,\mathcal{G}_2]} \ .$$

Proof. The first claim is obtained by construction, as we moved the execution of π into $\mathcal{Z}$. To verify the second claim, consider $\text{IDEAL}^{\mathcal{H}}_{\mathcal{G},\mathcal{S},\mathcal{Z}[\pi,\mathcal{G}_2]}$. Here π is run in the environment $\mathcal{Z}$, and the sub-protocol calls of π to $\mathcal{G}_2$ (the $s^{\mathcal{G}_2}_{i,r}$ and $t^{\mathcal{G}_2}_{i,r}$ values) are the inputs and outputs of $\mathcal{G}_2$ (run inside $\mathcal{Z}[\pi, \mathcal{G}_2]$). Furthermore, the sub-protocol calls of π to $\mathcal{G}_1$ (the $s^{\mathcal{G}_1}_{i,r}$ and $t^{\mathcal{G}_1}_{i,r}$ values) are the inputs and outputs of $\mathcal{G}_1$, and $\mathcal{S}$ then simulates a view of γ and $\mathcal{H}$ given access to $\mathcal{G}_1$. This view is then composed with the view of π (by $\mathcal{Z}[\pi, \mathcal{G}_2]$) and shown to $\mathcal{Z}$. In $\text{HYB}^{\mathcal{G}_1,\mathcal{G}_2}_{\pi,\mathcal{Z}[\mathcal{S}]}$ the protocol π is run in the environment $\mathcal{Z}[\mathcal{S}]$, which means that it communicates with $\mathcal{Z}$. Furthermore, the sub-protocol calls of π are the inputs and outputs of $\mathcal{G}_1$ and $\mathcal{G}_2$ as above, and in $\mathcal{Z}[\mathcal{S}]$ the simulator $\mathcal{S}$ simulates a view of γ and $\mathcal{H}$ using the access to $\mathcal{G}_1$. This view is then composed with the view of π (by $\mathcal{Z}[\mathcal{S}]$) and shown to $\mathcal{Z}$. This means that the data-flow in both executions is exactly the same, the only difference being that the entities are placed at different locations in the two. □

Environment $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$

initialize:

$\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ receives $(k, z, r_{\mathcal{Z}})$ and picks a section of $r_{\mathcal{Z}}$ as random bits r for $\mathcal{S}$. Let $r'_{\mathcal{Z}}$ be the remaining random bits and input $(k, z, r'_{\mathcal{Z}})$ to $\mathcal{Z}$.

environment:

When asked to produce a command, run $\mathcal{Z}$ to get a command c. If $c = (\texttt{guess}, b)$, then output c. Otherwise proceed as described below.

party activation:

If $c = (\texttt{activate}, i, x_{i,r}, \{m_{i,j,r-1}\}_{i \in C})$ then:

1. For $i \in C$ compute $(m^{\pi}_{i,j,r-1}, m^{\gamma}_{i,j,r-1}) \leftarrow m_{i,j,r-1}$ and output $(\texttt{activate}, i, x_{i,r}, \{m^{\pi}_{i,j,r-1}\}_{i \in C})$.
2. *Then in* $\text{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\mathcal{S}]}$ *the party* P_i *is activated and the values* $\{m^{\pi}_{i,j,r-1}\}_{i \in C}$ *and* $v_{\mathcal{F}}$ *are input to* $\mathcal{T}$ *which generates values* $\{m^{\gamma}_{i,j,r}\}_{j \in [n] \setminus \{i\}}$ *and* $(v_{\mathcal{G}_1}, v_{\mathcal{G}_2})$*, which are given to* $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$.
3. Input $\{m^{\gamma}_{i,j,r-1}\}_{i \in C}$ and $v_{\mathcal{G}_1}$ to $\mathcal{S}$ which generates values $\{m^{\gamma}_{i,j,r}\}_{j \in [n] \setminus \{i\}}$ and $v_{\mathcal{H}}$.
4. Input $\{m_{i,j,r}\}_{j \in [n] \setminus \{i\}}$ and $(v_{\mathcal{H}}, v_{\mathcal{G}_2})$ to $\mathcal{Z}$, where $m_{i,j,r} = (m^{\pi}_{i,j,r}, m^{\gamma}_{i,j,r})$.

corrupt:

If $c = (\texttt{corrupt}, i)$, then output $(\texttt{corrupt}, i)$.

1. *In* $\text{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\mathcal{H}]}$ *the value* $(\texttt{corrupt}, i, v_{\mathcal{F}})$ *is input to to* $\mathcal{T}$ *to generate* $(r^{\pi}_i, (v_{\mathcal{G}_1}, v_{\mathcal{G}_2}))$*, which is given to* $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$.
2. Input $(\texttt{corrupt}, i, v_{\mathcal{G}_1})$ to $\mathcal{S}$ to generate $(r^{\gamma}_i, v_{\mathcal{H}})$.
3. Input $(r_i = (r^{\pi}_i, r^{\gamma}_i), (v_{\mathcal{H}}, v_{\mathcal{G}_2}))$ to $\mathcal{Z}$.

end round:

If $c = (\texttt{end round}, (v_{\mathcal{H}}, v_{\mathcal{G}_2}))$, then

1. Activate $\mathcal{S}$ on input $(\texttt{end round}, v_{\mathcal{H}})$ and receive as output a value $v_{\mathcal{G}_1}$.
2. Output $(\texttt{end round}, (v_{\mathcal{G}_1}, v_{\mathcal{G}_2}))$.
3. *In* $\text{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\mathcal{H}]}$ *the interface* $\mathcal{T}$ *is activated on input* $(\texttt{end round}, (v_{\mathcal{G}_1}, v_{\mathcal{G}_2}))$ *to generate* $v_{\mathcal{F}}$*. Then* $(\texttt{activate}, v_{\mathcal{F}})$ *is input to* $\mathcal{F}$ *to generate* $\{y^{\mathcal{F}}_{i,r}\}_{i \in [n]}$*. Then* $\{y^{\mathcal{F}}_{i,r}\}_{i \in C}$ *is input to* $\mathcal{T}$ *to generate* $\{(t^{\mathcal{G}_1}_{i,r}, t^{\mathcal{G}_2}_{i,r})\}_{i \in C}$*, and* $(\{y^{\mathcal{F}}_{i,r}\}_{i \in H}, \{(t^{\mathcal{G}_1}_{i,r}, t^{\mathcal{G}_2}_{i,r})\}_{i \in C})$ *is given to* $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$.
4. Input $\{t^{\mathcal{G}_1}_{i,r}\}_{i \in C}$ to $\mathcal{S}$ to get $\{t^{\mathcal{H}}_{i,r}\}_{i \in C}$ and input $(\{y^{\mathcal{F}}_{i,r}\}_{i \in H}, \{(t^{\mathcal{H}}_{i,r}, t^{\mathcal{G}_2}_{i,r})\}_{i \in C})$ to $\mathcal{Z}$.

Figure 3.14: The $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid interface $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ obtained by composing a $(\mathcal{H}, \mathcal{G}_2)$-hybrid environment $\mathcal{Z}$ with a $\mathcal{H}$-hybrid interface $\mathcal{S}$.

Environment $\mathcal{Z}[\pi, \mathcal{G}_2]$

initialize:

$\mathcal{Z}[\pi, \mathcal{G}_2]$ receives $(k, z, r_{\mathcal{Z}})$ and picks a section of $r_{\mathcal{Z}}$ as random bits r_i^π for P_i^π and a section for $\mathcal{G}_2$. Let $r'_{\mathcal{Z}}$ be the remaining random bits and input $(k, z, r'_{\mathcal{Z}})$ to $\mathcal{Z}$. For $i, j \in [n]$, let $m_{i,j,0}^\pi = \epsilon$ and let $m_{i,i,0}^\pi = (k, r_i^\pi)$. For $i = 1, 2$, let $t_{i,-1}^{\mathcal{G}_i} = \epsilon$.

environment:

When asked to produce a command, run $\mathcal{Z}$ to get a command c. If $c = (\texttt{guess}, b)$, then output c. Otherwise proceed as described below.

party activation:

If $c = (\texttt{activate}, i, x_{i,r}, \{m_{i,j,r-1}\}_{i \in C})$ then:

1. For $i \in C$, let $(m_{i,j,r-1}^\pi, m_{i,j,r-1}^\gamma) \leftarrow m_{i,j,r-1}$. This defines sets $\{m_{i,j,r-1}^\pi\}_{i \in C}$ and $\{m_{i,j,r-1}^\gamma\}_{i \in C}$. Values $\{m_{j,i,r-1}^\pi\}_{j \in H}$ and $(t_{i,r-1}^{\mathcal{G}_1}, t_{i,r-1}^{\mathcal{G}_2})$ were defined in the previous round.[a] Add these to $\{m_{j,i,r-1}^\pi\}_{j \in C}$. Then let $x_{i,r}^\pi = x_{i,r}$ and compute

$$(\{m_{i,j,r}^\pi\}_{j \in [n]}, y_{i,r}^\pi, (s_{i,r}^{\mathcal{G}_1}, s_{i,r}^{\mathcal{G}_2})) = P_i^\pi(\{m_{j,i,r-1}^\pi\}_{j \in [n]}, x_{i,r}^\pi, (t_{i,r-1}^{\mathcal{G}_1}, t_{i,r-1}^{\mathcal{G}_2})) \ .$$

   Then let $x_{i,r}^\gamma \leftarrow s_{i,r}^{\mathcal{G}_1}$ and output $(\texttt{activate}, i, x_{i,r}^\gamma, \{m_{i,j,r-1}^\gamma\}_{i \in C})$.
2. Receive $(\{m_{i,j,r}^\gamma\}_{j \in [n] \setminus \{i\}}, v_{\mathcal{H}})$ (*from* $\mathrm{HYB}_{\gamma, \mathcal{Z}[\pi, \mathcal{G}_2]}^{\mathcal{H}}$).
3. Input $s_{i,r}^{\mathcal{G}_2}$ to $\mathcal{G}_2$ and receive a value $v_{\mathcal{G}_2}$ on its SOT.
4. Then for $j \in [n] \setminus \{i\}$, let $m_{i,j,r} \leftarrow (m_{i,j,r}^\pi, m_{i,j,r}^\gamma)$ and input $(\{m_{i,j,r}\}_{j \in [n] \setminus \{i\}}, (v_{\mathcal{H}}, v_{\mathcal{G}_2}))$ to $\mathcal{Z}$.

corrupt:

If $c = (\texttt{corrupt}, i)$ then:

1. Output $(\texttt{corrupt}, i)$.
2. Receive $(r_i^\gamma, v_{\mathcal{H}})$ (*from* $\mathrm{HYB}_{\gamma, \mathcal{Z}[\pi, \mathcal{G}_2]}^{\mathcal{H}}$).
3. Input $(\texttt{corrupt}, i)$ to $\mathcal{G}_2$ and receive $v_{\mathcal{G}_2}$.
4. Input $(r_i = (r_i^\pi, r_i^\gamma), (v_{\mathcal{H}}, v_{\mathcal{G}_2}))$ to $\mathcal{Z}$.

end round:

If $c = (\texttt{end round}, (v_{\mathcal{H}}, v_{\mathcal{G}_2}))$, then

1. Output $(\texttt{end round}, v_{\mathcal{H}})$.
2. Receive $(\{y_{i,r}^\gamma\}_{i \in H}, \{t_{i,r}^{\mathcal{H}}\}_{i \in H})$ (*from* $\mathrm{HYB}_{\gamma, \mathcal{Z}[\pi, \mathcal{G}_2]}^{\mathcal{H}}$).
3. Input $(\texttt{activate}, v_{\mathcal{G}_2})$ to $\mathcal{G}_2$ and receive the output $\{t_{i,r}^{\mathcal{G}_2}\}_{i \in [n]}$.
4. For $i \in H$, let $y_{i,r} = y_{i,r}^\pi$, where $y_{i,r}^\pi$ is the value defined in party activation Step 1.
5. Input $(\{y_{i,r}\}_{i \in H}, \{(t_{i,r}^{\mathcal{H}}, t_{i,r}^{\mathcal{G}_2})\}_{i \in H})$ to $\mathcal{Z}$.
6. For $i \in H$, let $t_{i,r}^{\mathcal{G}_1} \leftarrow y_{i,r}^\gamma$.

[a] The values $t_{i,r-1}^{\mathcal{G}_1}$ and $t_{i,r-1}^{\mathcal{G}_2}$ were defined in initialize or in end round in Step 6 resp. Step 3.

Figure 3.15: The $\mathcal{H}$-hybrid interface $\mathcal{Z}[\pi, \mathcal{G}_2]$ obtained by composing a $(\mathcal{H}, \mathcal{G}_2)$-hybrid environment $\mathcal{Z}$ with a $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid protocol π.

Lemma 3.4 *Let π be a $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid protocol and let γ by a $\mathcal{H}$-hybrid protocol. Let $\mathcal{S}$ and $\mathcal{T}$ be hybrid interfaces and assume that 1) $\mathcal{S} : \mathcal{G}_1 \lhd \gamma[\mathcal{H}]$ and 2) $\mathcal{T} : \mathcal{F} \lhd \pi[\mathcal{G}_1, \mathcal{G}_2]$. Then $\mathcal{T}[\mathcal{S}/\mathcal{G}_1] : \mathcal{F} \lhd \pi[\gamma[\mathcal{H}], \mathcal{G}_2]$.*

Proof. The lemma follows by

$$\begin{aligned}
\mathrm{IDEAL}^{\mathcal{H},\mathcal{G}_2}_{\mathcal{F},\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} &= \mathrm{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\mathcal{H}]} \\
&\overset{\mathrm{c}}{\approx} \mathrm{HYB}^{\mathcal{G}_1,\mathcal{G}_2}_{\pi,\mathcal{Z}[\mathcal{S}/\mathcal{H}]} \\
&= \mathrm{IDEAL}^{\mathcal{H}}_{\mathcal{G}_1,\mathcal{S},\mathcal{Z}[\pi,\mathcal{G}_2]} \\
&\overset{\mathrm{c}}{\approx} \mathrm{HYB}^{\mathcal{H}}_{\gamma,\mathcal{Z}[\pi,\mathcal{G}_2]} \\
&= \mathrm{HYB}^{\mathcal{H},\mathcal{G}_2}_{\pi[\gamma/\mathcal{G}_1],\mathcal{Z}} \ ,
\end{aligned}$$

using transitivity of $\overset{\mathrm{c}}{\approx}$ and, in that order, Lemma 3.2 on page 84, the second premise, Lemma 3.3 on page 86, the first premise, and once again Lemma 3.3 on page 86. □

3.5 Specifying Restricted Input-Output Behavior

In this section we address the question of how to specify that a protocol is only required to implement a functionality under a given use. One prominent example is that a protocol only implements a given functionality if all (honest) parties are activated in the same round with corresponding inputs. This is also an example of a restricted use of the protocol which cannot be checked among the parties, and which can be essential for the security of the protocol. It is therefore important that we have a way to formalize a statement like "π realizes $\mathcal{F}$ when all (honest) parties are activated in the same round". Surprisingly enough two approaches to formalizing this, which would seem equivalent at first, turn out to be essentially different. In the following sections, we demonstrate this, we point to the formalization that we think is the right one, and argue why. Then we proceed to formalize the chosen notion.

We define an input-output behavior for a protocol to be an ITM $\mathcal{IO}$. We will simply show the input-output sequence to $\mathcal{IO}$, and if $\mathcal{IO}$ ever outputs `fail`, then by definition the observed sequence was not allowed, we say that $\mathcal{IO}$ was violated.

3.5.1 Changing the Ideal Functionality

The first way to formalize restricted input-output behavior for a protocol will be by using the functionality to check the input-output sequence. We then let the functionality break down if the input-output sequence is not allowed. This approach has the advantage that the notion of restricted input-output behavior for a protocol is captured within the current framework. Given an ideal functionality $\mathcal{F}$ and an IO behavior $\mathcal{IO}$, the functionality $\mathcal{F}$ restricted to $\mathcal{IO}$ is given in Fig. 3.16 on the following page. The intuition behind $\mathcal{F}_{|\mathcal{IO}}$ is that if the IO restriction $\mathcal{IO}$ is violated, then $\mathcal{F}_{|\mathcal{IO}}$ is completely corrupted. Therefore simulating a protocol π having access to $\mathcal{F}_{|\mathcal{IO}}$ is trivial after the violation. One is given all the inputs to $\mathcal{F}_{|\mathcal{IO}}$ on the SOT of $\mathcal{F}_{|\mathcal{IO}}$ and can therefore run π on these inputs, and can make $\mathcal{F}_{|\mathcal{IO}}$

Functionality $\mathcal{F}_{|\mathcal{IO}}$

correct inputs:

Initially $\mathcal{F}_{|\mathcal{IO}}$ proceeds as follows:

initialization:

On input (k, r), input (k, r) to $\mathcal{F}$.

party activation:

On input (i, x_i), input (i, x_i) to $\mathcal{IO}$. If $\mathcal{IO}$ outputs `fail`, then go to incorrect inputs. Otherwise, input (i, x_i) to $\mathcal{F}$, receive a value v from $\mathcal{F}$ and output v. Also, input v to $\mathcal{IO}$.

corrupt:

On input $(\texttt{corrupt}, i)$, input $(\texttt{corrupt}, i)$ to $\mathcal{IO}$.[a] If $\mathcal{IO}$ outputs `fail`, then go to incorrect inputs. Otherwise, input $(\texttt{corrupt}, i)$ to $\mathcal{F}$, receive a value r from $\mathcal{F}$ and output r.

end round:

On input $(\texttt{activate}, v)$, input $(\texttt{activate}, v)$ to $\mathcal{F}$ and receive a value $\{t_i\}_{i\in[n]}$ from $\mathcal{F}$. Input $\{y_i\}_{i\in H}$ be $\mathcal{IO}$ and output $\{t_i\}_{i\in[n]}$.

incorrect inputs:

If $\mathcal{IO}$ ever outputs `fail`, then break down as follows: First output `fail` and then output the entire internal state of $\mathcal{F}$ on the SOT, including randomness and all previous inputs and outputs, and from now on handle commands as follows:

party activation:

On input (i, x_i), output (i, x_i) on the SOT.

corrupt:

On input $(\texttt{corrupt}, i)$ output $\perp$.

end round:

On input $(\texttt{activate}, v)$ interpret v as a set $\{y_i\}_{i\in[n]}$ and output $\{y_i\}_{i\in[n]}$.

[a] Recall that when party P_i is corrupted, the functionality is given input $(\texttt{corrupt}, i)$.

Figure 3.16: The functionality $\mathcal{F}$ restricted to the input-output behavior for a protocol $\mathcal{IO}$.

output the outputs of π to $\mathcal{Z}$ by putting the desired outputs on the SIT of $\mathcal{F}_{|\mathcal{IO}}$. Notice that we input the corruption pattern of $\mathcal{Z}$ to $\mathcal{IO}$. This will allow to define restrictions on the corruption patterns of the environment. We return to this issue in Section 3.5.11.

Given a protocol π, an ideal functionality $\mathcal{F}$ and an IO behavior $\mathcal{IO}$, we say that π realizes $\mathcal{F}$ under the input-output behavior for a protocol $\mathcal{IO}$, if π realizes $\mathcal{F}_{|\mathcal{IO}}$. An example of a Byzantine agreement functionality which breaks down on invalid input-output behavior is given in Fig. 3.17 on the next page to exemplify the approach. A Byzantine agreement basically allows n parties, each starting with an arbitrary input value, to decide on one common value, with the guarantee that if they started out with the same value, then that value will be the result. The functionality in Fig. 3.17 basically says that the protocol is only required to work if the parties start executing each given Byzantine agreement in the same round.

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

The functionality runs with parties $P_1, \ldots, P_n$. All inputs to the functionality are output on the SOT. The functionality proceeds as follows:

activation:

Let H denote the current set of indices of honest parties. If in some round each P_i for $i \in H$ inputs $(baid, m_i)$, where $baid \in \{0,1\}^*$ is a BA id and $m_i \in \{0,1\}^*$ is the message, then output $(\texttt{decide}, baid, m)$ to all parties, where m is defined as follows: If there exists $m \in \{0,1\}^*$ such that $m = m_i$ for $i \in H$, then use that value. Otherwise, let the adversary decide. I.e., in the command $(\texttt{activate}, v)$ at the end of the round, interpret part of v as a value $(baid, m)$, and use that m value.

incorrect inputs:

If in some round an honest party inputs a value which is not of the form $(baid, m_i)$ for $m_i \in \{0,1\}^*$ or some honest party inputs $(baid, m_i)$ for $m_i \in \{0,1\}^*$ and some honest party does not input a value of the form $(baid, m_j)$ for $m_j \in \{0,1\}^*$, then we say that $\mathcal{F}_{\text{BA}}$ was activated incorrectly. This defines an IO restriction $\mathcal{IO}_{\text{BA}}$. If $\mathcal{IO}_{\text{BA}}$ is ever violated, then break down as in incorrect inputs in Fig. 3.16 on the preceding page.

Figure 3.17: The BA functionality.

Definition 3.9 *When we instruct an ideal functionality to break down we mean that it should start behaving as in incorrect inputs in Fig. 3.16 on the facing page.*

3.5.2 Changing the Model

We can also capture the notion of restricted IO behavior for a protocol by extending the model; By saying that π realizes $\mathcal{F}$ under the input-output behavior for a protocol $\mathcal{IO}$ if there exists an interface which works for all environments which do not violate the IO behavior. We can formalize this by making the change to the ideal process that as soon as $\mathcal{IO}$ outputs `fail`, the ideal process halts with output `fail`. We then require that the interface works for all environments $\mathcal{Z}$, where $\Pr[\text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}} = \texttt{fail}] = 0$. In the simple case of threshold adversaries, which is only allowed to corrupt a certain number of parties, the definition in [Can01b] already takes this approach (since it quantifies only over environments $\mathcal{Z}$ which corrupt up to t parties).

The problem with the above definition is that if we adopt it, then we have to also extend the composition theorem to account for this new definition. We would have to somehow formalize a claim like: If π realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model and uses $\mathcal{G}$ according to the IO restriction $\mathcal{IO}$ and if γ realizes $\mathcal{G}$ under the IO behavior $\mathcal{IO}$, then $\pi[\gamma/\mathcal{G}]$ realizes $\mathcal{F}$ in the real-life model. This is a good reason for being biased towards the definition within the model. And, as we demonstrate now, we actually do have to pick a choice, as the two notions are not equivalent.

3.5.3 The Difference

It is fairly straight-forward to give an intuitive argument why the two notions are equivalent:

Environment $\mathcal{Z}_{|\mathcal{IO}}$

initialization:
On input $(k, z, r_{\mathcal{Z}})$, input $(k, z, r_{\mathcal{Z}})$ to $\mathcal{Z}$.

environment activation:
When run to produce a command, run $\mathcal{Z}$ to produce a command c. If $c = (\texttt{guess}, b)$, then output $(\texttt{guess}, b)$. Otherwise proceed as below.

party activation:
If $c = (\texttt{activate}, i, x_i, M)$, then input (i, x_i) to $\mathcal{IO}$. If $\mathcal{IO}$ outputs `fail`, then output $(\texttt{guess}, 1)$ and terminate. Otherwise, output $(\texttt{activate}, i, x_i, M)$, receive a value and hand it to $\mathcal{Z}$.

corrupt:
If $c = (\texttt{corrupt}, i)$, then input $(\texttt{corrupt}, i)$ to $\mathcal{IO}$. If $\mathcal{IO}$ outputs `fail`, then output $(\texttt{guess}, 1)$ and terminate. Otherwise, output c, receive a value and hand it to $\mathcal{Z}$.

end round:
If $c = (\texttt{end round})$ (or $c = (\texttt{end round}, v)$, in the hybrid models), then output c, receive a value and hand it to $\mathcal{Z}$.

Furthermore, extract from the value handed to $\mathcal{Z}$ the values $\{y_i\}_{i \in H}$ and input $(\texttt{end round}, \{y_i\}_{i \in H})$ to $\mathcal{IO}$.

Figure 3.18: The environment $\mathcal{Z}$ restricted to the input-output behavior for a protocol $\mathcal{IO}$.

Assume first that π realizes $\mathcal{F}$ under the IO restriction $\mathcal{IO}$, according to the second definition. This means that there exists a PPT interface $\mathcal{S}$ such that for all PPT environments $\mathcal{Z}$ which does not violate $\mathcal{IO}$ it holds that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \stackrel{c}{\approx} \text{REAL}_{\pi,\mathcal{Z}}$. To prove security according to the first definition we have to prove that there exists a PPT interface $\mathcal{S}$ such that for all PPT environments $\mathcal{Z}$ it holds that $\text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}} \stackrel{c}{\approx} \text{REAL}_{\pi,\mathcal{Z}}$. To do this consider any environment $\mathcal{Z}$. Then consider the self-restricting environment $\mathcal{Z}_{\mathcal{IO}}$ which runs exactly as $\mathcal{Z}$, except that it runs $\mathcal{IO}$ on the outputs of $\mathcal{Z}$ and terminates with a fixed output, 1 say, when the internal copy of $\mathcal{IO}$ outputs `fail`. This guarantees that $\mathcal{Z}_{|\mathcal{IO}}$ never violates the copy of $\mathcal{IO}$ in $\text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}_{|\mathcal{IO}}}$. Therefore $\Pr[\text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}_{|\mathcal{IO}}} = \texttt{fail}] = 0$, and we have that $\text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}_{|\mathcal{IO}}} \stackrel{c}{\approx} \text{REAL}_{\pi,\mathcal{Z}_{|\mathcal{IO}}}$. Now, as long as $\mathcal{Z}$ does not violate $\mathcal{IO}$, clearly $\text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}} = \text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}_{|\mathcal{IO}}}$ and $\text{REAL}_{\pi,\mathcal{Z}} = \text{REAL}_{\pi,\mathcal{Z}_{|\mathcal{IO}}}$. If however $\mathcal{Z}$ violates $\mathcal{IO}$, then $\text{REAL}_{\pi,\mathcal{Z}_{|\mathcal{IO}}}$ outputs 1, whereas $\text{REAL}_{\pi,\mathcal{Z}}$ might run berserk, so we have to take care. However, until $\mathcal{Z}$ violates $\mathcal{IO}$ we have that $\text{REAL}_{\pi,\mathcal{Z}} \stackrel{c}{\approx} \text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}}$, and after $\mathcal{Z}$ violates $\mathcal{IO}$, the simulation $\text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}}$ becomes trivial: All present and future inputs to and outputs from $\mathcal{F}_{|\mathcal{IO}}$ are given to $\mathcal{S}$, and $\mathcal{S}$ gains complete control over $\mathcal{F}_{|\mathcal{IO}}$. Therefore $\mathcal{S}$ can simply run π on the inputs given by $\mathcal{Z}$ and deliver the outputs of π back to $\mathcal{Z}$ through $\mathcal{F}_{|\mathcal{IO}}$. This means that after $\mathcal{Z}$ violates we have that $\text{REAL}_{\pi,\mathcal{Z}} = \text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}}$.

The other direction is trivial. Assume namely that there exists a PPT interface $\mathcal{S}$ such that for all PPT environments $\mathcal{Z}$ it holds that $\text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}} \stackrel{c}{\approx} \text{REAL}_{\pi,\mathcal{Z}}$. Again, as long as $\mathcal{Z}$ does not violate $\mathcal{IO}$, clearly $\text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}} = \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}_{|\mathcal{IO}}}$ and $\text{REAL}_{\pi,\mathcal{Z}} = \text{REAL}_{\pi,\mathcal{Z}_{|\mathcal{IO}}}$, and after $\mathcal{Z}$ violates $\mathcal{IO}$ we have that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}_{|\mathcal{IO}}} = \text{REAL}_{\pi,\mathcal{Z}_{|\mathcal{IO}}}$, as both experiments output 1. Therefore $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}_{|\mathcal{IO}}} \stackrel{c}{\approx} \text{REAL}_{\pi,\mathcal{Z}_{|\mathcal{IO}}}$.

As it turns out, the second argument is sound, but the first one is not. The first definition implies the second, but the second definition does not imply the first, at least not if one-way permutations exists. The problem with the first line of argument is that even though the simulator, when $\mathcal{IO}$ is violated, learns all inputs to π and can deliver all outputs from π back to $\mathcal{Z}$, it is not necessarily the case that it can create a copy of π which is consistent with the values that it showed to $\mathcal{Z}$ *before* $\mathcal{IO}$ was violated.

Theorem 3.4 *If one-way functions exists, bijectively mapping k-bit strings to k-bits strings, then there exists a protocol π, a functionality $\mathcal{F}$ and an IO restriction $\mathcal{IO}$, such that π realizes $\mathcal{F}$ under the IO restriction $\mathcal{IO}$, in the sense defined immediately above, and such that π does not realizes $\mathcal{F}_{|\mathcal{IO}}$.*

Proof. Let f be a one-way permutation mapping k-bit strings to k-bits strings. Consider the functionality which on input `activate` from party P_1 outputs a uniformly random k-bit string y to P_1. Consider the IO restriction $\mathcal{IO}$ that P_1 only receives an input once and that P_1 is never corrupted. Formally, if $\mathcal{IO}$ twice sees an input different from ϵ or it sees that P_1 was corrupted, then it outputs `fail`, otherwise it just outputs ϵ. Consider the protocol $\pi = (P_1)$, where P_1 on input `activate` generates a uniformly random k-bit string x, computes $y = f(x)$ and outputs y. Furthermore, if P_1 ever receives another input different from ϵ, it outputs x. Otherwise it outputs ϵ in each round.

Clearly π realizes $\mathcal{F}$ under the IO restriction $\mathcal{IO}$, in the sense defined immediately above. Let $\mathcal{S}$ be any simulator that does nothing. After the first activation of P_1 the environment $\mathcal{Z}$ sees a uniformly random element in both $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\text{REAL}_{\pi,\mathcal{Z}}$, so until then the simulation is perfect. After that all possible outputs from $\mathcal{Z}$ make both executions behave identically. Clearly so for the command $(\texttt{guess}, b)$, and the commands $(\texttt{corrupt}, 1)$ and $(\texttt{activate}, 1, x)$, for $x \neq \epsilon$, would make $\mathcal{Z}$ violate $\mathcal{IO}$, so we do not have to consider an environment making such outputs. Finally, the command $(\texttt{activate}, 1, \epsilon)$ makes P_1 output ϵ in both executions.

We claim that π does not realize $\mathcal{F}_{|\mathcal{IO}}$. Assume that there exists a PPT interface $\mathcal{S}$ such that for all PPT environments $\mathcal{Z}$ it holds that $\text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}} \overset{\text{c}}{\approx} \text{REAL}_{\pi,\mathcal{Z}}$. This would in particular hold for the environment $\mathcal{Z}$ which first inputs `activate` to P_1, ends the round and saves the output y from P_1, inputs `activate` to P_1, ends the round and saves the output x. Then it checks whether $y = f(x)$, and if so it outputs 0, otherwise it outputs 1. Clearly, $\text{REAL}_{\pi,\mathcal{Z}} = 0$. However, in $\text{IDEAL}_{\mathcal{F}_{|\mathcal{IO}},\mathcal{S},\mathcal{Z}}$ the value y is chosen at random by $\mathcal{F}$ and showed to $\mathcal{Z}$, and when $\mathcal{Z}$ violates $\mathcal{IO}$ the interface is given y and must compute x such that $y = f(x)$ and output x to $\mathcal{Z}$. This would contradict the one-wayness of f. □

We feel that the formalization, where we only quantify over non-violating environments is the better formalization of restricted input-output behavior for a protocol. First of all, it is the weaker notion, and it makes sense. The author feels that this is a strong motivation for adopting it. We get a sound model which deems more protocols secure. Second, and more importantly, some very intuitive results fails to hold if the other notion is adapted, e.g. some results about implementing the broadcast model only holds under the notion that we have adopted. Last, but not least, a slight generalization of the approach allows to define IO restrictions which are not PPT. This will come in handy later.

3.5.4 Formalizing IO Behavior

In this section we formalize our notion of input-output behavior.

3.5.4.1 IO Behavior for the ST

Until now we only discussed restricted IO behavior for a protocol. For an ideal functionality we will in addition to putting a restriction on the inputs of honest parties, also put a restriction on the inputs and outputs on the special tapes (ST) (modeling a.o.t. the inputs of the corrupted parties). We will for two reasons formalize this using a separate mechanism. First of all the inputs to honest parties and the inputs on the SIT are supplied by separate entities: In the ideal process the inputs to honest parties are supplied by $\mathcal{Z}$, whereas the inputs on the SIT are supplied by $\mathcal{S}$, and in the hybrid model, the inputs to honest parties are supplied by π, whereas the inputs on the SIT are supplied by $\mathcal{Z}$. It is therefore separate parties that should be penalized if the IO restriction is violated. Second, we can allow the restriction on the ST to be arbitrary, whereas we only know how to prove the composition theorem for PPT restrictions on the inputs and outputs of the honest parties. We let an IO restriction for the ST $\mathcal{IO}$ be an ITM which is initially given k and is then given all outputs on the SOT of $\mathcal{F}$ and all inputs on the SIT of $\mathcal{F}$ in the order in which they appear; Here $\mathcal{F}$ is the ideal functionality $\mathcal{F}$ being restricted

For reporting that the IO behavior was violated $\mathcal{IO}$ has two reserved symbols ``H $\rightsquigarrow \mathcal{F}$'' and ``C $\rightsquigarrow \mathcal{F}$''. Intuitively, outputting ``H $\rightsquigarrow \mathcal{F}$'' will mean that the IO restriction was violated by the honest parties and will always be output after seeing the $v_{\mathcal{F}}$ value from an activation of a party. Outputting ``C $\rightsquigarrow \mathcal{F}$'' will intuitively mean that the IO restriction was violated by the corrupted parties (being controlled by the environment) and will always be output after seeing the $v_{\mathcal{F}}$ value from an `end round`-command, as $v_{\mathcal{F}}$ specifies the inputs of the honest parties. If at some point $\mathcal{IO}$ outputs ``H $\rightsquigarrow \mathcal{F}$'' or ``C $\rightsquigarrow \mathcal{F}$'' we say that the IO restriction on the ST of $\mathcal{F}$ was violated. Formally we let ``H $\rightsquigarrow \mathcal{F}$'' and ``C $\rightsquigarrow \mathcal{F}$'' be sets of symbols, which can all equally be output to communicate a violation. This will become convenient later as we can use the particular symbol output to indicate the type of violation. If x is a variable ranging over strings, we often use the notation $x =$ ``H $\rightsquigarrow \mathcal{F}$'' to mean $x \in$ ``H $\rightsquigarrow \mathcal{F}$''.

Definition 3.10 *An IO behavior for the ST (of $\mathcal{F}$) is an ITM taking inputs of one of the following four forms: 1) (k, r), where k is the security parameter and r the random bits (this is the first input and is only given once), 2) (i, v), where $i \in \mathbf{N}$ and $v \in \{0,1\}^*$, 3) $(\texttt{corrupt}, i, v)$, where $i \in \mathbf{N}$ and $v \in \{0,1\}^*$, or 4) $(\texttt{activate}, v)$, where $v \in \{0,1\}^*$. The ITM delivers outputs from $\{\epsilon\} \cup$ ``H $\rightsquigarrow \mathcal{F}$'' $\cup$ ``C $\rightsquigarrow \mathcal{F}$'', where ``H $\rightsquigarrow \mathcal{F}$'' and ``C $\rightsquigarrow \mathcal{F}$'' are two disjoint finite sets of reserved symbols not in $\{0,1\}^*$. They only output values from ``H $\rightsquigarrow \mathcal{F}$'' after an input of form 2 or 3 and they only output values from ``C $\rightsquigarrow \mathcal{F}$'' after an input of form 4). We enforce the convention on IO behaviors that if they ever see the input `fail` as a v value in (i, v) or $(\texttt{corrupt}, i, v)$, then they always output a value from ``H $\rightsquigarrow \mathcal{F}$''.*

It is convenient to consider the IO behavior for the ST as part of the ideal functionality.

Functionality $\langle \mathcal{F}, \mathcal{IO} \rangle$

initialization:

On input (k, r), split r into $(r^{\mathcal{F}}, r^{\mathcal{IO}})$ and input $(k, r^{\mathcal{F}})$ to $\mathcal{F}$ and input $(k, r^{\mathcal{IO}})$ to $\mathcal{F}$.

party activation:

On input (i, x_i), proceed as follows:

1. Input (i, x_i) to $\mathcal{F}$ and receive a value v from $\mathcal{F}$.
2. Input (i, v) to $\mathcal{IO}$ and receive a value w.
3. If $w \in$ ``$\mathtt{H} \rightsquigarrow \mathcal{F}$'', then output w. Otherwise, output v.

corrupt:

On input $(\mathtt{corrupt}, i)$, proceed as follows:

1. Input $(\mathtt{corrupt}, i)$ to $\mathcal{F}$ and receive a value v from $\mathcal{F}$.
2. Input $(\mathtt{corrupt}, i, v)$ to $\mathcal{IO}$ and receive a value w.
3. If $w \in$ ``$\mathtt{H} \rightsquigarrow \mathcal{F}$'', then output w. Otherwise, output v.

end round:

On input $(\mathtt{activate}, v)$, proceed as follows:

1. Input $(\mathtt{activate}, v)$ to $\mathcal{F}$ and receive a value $\{t_{i,r}\}_{i \in [n]}$ from $\mathcal{F}$.
2. Input $(\mathtt{activate}, v)$ to $\mathcal{IO}$ and receive a value w.
3. If $w \in$ ``$\mathtt{C} \rightsquigarrow \mathcal{F}$'', then output w. Otherwise, output $\{t_{i,r}\}_{i \in [n]}$.

Figure 3.19: The ideal functionality with IO restriction obtained by composing an ideal functionality $\mathcal{F}$ with an IO behavior $\mathcal{IO}$ for the ST.

Definition 3.11 *An ideal functionality with IO restriction $\langle \mathcal{F}, \mathcal{IO} \rangle$ is an ITM given by an ideal functionality $\mathcal{F}$ and an IO behavior for the ST $\mathcal{IO}$, as in Fig. 3.19. Given an ideal functionality with IO restriction $\langle \mathcal{F}, \mathcal{IO} \rangle$ we call $\mathcal{F}$ the core functionality of $\langle \mathcal{F}, \mathcal{IO} \rangle$ and call $\mathcal{IO}$ the IO restriction of $\langle \mathcal{F}, \mathcal{IO} \rangle$.*

In the following, when the IO restriction is given by the context, we often denote the ideal functionality with IO restriction $\langle \mathcal{F}, \mathcal{IO}(\mathcal{F}) \rangle$ by $\mathcal{F}$. When $\mathcal{F}$ denotes an ideal functionality with IO restriction we use $\mathcal{IO}(\mathcal{F})$ to denote the IO restriction of $\mathcal{F}$.

Jumping ahead a bit, we will later change the definition of the executions, real, hybrid and ideal so that if an ideal functionality with IO restriction ever outputs ``$\mathtt{H} \rightsquigarrow \mathcal{F}$'' or ``$\mathtt{C} \rightsquigarrow \mathcal{F}$'', then the execution is stopped immediately, with output ``$\mathtt{H} \rightsquigarrow \mathcal{F}$'' respectively ``$\mathtt{C} \rightsquigarrow \mathcal{F}$''.

Notice that an ideal functionality with IO restriction $\langle \mathcal{F}, \mathcal{IO} \rangle$ not necessarily is an ideal functionality, as $\mathcal{IO}$ and therefore $\langle \mathcal{F}, \mathcal{IO} \rangle$ need not be PPT. We will later show that this is not a problem.

3.5.4.2 IO Behavior for a Protocol Via IO Behavior for the ST

Before we do this we will however make an observation which allows us to consider only the IO restriction of the ST. Assume that we are given an IO restriction for a protocol $\mathcal{IO}$. This allows us to consider the restricted ideal functionality $\mathcal{F}_{|\mathcal{IO}}$, which breaks down each time $\mathcal{IO}$ is violated, see Fig. 3.16 on page 90. Assume that we are furthermore given an IO restriction for the ST $\mathcal{IO}'$. This allows us to consider the ideal functionality with IO restriction $\langle \mathcal{F}_{|\mathcal{IO}}, \mathcal{IO}' \rangle$. We already discussed that if this ideal functionality with IO restriction is run in a hybrid model and it outputs ‘‘H $\rightsquigarrow \mathcal{F}$’’ or ‘‘C $\rightsquigarrow \mathcal{F}$’’, because $\mathcal{IO}'$ was violated, then the execution is stopped. We now also have the possibility that $\mathcal{IO}$ is violated and that $\langle \mathcal{F}_{|\mathcal{IO}}, \mathcal{IO}' \rangle$ then outputs `fail`. To avoid having to deal with this extra case we enforced the convention on IO restrictions for the ST that if they ever see the input `fail` as a v value after a party activation or a corruption, then it outputs ‘‘H $\rightsquigarrow \mathcal{F}$’’, and if they ever see the input `fail` as a v value after an `end round`-activation, then it outputs ‘‘C $\rightsquigarrow \mathcal{F}$’’. This means that if $\mathcal{IO}$ is ever violated and $\mathcal{F}_{|\mathcal{IO}}$ outputs `fail`, then $\langle \mathcal{F}_{|\mathcal{IO}}, \mathcal{IO}' \rangle$ will output ‘‘H $\rightsquigarrow \mathcal{F}$’’ or ‘‘C $\rightsquigarrow \mathcal{F}$’’. The intuition for which symbol is output when is as follows: If `fail` is output after a party activation or a corruption, then the entity controlling the honest parties broke the IO behavior, which is reported by outputting ‘‘H $\rightsquigarrow \mathcal{F}$’’. If `fail` is output after an `end round`-activation, then the entity controlling the corrupted parties broke the IO behavior, which is reported by outputting ‘‘C $\rightsquigarrow \mathcal{F}$’’. Since $\mathcal{F}_{|\mathcal{IO}}$ is in itself an ideal functionality (as an IO restriction for a protocol is required to be PPT) we can avoid dealing further with IO restrictions on the honest parties. We simply allow that in specifying an ideal functionality $\mathcal{F}$ one is allowed to make it output `fail` to indicate that the expected IO restriction was violated. In this case the execution in which it takes part will terminate with output ‘‘H $\rightsquigarrow \mathcal{F}$’’ or ‘‘C $\rightsquigarrow \mathcal{F}$’’.

When considering the IO restriction for the ST on the other hand we are not guaranteed that $\langle \mathcal{F}, \mathcal{IO} \rangle$ is again PPT. We therefore have to carefully extend our model of execution and reprove the composition theorem for the new model.

3.5.5 The Real-Life Model and the Ideal Process with Restricted IO

Let $\langle \mathcal{F}, \mathcal{IO} \rangle$ be an ideal functionality with IO restriction and let π be a real-life protocol. We formalize what it means for π to realize $\langle \mathcal{F}, \mathcal{IO} \rangle$ in the real-life model. For this purpose we have to extend the real-life model and the ideal process with IO restriction.

Consider first the ideal process $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$. We want to detect when $\mathcal{IO}$ is violated and then terminate the simulation when this happens. To do this we simply run the process $\text{IDEAL}_{\langle \mathcal{F},\mathcal{IO} \rangle,\mathcal{S},\mathcal{Z}}$ according to Fig. 3.2 on page 73. We then define the output of $\text{IDEAL}_{\langle \mathcal{F},\mathcal{IO} \rangle,\mathcal{S},\mathcal{Z}}$ to be b if $\mathcal{Z}$ outputs b, but add the rule that if during the process $\langle \mathcal{F}, \mathcal{IO} \rangle$ outputs $v \in$ ‘‘H $\rightsquigarrow \mathcal{F}$’’$\cup$‘‘C $\rightsquigarrow \mathcal{F}$’’, then the output of $\text{IDEAL}_{\langle \mathcal{F},\mathcal{IO} \rangle,\mathcal{S},\mathcal{Z}}$ is v. This means that $\text{IDEAL}_{\langle \mathcal{F},\mathcal{IO} \rangle,\mathcal{S},\mathcal{Z}} \in \{0,1\} \cup$ ‘‘H $\rightsquigarrow \mathcal{F}$’’ $\cup$ ‘‘C $\rightsquigarrow \mathcal{F}$’’.

Recall that for an IO restriction for a protocol $\mathcal{IO}$ we preferred the formalization that only quantified over environments that did not violate $\mathcal{IO}$. We apply the same strategy here. The output of $\text{IDEAL}_{\langle \mathcal{F},\mathcal{IO} \rangle,\mathcal{S},\mathcal{Z}}$ is a value from $\{0,1\} \cup$ ‘‘H $\rightsquigarrow \mathcal{F}$’’ $\cup$ ‘‘C $\rightsquigarrow \mathcal{F}$’’. If the output is 0 or 1, then $\mathcal{IO}$ was not violated. If the output is from ‘‘C $\rightsquigarrow \mathcal{F}$’’, then $\mathcal{IO}$ was violated by $\mathcal{S}$ while specifying the inputs of corrupted parties. Therefore an output from

$\text{``C} \rightsquigarrow \mathcal{F}\text{''}$ should not be interpreted as a violation of $\mathcal{IO}$ by $\mathcal{Z}$. However, if the output is from $\text{``H} \rightsquigarrow \mathcal{F}\text{''}$, then $\mathcal{IO}$ was violated by $\mathcal{Z}$ while specifying the inputs of honest parties. In particular we have that if $\mathcal{F}$ outputs `fail` after seeing the input of an honest party, then $\langle \mathcal{F}, \mathcal{IO} \rangle$, and therefore $\text{IDEAL}_{\langle \mathcal{F}, \mathcal{IO} \rangle, \mathcal{S}, \mathcal{Z}}$, will output a value from $\text{``H} \rightsquigarrow \mathcal{F}\text{''}$. Therefore an output from $\text{``H} \rightsquigarrow \mathcal{F}\text{''}$ should be interpreted as a violation of $\mathcal{IO}$ by $\mathcal{Z}$. We therefore only wants to quantify over $\mathcal{Z}$ which does not make $\mathcal{IO}$ output a value from $\text{``H} \rightsquigarrow \mathcal{F}\text{''}$. This leads us to the following formalization of security.

Definition 3.12 *Let $\mathcal{S}$ be an interface, let π be a real-life protocol and let $\langle \mathcal{F}, \mathcal{IO} \rangle$ be an ideal functionality with IO restriction. We say that $\mathcal{S}$ can produce π given $\mathcal{F}$, written $\mathcal{S} : \langle \mathcal{F}, \mathcal{IO} \rangle \triangleleft \pi$, if for all $\mathcal{G}$-hybrid environments $\mathcal{Z}$ where $\Pr[\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \in \text{``H} \rightsquigarrow \mathcal{F}\text{''}]$ it holds that $\text{IDEAL}_{\langle \mathcal{F}, \mathcal{IO} \rangle, \mathcal{S}, \mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\pi, \mathcal{Z}}$.*

The reason why we only required from $\mathcal{Z}$ that $\Pr[\text{IDEAL}_{\langle \mathcal{F}, \mathcal{IO} \rangle, \mathcal{S}, \mathcal{Z}} \in \text{``H} \rightsquigarrow \mathcal{F}\text{''}] \overset{c}{\approx} 0$, as opposed to $\Pr[\text{IDEAL}_{\langle \mathcal{F}, \mathcal{IO} \rangle, \mathcal{S}, \mathcal{Z}} \in \text{``H} \rightsquigarrow \mathcal{F}\text{''}] = 0$, is technical. It gives a slightly stronger security notion as we quantify over more environments. This strengthening is needed to prove the composition theorem.

Notice that we did not change the real-life model. In particular we have that $\text{REAL}_{\pi,\mathcal{Z}} \in \{0,1\}$, so $\Pr[\text{REAL}_{\pi,\mathcal{Z}} \in \text{``C} \rightsquigarrow \mathcal{F}\text{''}] = 0$. This means that if $\text{IDEAL}_{\langle \mathcal{F}, \mathcal{IO} \rangle, \mathcal{S}, \mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\pi,\mathcal{Z}}$, then in particular $\Pr[\text{IDEAL}_{\langle \mathcal{F}, \mathcal{IO} \rangle, \mathcal{S}, \mathcal{Z}} \in \text{``C} \rightsquigarrow \mathcal{F}\text{''}] \overset{c}{\approx} 0$. This means that if $\mathcal{S}$ can produce π from $\langle \mathcal{F}, \mathcal{IO} \rangle$, then for any environment $\mathcal{Z}$ it holds that if $\Pr[\text{IDEAL}_{\langle \mathcal{F}, \mathcal{IO} \rangle, \mathcal{S}, \mathcal{Z}} \in \text{``H} \rightsquigarrow \mathcal{F}\text{''}] \overset{c}{\approx} 0$, then $\Pr[\text{IDEAL}_{\langle \mathcal{F}, \mathcal{IO} \rangle, \mathcal{S}, \mathcal{Z}} \in \text{``C} \rightsquigarrow \mathcal{F}\text{''}] \overset{c}{\approx} 0$. In other words, $\mathcal{S}$ will never be the first to violate $\mathcal{IO}$. This is of course a reasonable restriction on the interface.

Definition 3.13 *Let π be a real-life protocol and let $\langle \mathcal{F}, \mathcal{IO} \rangle$ be an ideal functionality with IO restriction. We say that π realizes $\mathcal{F}$ if there exists a PPT $\mathcal{G}$-hybrid interface $\mathcal{S}$ such that $\mathcal{S} : \langle \mathcal{F}, \mathcal{IO} \rangle \triangleleft \pi$.*

3.5.6 The Hybrid Model With Restricted IO

We now extend the hybrid models and the hybrid ideal process accordingly. Let $\langle \mathcal{F}, \mathcal{IO}(\mathcal{F}) \rangle$ and $\langle \mathcal{G}, \mathcal{IO}(\mathcal{G}) \rangle$ be ideal functionalities with IO restriction and let π be a $\mathcal{G}$-hybrid protocol. We formalize what it means for π to realize $\langle \mathcal{F}, \mathcal{IO}(\mathcal{F}) \rangle$ in the $\langle \mathcal{G}, \mathcal{IO}(\mathcal{G}) \rangle$-hybrid model.

We consider first the $\langle \mathcal{G}, \mathcal{IO}(\mathcal{G}) \rangle$-hybrid model. Similar to above we simply consider the execution $\text{HYB}_{\pi,\mathcal{Z}}^{\langle \mathcal{G}, \mathcal{IO}(\mathcal{G}) \rangle}$ and we say that the output of $\text{HYB}_{\pi,\mathcal{Z}}^{\langle \mathcal{G}, \mathcal{IO}(\mathcal{G}) \rangle}$ is the output of $\mathcal{Z}$, unless $\mathcal{IO}(\mathcal{G})$ ever outputs a value from $\text{``H} \rightsquigarrow \mathcal{G}\text{''}$ or $\text{``C} \rightsquigarrow \mathcal{G}\text{''}$, in which case the output of $\text{HYB}_{\pi,\mathcal{Z}}^{\langle \mathcal{G}, \mathcal{IO}(\mathcal{G}) \rangle}$ is the value from $\text{``H} \rightsquigarrow \mathcal{G}\text{''}$ respectively $\text{``C} \rightsquigarrow \mathcal{G}\text{''}$. We use $\text{``H} \rightsquigarrow \mathcal{G}\text{''}$ and $\text{``C} \rightsquigarrow \mathcal{G}\text{''}$ as the reserved sets of symbols of $\mathcal{G}$ and assume that they are disjoint from the reserved sets of symbols $\text{``H} \rightsquigarrow \mathcal{F}\text{''}$ respectively $\text{``C} \rightsquigarrow \mathcal{F}\text{''}$ of $\mathcal{F}$.

For the hybrid ideal process for the $\langle \mathcal{G}, \mathcal{IO}(\mathcal{G}) \rangle$-hybrid model we consider the execution $\text{IDEAL}_{\langle \mathcal{F}, \mathcal{IO}(\mathcal{F}) \rangle, \mathcal{S}, \mathcal{Z}}^{\langle \mathcal{G}, \mathcal{IO}(\mathcal{G}) \rangle}$. Remember that for the symbol $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}^{\mathcal{G}}$, the machine $\mathcal{G}$ does not actually occur in the process. This will not be the case for $\text{IDEAL}_{\langle \mathcal{F}, \mathcal{IO}(\mathcal{F}) \rangle, \mathcal{S}, \mathcal{Z}}^{\langle \mathcal{G}, \mathcal{IO}(\mathcal{G}) \rangle}$, where $\mathcal{IO}(\mathcal{G})$ will play an actual part of the process. The reason for this is that a copy of $\mathcal{G}$ is being

Algorithm $\mathrm{IDEAL}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}}(k,r,z)$

initialize:

The input to an ideal process is the security parameter k, the random bits $(r_{\mathcal{F}}, r_{\mathcal{IO}(\mathcal{F})})$, $r_{\mathcal{T}}$, $r_{\mathcal{IO}(\mathcal{G})}$ used by $\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle$, $\mathcal{T}$ and $\mathcal{IO}(\mathcal{G})$ and an auxiliary input $z \in \{0,1\}^*$ and random bits $r_{\mathcal{Z}}$ for $\mathcal{Z}$.

Initialize the round counter $r = 1$ and initialize the set of corrupted parties $C = \emptyset$.

Provide the various machines with their random inputs and k and give k, z and $r_{\mathcal{Z}}$ to $\mathcal{Z}$ and activate $\mathcal{Z}$.

environment activation:

$\mathcal{Z}$ is defined exactly as in the hybrid model in Fig. 3.4 on page 76, but now the commands are handled by $\mathcal{T}$, as described below.

party activation:

On $(\texttt{activate}, i, x_{i,r}, \{m_{j,i,r-1}\}_{j\in C})$, input $(i, x_{i,r})$ to $\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle$ and receive $v_{\mathcal{F}}$ on the SOT of $\mathcal{F}$. **If** $v_{\mathcal{F}} \in$ ``$\texttt{H} \rightsquigarrow \mathcal{F}$''**, then terminate the execution with output** $v_{\mathcal{F}}$**.** Input $(\texttt{activate}, i, v_{\mathcal{F}}, \{m_{j,i,r-1}\}_{i\in C})$ to $\mathcal{T}$ to generate $(\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}, v_{\mathcal{G}})$. **Input** $(i, v_{\mathcal{G}})$ **to** $\mathcal{IO}(\mathcal{G})$ **and receive a value** w**. If** $w \in$ ``$\texttt{H} \rightsquigarrow \mathcal{G}$''**, then terminate the execution with output** w**.** Input $(\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}, v_{\mathcal{G}})$ to $\mathcal{Z}$.

corrupt:

On $(\texttt{corrupt}, i)$, input $(\texttt{corrupt}, i)$ on the SIT of $\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle$ and receive $v_{\mathcal{F}}$ on the SOT of $\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle$. **If** $v_{\mathcal{F}} \in$ ``$\texttt{H} \rightsquigarrow \mathcal{F}$''**, then terminate the execution with output** $v_{\mathcal{F}}$**.** Input $(\texttt{corrupt}, i, v_{\mathcal{F}})$ to $\mathcal{T}$ and receive a value $(r_i, v_{\mathcal{G}})$. **Give** $(\texttt{corrupt}, i, v_{\mathcal{G}})$ **to** $\mathcal{IO}(\mathcal{G})$ **and receive a value** w**. If** $w \in$ ``$\texttt{H} \rightsquigarrow \mathcal{G}$''**, then terminate the execution with output** w**.** Input $(r_i, v_{\mathcal{G}})$ to $\mathcal{Z}$. Set $C = C \cup \{i\}$.

end round:

On $(\texttt{end round}, v_{\mathcal{G}})$, **input** $(\texttt{activate}, v_{\mathcal{G}})$ **to** $\mathcal{IO}(\mathcal{G})$ **and receive a value** w**. If** $w \in$ ``$\texttt{C} \rightsquigarrow \mathcal{G}$''**, then terminate the execution with output** w**.** Input $(\texttt{end round}, v_{\mathcal{G}})$ to $\mathcal{T}$ to generate $v_{\mathcal{F}}$. Input $(\texttt{activate}, v_{\mathcal{F}})$ to $\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle$ to generate $w \in$ ``$\texttt{C} \rightsquigarrow \mathcal{F}$'' $\cup \{\{y_{i,r}\}_{i\in[n]}\}$. **If** $w \in$ ``$\texttt{C} \rightsquigarrow \mathcal{F}$''**, then terminate the execution with output** w**.** Input $\{y_{i,r}\}_{i\in C}$ to $\mathcal{T}$ to generate $\{t^{\mathcal{G}}_{i,r}\}_{i\in C}$. Then input $(\{y_{i,r}\}_{i\in H}, \{t^{\mathcal{G}}_{i,r}\}_{i\in C})$ to $\mathcal{Z}$. Set $r = r + 1$.

Figure 3.20: The ideal process with a hybrid-adversary.

simulated by $\mathcal{S}$ to $\mathcal{Z}$ and that $\mathcal{Z}$ inputs values to this simulated version of $\mathcal{G}$. Therefore we have to verify that the simulated version of $\mathcal{G}$ is used correctly by $\mathcal{Z}$. The details of how this is done is given in Fig. 3.20.

Definition 3.14 *Let $\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle$ and $\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle$ be ideal functionalities with IO restriction and let π be a $\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle$-hybrid protocol. Let $\mathcal{T}$ be a hybrid interface. We say that $\mathcal{T}$ can produce $\pi[\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle]$ given $\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle$, written $\mathcal{T} : \langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle \triangleleft \pi[\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle]$, if for all $\mathcal{G}$-hybrid environments $\mathcal{Z}$ where* $\Pr[\mathrm{IDEAL}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}} \in$ ``$\texttt{H} \rightsquigarrow \mathcal{F}$''$] \overset{\mathrm{c}}{\approx} 0$ *and* $\mathrm{IDEAL}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}} \overset{\mathrm{c}}{\approx} \mathrm{HYB}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}_{\pi,\mathcal{Z}}$.

Notice that from $\Pr[\mathrm{IDEAL}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}} \in$ ``$\texttt{H} \rightsquigarrow \mathcal{G}$''$] \overset{\mathrm{c}}{\approx} 0$ and $\mathrm{IDEAL}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}} \overset{\mathrm{c}}{\approx}$

$\mathrm{HYB}_{\pi,\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}$ it follows that $\Pr[\mathrm{HYB}_{\pi,\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}\text{''}] \overset{c}{\approx} 0$, meaning that π uses $\mathcal{G}$ correctly in $\mathrm{HYB}_{\pi,\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}$, except with negligible probability. Therefore, if $\mathcal{T}$ can produce $\pi[\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle]$, then π used the ideal functionality $\mathcal{G}_1$ correctly.

There might be one surprise in the above definition. Why do we not require from $\mathcal{Z}$ that

$$\Pr[\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}\text{''} \cup \text{``}\mathtt{C} \rightsquigarrow \mathcal{G}\text{''}] \overset{c}{\approx} 0 \ ,$$

instead of just

$$\Pr[\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}\text{''}] \overset{c}{\approx} 0 \ ,$$

and why do we not require from $\mathcal{T}$ that

$$\Pr[\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle} \in \text{``}\mathtt{C} \rightsquigarrow \mathcal{F}\text{''} \cup \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}\text{''}] \overset{c}{\approx} 0 \ ,$$

instead of just

$$\Pr[\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}\text{''}] \overset{c}{\approx} 0 \ .$$

The second question is the easier one to answer. The experiment $\mathrm{HYB}_{\pi,\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}$ never outputs ``$\mathtt{C} \rightsquigarrow \mathcal{F}$'', so from

$$\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle} \overset{c}{\approx} \mathrm{HYB}_{\pi,\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}$$

it follows that

$$\Pr[\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle} \in \text{``}\mathtt{C} \rightsquigarrow \mathcal{F}\text{''}] \overset{c}{\approx} 0 \ .$$

Then for the first question. What about the condition

$$\Pr[\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle} \in \text{``}\mathtt{C} \rightsquigarrow \mathcal{G}\text{''}] \overset{c}{\approx} 0 \ ?$$

It would seem reasonable to require this as it is $\mathcal{Z}$ which inputs to the corrupted parties of the simulated $\mathcal{G}$ in $\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}$, and $\Pr[\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle} \in \text{``}\mathtt{C} \rightsquigarrow \mathcal{G}\text{''}] \overset{c}{\approx} 0$ does not follow from $\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle} \overset{c}{\approx} \mathrm{HYB}_{\pi,\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}$, as $\mathrm{HYB}_{\pi,\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}$ outputs ``$\mathtt{C} \rightsquigarrow \mathcal{G}$'' whenever $\mathcal{Z}$ violates $\mathcal{IO}(\mathcal{G})$. As of now we allow environments which violates $\mathcal{IO}(\mathcal{G})$. There is a reason for this which we return to below. But notice that it is not a serious problem for the simulator to consider environments which violates $\mathcal{IO}(\mathcal{G})$ as both $\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}$ and $\mathrm{HYB}_{\pi,\mathcal{Z}}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle}$ terminate with output ``$\mathtt{C} \rightsquigarrow \mathcal{G}$'' when $\mathcal{IO}(\mathcal{G})$ is violated. Therefore the simulator only has to simulate successfully up to the point where $\mathcal{IO}(\mathcal{G})$ is violated.

Definition 3.15 *Let $\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle$ and $\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle$ be ideal functionalities with IO restriction and let π be a $\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle$-hybrid protocol. We say that π realizes $\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle$ in the $\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle$-hybrid model if there exists a PPT hybrid interface $\mathcal{T}$ such that $\mathcal{T} : \langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle \triangleleft \pi[\langle\mathcal{G},\mathcal{IO}(\mathcal{G})\rangle]$.*

3.5.7 Multiple Ideal Functionalities with Restricted IO

In this section we describe how to deal with hybrid models with multiple functionalities. We extend the approach in Section 3.3.1 to also consider IO restrictions. We first extend Definition 3.5 on page 76.

Definition 3.16 *Let $\mathcal{IO}(\mathcal{G}_1)$ and $\mathcal{IO}(\mathcal{G}_2)$ be IO restriction. The* double IO restriction *$\mathcal{IO} = [\mathcal{IO}(\mathcal{G}_1), \mathcal{IO}(\mathcal{G}_2)]$ is defined as follows: On input (k, r) it splits r into r_1 and r_2 and inputs (k, r_m) to $\mathcal{IO}(\mathcal{G}_m)$, where m ranges over $1, 2$. On input (i, v) it parses v as (v_1, v_2) and inputs (i, v_m) to $\mathcal{IO}(\mathcal{G}_m)$. If $\mathcal{IO}(\mathcal{G}_1)$ outputs $x \in$ ``$\mathtt{H} \rightsquigarrow \mathcal{G}_1$'', then $\mathcal{IO}$ outputs $x$; Otherwise, if $\mathcal{IO}(\mathcal{G}_2)$ outputs $x \in$ ``$\mathtt{H} \rightsquigarrow \mathcal{G}_2$'', then $\mathcal{IO}$ outputs x; Otherwise, $\mathcal{IO}$ outputs ϵ. On input $(\mathtt{corrupt}, i)$ it inputs $(\mathtt{corrupt}, i)$ to $\mathcal{IO}(\mathcal{G}_m)$. The output is defined as for (i, v) above. On input $(\mathtt{activate}, v)$ it parses v as (v_1, v_2) and inputs $(\mathtt{activate}, v_m)$ to $\mathcal{IO}(\mathcal{G}_m)$. If $\mathcal{IO}(\mathcal{G}_1)$ outputs $x \in$ ``$\mathtt{C} \rightsquigarrow \mathcal{G}_1$'', then $\mathcal{IO}$ outputs $x$; Otherwise, if $\mathcal{IO}(\mathcal{G}_2)$ outputs $x \in$ ``$\mathtt{C} \rightsquigarrow \mathcal{G}_2$'', then $\mathcal{IO}$ outputs x; Otherwise, $\mathcal{IO}$ outputs ϵ. We define the* multi IO restriction *$[\mathcal{IO}(\mathcal{G}_1), \mathcal{IO}(\mathcal{G}_2), \ldots, \mathcal{IO}(\mathcal{G}_l)]$ as follows: If $l = 0$, we write $\mathcal{IO} = []$, then we let $\mathcal{IO} = \mathcal{IO}_{\text{triv}}$, where $\mathcal{IO}_{\text{triv}}$ is the* trivial IO restriction *which always outputs ϵ, except when it sees the input $\mathtt{fail}$, in which case it must by definition output a symbol from ``$\mathtt{H} \rightsquigarrow \cdot$'' or ``$\mathtt{C} \rightsquigarrow \cdot$'' — we let ``$\mathtt{H} \rightsquigarrow \cdot$'' $= \{2\}$ and ``$\mathtt{C} \rightsquigarrow \cdot$'' $= \{2\}$, and if $\mathtt{fail}$ is seen after a party activation or a corruption, then we let $\mathcal{IO}_{\text{triv}}$ output 2 and if if $\mathtt{fail}$ is seen after an* end round-activation, *then we let $\mathcal{IO}_{\text{triv}}$ output 3. When $l = 1$ we let $\mathcal{IO} = \mathcal{IO}(\mathcal{G}_1)$. And, when $l \geq 2$, we let $\mathcal{IO} = [\mathcal{IO}(\mathcal{G}_1), [\mathcal{IO}(\mathcal{G}_2), \ldots, \mathcal{IO}(\mathcal{G}_l)]]$. The* multi ideal functionality with IO restriction *$\mathcal{G} = [\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle, \ldots, \langle \mathcal{G}_l, \mathcal{IO}(\mathcal{G}_l)\rangle]$ is defined to be the ideal functionality with IO restriction $\langle [\mathcal{G}_1, \ldots, \mathcal{G}_l], [\mathcal{IO}(\mathcal{G}_1), \ldots, \mathcal{IO}(\mathcal{G}_l)]\rangle$.*

Definition 3.17 *Let $\mathcal{F}$, $\mathcal{G}_1, \ldots, \mathcal{G}_l$ be ideal functionalities with IO restriction and let π be a hybrid protocol. We say that π realizes $\mathcal{F}$ in the $(\mathcal{G}_1, \ldots, \mathcal{G}_l)$-hybrid model if π realizes $\mathcal{F}$ in the $[\mathcal{G}_1, \ldots, \mathcal{G}_l]$-hybrid model.*

It is straight-forward to verify the following lemma.

Lemma 3.5 *Let $\mathcal{F}$, $\mathcal{G}_1, \mathcal{G}_2$ be ideal functionalities with IO restriction and let π be a protocol for the $[\mathcal{G}_1, \mathcal{G}_2]$-hybrid model. We can consider π a protocol for the $[\mathcal{G}_2, \mathcal{G}_1]$-hybrid model as in Lemma 3.1 on page 78. With this convention we have that π realizes $\mathcal{F}$ in the $[\mathcal{G}_1, \mathcal{G}_2]$-hybrid model iff π realizes $\mathcal{F}$ in the $[\mathcal{G}_2, \mathcal{G}_1]$-hybrid model. Let $\mathcal{F}$, $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ be ideal functionalities with IO restriction and let π be a protocol for the $[\mathcal{G}_1, [\mathcal{G}_2, \mathcal{G}_3]]$-hybrid model. We can consider π as a protocol for the $[[\mathcal{G}_1, \mathcal{G}_2], \mathcal{G}_3]$-hybrid protocol as in Lemma 3.1 on page 78. With this convention π realizes $\mathcal{F}$ in the $[\mathcal{G}_1, [\mathcal{G}_2, \mathcal{G}_3]]$-hybrid model if π realizes $\mathcal{F}$ in the $[[\mathcal{G}_1, \mathcal{G}_2], \mathcal{G}_3]$-hybrid model.*

Theorem 3.5 *Let $\mathcal{F}$, $\mathcal{G}_1, \ldots, \mathcal{G}_l, \mathcal{H}_1, \ldots, \mathcal{H}_m$ be ideal functionalities with IO restriction, where the IO restrictions of $\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}, \mathcal{G}_{i+1}, \mathcal{G}_l, \mathcal{H}_1, \ldots, \mathcal{H}_m$ are PPT, for some $i \in \{1, \ldots, l\}$. Let π be a $(\mathcal{G}_1, \ldots, \mathcal{G}_l)$-hybrid protocol and let γ by a $(\mathcal{H}_1, \ldots, \mathcal{H}_m)$-hybrid protocol. Assume that γ realizes $\mathcal{G}_i$ in the $(\mathcal{H}_1, \ldots, \mathcal{H}_m)$-hybrid model and assume that π realizes $\mathcal{F}$ in the $(\mathcal{G}_1, \ldots, \mathcal{G}_l)$-hybrid model. Then $\pi[\gamma/\mathcal{G}_i]$ realizes $\mathcal{F}$ in the $(\mathcal{G}_1, \ldots, \mathcal{G}_{i-1}, \mathcal{H}_1, \ldots, \mathcal{H}_m, \mathcal{G}_{i+1}, \ldots, \mathcal{G}_l)$-hybrid model.*

Proof. The claim follows directly from Lemma 3.8 on page 104, using Lemma 3.5 on the preceding page. □

Notice that we allow that the IO restriction of the ideal functionality being substituted, $\mathcal{G}_i$, has a super-polynomial time complexity.

3.5.8 Conjunction and Disjunction of IO Restrictions

To make it easier to manipulate IO restrictions we define the conjunction $\mathcal{IO}_1 \wedge \mathcal{IO}_2$ and the disjunction $\mathcal{IO}_1 \vee \mathcal{IO}_2$ of IO restrictions. We define the compositions such that $\mathcal{IO}_1 \wedge \mathcal{IO}_2$ is not violated iff $\mathcal{IO}_1$ is not violated and $\mathcal{IO}_2$ is not violated and we want that $\mathcal{IO}_1 \vee \mathcal{IO}_2$ is not violated iff $\mathcal{IO}_1$ is not violated or $\mathcal{IO}_2$ is not violated.

Definition 3.18 *Let $\mathcal{IO}(\mathcal{G}_1)$ and $\mathcal{IO}(\mathcal{G}_2)$ be IO restrictions. The conjunction of IO restrictions $\mathcal{IO} = \mathcal{IO}(\mathcal{G}_1) \wedge \mathcal{IO}(\mathcal{G}_2)$ is defined as follows: Let the set $\text{``}\mathtt{H} \rightsquigarrow \cdot\text{''}$ for $\mathcal{IO}(\mathcal{G}_1) \wedge \mathcal{IO}(\mathcal{G}_2)$ be $\text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_1\text{''} \cup \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_2\text{''}$ and let the set $\text{``}\mathtt{C} \rightsquigarrow \cdot\text{''}$ be $\text{``}\mathtt{C} \rightsquigarrow \mathcal{G}_1\text{''} \cup \text{``}\mathtt{C} \rightsquigarrow \mathcal{G}_2\text{''}$. On input (k, r) it splits r into r_1 and r_2 and inputs (k, r_m) to $\mathcal{IO}(\mathcal{G}_m)$, where m ranges over $1, 2$. On input (i, v) it inputs (i, v) to $\mathcal{IO}(\mathcal{G}_m)$. If $\mathcal{IO}(\mathcal{G}_1)$ outputs $x \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_1\text{''}$, then $\mathcal{IO}(\mathcal{G}_1) \wedge \mathcal{IO}(\mathcal{G}_2)$ outputs $x$; Otherwise, if $\mathcal{IO}(\mathcal{G}_2)$ outputs $x \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_2\text{''}$, then $\mathcal{IO}(\mathcal{G}_1) \wedge \mathcal{IO}(\mathcal{G}_2)$ outputs x; Otherwise, $\mathcal{IO}(\mathcal{G}_1) \wedge \mathcal{IO}(\mathcal{G}_2)$ outputs ϵ. The remaining commands are handled equivalently, checking $\mathcal{IO}(\mathcal{G}_1)$ before $\mathcal{IO}(\mathcal{G}_2)$. The disjunction of IO restrictions $\mathcal{IO} = \mathcal{IO}(\mathcal{G}_1) \vee \mathcal{IO}(\mathcal{G}_2)$ is defined as follows: Let the set $\text{``}\mathtt{H} \rightsquigarrow \cdot\text{''}$ for $\mathcal{IO}(\mathcal{G}_1) \vee \mathcal{IO}(\mathcal{G}_2)$ be $\text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_1\text{''} \times \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_2\text{''}$ and let the set $\text{``}\mathtt{C} \rightsquigarrow \cdot\text{''}$ be $\text{``}\mathtt{C} \rightsquigarrow \mathcal{G}_1\text{''} \times \text{``}\mathtt{C} \rightsquigarrow \mathcal{G}_2\text{''}$. On input (k, r) it splits r into r_1 and r_2 and inputs (k, r_m) to $\mathcal{IO}(\mathcal{G}_m)$, where m ranges over $1, 2$. On input (i, v) it inputs (i, v) to $\mathcal{IO}(\mathcal{G}_m)$. If $\mathcal{IO}(\mathcal{G}_1)$ outputs $x_1 \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_1\text{''}$ and $\mathcal{IO}(\mathcal{G}_2)$ outputs $x_2 \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_2\text{''}$, then $\mathcal{IO}(\mathcal{G}_1) \vee \mathcal{IO}(\mathcal{G}_2)$ outputs (x_1, x_2); Otherwise $\mathcal{IO}(\mathcal{G}_1) \vee \mathcal{IO}(\mathcal{G}_2)$ outputs ϵ. The remaining commands are handled equivalently.*

3.5.9 Composing an Interface with an Environment

We proceed to prove the composition theorem with IO restriction. For this purpose we modify our notions of composition of the entities in the framework to deal with IO restriction. Our definition of what is a protocol and what is an interface have not change, and therefore we can maintain the definition of composition of protocols $\pi[\gamma/\mathcal{G}_1]$ and the definition of composition of an interface with and interface $\mathcal{T}[\mathcal{S}/\mathcal{G}_1]$. It turns out that we have to change the definition of composing an interface with an environment $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ and the definition of composing an environment with a protocol $\mathcal{Z}[\pi, \mathcal{G}]$. In this section we redefine the notion of composing an interface with an environment. Basically we add to the previous definition that $\mathcal{Z}[\mathcal{S}/\mathcal{H}]$ checks the IO restriction of $\mathcal{H}$ and terminates the execution if it is violated. The details are given in Fig. 3.21 on page 103. Notice that we allow $\mathcal{Z}[\mathcal{S}/\langle \mathcal{H}, \mathcal{IO}(\mathcal{H}) \rangle]$ to output values from the finite domain $\{0, 1\} \cup \text{``}\mathtt{H} \rightsquigarrow \mathcal{H}\text{''} \cup \text{``}\mathtt{C} \rightsquigarrow \mathcal{H}\text{''}$. This means that $\mathcal{Z}[\mathcal{S}/\langle \mathcal{H}, \mathcal{IO}(\mathcal{H}) \rangle]$ is not an environment in the sense that environments only output values from $\{0, 1\}$. We deal with this later, using Lemma 2.3 on page 12.

Lemma 3.6 *For all $(\mathcal{H},\mathcal{G}_2)$-hybrid environments $\mathcal{Z}$, all $\mathcal{H}$-hybrid interfaces $\mathcal{S}$, all ideal functionalities with restricted IO $\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle$, $\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle$ and $\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle$ and all $(\mathcal{G}_1,\mathcal{G}_2)$-hybrid interfaces $\mathcal{T}$ it holds for all $c \notin$ ``$\mathtt{H} \rightsquigarrow \mathcal{G}_1$'' $\cup$ ``$\mathtt{C} \rightsquigarrow \mathcal{G}_1$'' that*

$$\Pr[\mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]} = c] \leq \Pr[\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} = c] \ .$$

In particular, if $\Pr[\mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]} \in$ ``$\mathtt{H} \rightsquigarrow \mathcal{G}_1$'' $\cup$ ``$\mathtt{C} \rightsquigarrow \mathcal{G}_1$''$] \overset{c}{\approx} 0$, *then*

$$\mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]} \overset{c}{\approx} \mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} \ .$$

Proof. By definition we have that $\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} = \mathrm{IDEAL}^{\mathcal{H},\mathcal{G}_2}_{\mathcal{F},\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}}$ and that $\mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]} = \mathrm{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]}$ as long as no IO restriction is violated. By Lemma 3.2 on page 84 we therefore have that $\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} = \mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]}$ as long as no IO restriction is violated. We then consider what happens when an IO restriction is violated. In the processes four different IO restrictions are present: $\mathcal{IO}(\mathcal{H}),\mathcal{IO}(\mathcal{G}_1),\mathcal{IO}(\mathcal{G}_2)$ and $\mathcal{IO}(\mathcal{F})$. Since the processes are identical until one of them is violated, the first to be violated is the same in both. We consider for each what happens if it is violated first. If $\mathcal{IO}(\mathcal{F})$ is first, then the output of both processes will be in ``$\mathtt{H} \rightsquigarrow \mathcal{F}$'' or ``$\mathtt{C} \rightsquigarrow \mathcal{F}$'' and will be the same in both as $\mathcal{IO}(\mathcal{F})$ is checked by $\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle$ in the same way in both processes. If $\mathcal{IO}(\mathcal{G}_2)$ is first, then the output of both processes will be in ``$\mathtt{H} \rightsquigarrow \mathcal{G}_2$'' or ``$\mathtt{C} \rightsquigarrow \mathcal{G}_2$'' and will be the same in both as $\mathcal{IO}(\mathcal{G}_2)$ is checked in the same way in both processes by the rules of Fig. 3.20 on page 98. If $\mathcal{IO}(\mathcal{H})$ is the first, then in $\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}}$ the output will be in ``$\mathtt{H} \rightsquigarrow \mathcal{H}$'' or ``$\mathtt{C} \rightsquigarrow \mathcal{H}$''. Since $\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$ is checking $\mathcal{IO}(\mathcal{H})$ exactly as is done in $\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}}$ it follows that the outputs of the processes are the same. Notice in particular that the points at which the checks of $\mathcal{IO}(\mathcal{H})$ is placed in $\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$ was chosen such that if both $\mathcal{IO}(\mathcal{H})$ and $\mathcal{IO}(\mathcal{G}_2)$ are violated in the same activation, then $\mathcal{IO}(\mathcal{H})$ is checked by $\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$ before it outputs to $\mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\mathcal{H}]}$, where $\mathcal{IO}(\mathcal{G}_2)$ is checked. This is consistent with the order in which the checks are done in $\langle[\mathcal{H},\mathcal{G}_2],[\mathcal{IO}_{\mathcal{H}},\mathcal{IO}_{\mathcal{G}_2}]\rangle$ in $\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}}$. This proves that the processes are different only when $\mathcal{IO}(\mathcal{G}_1)$ is violated, which established the claim of the lemma. □

3.5.10 Composing an Environment with a Protocol

In this section we redefine the notion of composing an environment with a protocol. Basically we add to the previous definition that $\mathcal{Z}[\pi,\mathcal{G}_2]$ checks the IO restriction of $\mathcal{G}_2$ and terminates the execution if it is violated. We define $\mathcal{Z}[\pi,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle]$ to be the construction $\mathcal{Z}[\pi,\mathcal{G}_2']$ from Fig. 3.15 on page 88, with $\mathcal{G}_2' = \langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle$ and with the change that if $\mathcal{G}_2'$ outputs $w \in$ ``$\mathtt{H} \rightsquigarrow \mathcal{G}_2$'' $\cup$ ``$\mathtt{C} \rightsquigarrow \mathcal{G}_2$'' in party activation Step 3, corrupt Step 3 or end round Step 3, then $\mathcal{Z}[\pi,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle]$ terminates the execution with output w. We allowed $\mathcal{Z}$ to output from $\{0,1\} \cup$ ``$\mathtt{H} \rightsquigarrow \mathcal{G}_2$'' $\cup$ ``$\mathtt{C} \rightsquigarrow \mathcal{G}_2$''. Later we deal with this using Lemma 2.3 on page 12.

Environment $\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$

initialize:

$\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$ receives $(k,z,r_{\mathcal{Z}})$ and picks a section of $r_{\mathcal{Z}}$ as random bits r for $\mathcal{S}$. Let $r'_{\mathcal{Z}}$ be the remaining random bits and input $(k,z,r'_{\mathcal{Z}})$ to $\mathcal{Z}$.

environment:

When asked to produce a command, run $\mathcal{Z}$ to get a command c. If $c=(\texttt{guess},b)$, then output c. Otherwise proceed as described below.

party activation:

If $c=(\texttt{activate},i,x_{i,r},\{m_{i,j,r-1}\}_{i\in C})$ then:

1. For $i\in C$ compute $(m^{\pi}_{i,j,r-1},m^{\gamma}_{i,j,r-1})\leftarrow m_{i,j,r-1}$ and output $(\texttt{activate},i,x_{i,r},\{m^{\pi}_{i,j,r-1}\}_{i\in C})$.
2. *Then in* $\text{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]}$ *the party P_i is activated and the values $\{m^{\pi}_{i,j,r-1}\}_{i\in C}$ and $v_{\mathcal{F}}$ are input to $\mathcal{T}$ which generates values $\{m^{\gamma}_{i,j,r}\}_{j\in[n]\setminus\{i\}}$ and $(v_{\mathcal{G}_1},v_{\mathcal{G}_2})$, which are given to $\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$.*
3. Input $\{m^{\gamma}_{i,j,r-1}\}_{i\in C}$ and $v_{\mathcal{G}_1}$ to $\mathcal{S}$ which generates values $\{m^{\gamma}_{i,j,r}\}_{j\in[n]\setminus\{i\}}$ and $v_{\mathcal{H}}$.
4. **Input $(i,v_{\mathcal{H}})$ to $\mathcal{IO}(\mathcal{H})$ and receive a value w. If $w\in$ ``$\texttt{H}\rightsquigarrow\mathcal{H}$'', then terminate with output w.**
5. Then input $\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}$ and $(v_{\mathcal{H}},v_{\mathcal{G}_2})$ to $\mathcal{Z}$, where $m_{i,j,r}=(m^{\pi}_{i,j,r},m^{\gamma}_{i,j,r})$.

corrupt:

If $c=(\texttt{corrupt},i)$, then output $(\texttt{corrupt},i)$ and:

1. *In* $\text{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]}$ *the value $(\texttt{corrupt},iv_{\mathcal{F}})$ is input to to $\mathcal{T}$ to generate $(r^{\pi}_i,(v_{\mathcal{G}_1},v_{\mathcal{G}_2}))$, which is given to $\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$.*
2. Input $(\texttt{corrupt},i,v_{\mathcal{G}_1})$ to $\mathcal{S}$ to generate $(r^{\gamma}_i,v_{\mathcal{H}})$.
3. **Input $(\texttt{corrupt},i,v_{\mathcal{H}})$ to $\mathcal{IO}(\mathcal{H})$ and receive a value w. If $w\in$ ``$\texttt{H}\rightsquigarrow\mathcal{H}$'', then terminate with output w.**
4. Input $(r_i=(r^{\pi}_i,r^{\gamma}_i),(v_{\mathcal{H}},v_{\mathcal{G}_2}))$ to $\mathcal{Z}$.

end round:

If $c=(\texttt{end round},(v_{\mathcal{H}},v_{\mathcal{G}_2})$, then:

1. **Input $(\texttt{activate},v_{\mathcal{H}})$ to $\mathcal{IO}(\mathcal{H})$ and receive a value w. If $w\in$ ``$\texttt{C}\rightsquigarrow\mathcal{H}$'', then terminate with output w.**
2. Activate $\mathcal{S}$ on input $(\texttt{end round},v_{\mathcal{H}})$ and receive as output a value $v_{\mathcal{G}_1}$.
3. Output $(\texttt{end round},(v_{\mathcal{G}_1},v_{\mathcal{G}_2})$.
4. *In* $\text{IDEAL}^{\mathcal{G}_1,\mathcal{G}_2}_{\mathcal{F},\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]}$ *$\mathcal{T}$ is activated on input $(\texttt{end round},(v_{\mathcal{G}_1},v_{\mathcal{G}_2}))$ and produces a value $v_{\mathcal{F}}$. Then $(\texttt{activate},v_{\mathcal{F}})$ is input to $\mathcal{F}$ and values $\{y^{\mathcal{F}}_{i,r}\}_{i\in[n]}$ are produced. Then $\{y^{\mathcal{F}}_{i,r}\}_{i\in C}$ are input to $\mathcal{T}$ which produces output $\{(t^{\mathcal{G}_1}_{i,r},t^{\mathcal{G}_2}_{i,r})\}_{i\in C}$, which are given to $\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$ along with $\{y^{\mathcal{F}}_{i,r}\}_{i\in H}$.*
5. Input $\{t^{\mathcal{G}_1}_{i,r}\}_{i\in C}$ to $\mathcal{S}$, get $\{t^{\mathcal{H}}_{i,r}\}_{i\in C}$ and input $(\{y^{\mathcal{F}}_{i,r}\}_{i\in H},\{(t^{\mathcal{H}}_{i,r},t^{\mathcal{G}_2}_{i,r})\}_{i\in C})$ to $\mathcal{Z}$.

Figure 3.21: The $(\mathcal{G}_1,\mathcal{G}_2)$-hybrid interface $\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$ obtained by composing a $(\mathcal{H},\mathcal{G}_2)$-hybrid environment $\mathcal{Z}$ with a $\mathcal{H}$-hybrid interface $\mathcal{S}$.

Lemma 3.7 *For all $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid protocols π, all $\mathcal{H}$-hybrid protocols γ and all $(\mathcal{H}, \mathcal{G}_2)$-hybrid environments $\mathcal{Z}$, it holds that*

$$\mathrm{HYB}^{\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle}_{\pi[\gamma/\mathcal{G}_1], \mathcal{Z}} = \mathrm{HYB}^{\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle}_{\gamma, \mathcal{Z}[\pi, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle]} \ .$$

For all $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid protocols π, all ideal functionalities $(\mathcal{G}_1, \mathcal{G}_2)$, all $(\mathcal{H}, \mathcal{G}_2)$-hybrid environments $\mathcal{Z}$ and all $\mathcal{H}$-hybrid interfaces $\mathcal{S}$ it holds that

$$\mathrm{HYB}^{\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle}_{\pi, \mathcal{Z}[\mathcal{S}/\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle]} = \mathrm{IDEAL}^{\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle}_{\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle, \mathcal{S}, \mathcal{Z}[\pi, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle]} \ .$$

Proof. The proof is an extension of Lemma 3.3 on page 86, proceeding as in the proof of Lemma 3.6 on page 101. The crucial observation for the first claim is that $\mathcal{Z}[\pi, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle]$ checks $\mathcal{IO}(\mathcal{G}_2)$ exactly as it is done in $\mathrm{HYB}^{\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle}_{\pi[\gamma/\mathcal{G}_1], \mathcal{Z}}$. For the second claim observe that $\mathcal{Z}[\pi, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle]$ checks $\mathcal{IO}(\mathcal{G}_2)$ exactly as it is done in $\mathrm{HYB}^{\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle}_{\pi, \mathcal{Z}[\mathcal{S}/\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle]}$ and observe that $\mathcal{Z}[\mathcal{S}/\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle]$ checks $\mathcal{IO}(\mathcal{H})$ exactly as it is done in $\mathrm{IDEAL}^{\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle}_{\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle, \mathcal{S}, \mathcal{Z}[\pi, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle]}$. Furthermore $\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle$ is present in both executions and is checked in the same way in both. □

Lemma 3.8 *Let $\mathcal{H}$, $\mathcal{G}_1$ and $\mathcal{G}_2$ be ideal functionalities and assume that $\mathcal{IO}(\mathcal{H})$ and $\mathcal{IO}(\mathcal{G}_2)$ are PPT. Let π be a $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid protocol and let γ by a $\mathcal{H}$-hybrid protocol.*

- *Assume that for all $\mathcal{H}$-hybrid environments $\mathcal{Z}$ it holds that* if

  $$\Pr[\mathrm{IDEAL}^{\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle}_{\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle, \mathcal{S}, \mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{G}_1\text{''}] \overset{c}{\approx} 0 \ ,$$

  then

  $$\Pr[\mathrm{IDEAL}^{\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle}_{\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle, \mathcal{S}, \mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{H}\text{''}] \overset{c}{\approx} 0$$

  and

  $$\mathrm{IDEAL}^{\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle}_{\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle, \mathcal{S}, \mathcal{Z}} \overset{c}{\approx} \mathrm{HYB}^{\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle}_{\gamma, \mathcal{Z}} \ ;$$

- *And that for all $(\mathcal{G}_1, \mathcal{G}_2)$-hybrid environments $\mathcal{Z}$ it holds that* if

  $$\Pr[\mathrm{IDEAL}^{\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle \mathcal{F}, \mathcal{IO}(\mathcal{F})\rangle, \mathcal{T}, \mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{F}\text{''}] \overset{c}{\approx} 0 \ ,$$

  then

  $$\Pr[\mathrm{IDEAL}^{\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle \mathcal{F}, \mathcal{IO}(\mathcal{F})\rangle, \mathcal{T}, \mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{G}_1\text{''} \cup \text{``}H \rightsquigarrow \mathcal{G}_2\text{''}] \overset{c}{\approx} 0$$

  and

  $$\mathrm{IDEAL}^{\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle \mathcal{F}, \mathcal{IO}(\mathcal{F})\rangle, \mathcal{T}, \mathcal{Z}} \overset{c}{\approx} \mathrm{HYB}^{\langle \mathcal{G}_1, \mathcal{IO}(\mathcal{G}_1)\rangle, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle}_{\pi, \mathcal{Z}} \ ;$$

- *Then for all $(\mathcal{H}, \mathcal{G}_2)$-hybrid environments $\mathcal{Z}$ it holds that* if

  $$\Pr[\mathrm{IDEAL}^{\langle \mathcal{H}, \mathcal{IO}(\mathcal{H})\rangle, \langle \mathcal{G}_2, \mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle \mathcal{F}, \mathcal{IO}(\mathcal{F})\rangle, \mathcal{T}[\mathcal{S}/\mathcal{G}_1], \mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{F}\text{''}] \overset{c}{\approx} 0 \ ,$$

  then

$$\Pr[\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{H}\text{''} \cup \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_2\text{''}] \overset{c}{\approx} 0$$

and

$$\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} \overset{c}{\approx} \mathrm{HYB}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\pi[\gamma/\mathcal{G}_1],\mathcal{Z}} .$$

Proof. Let $\mathcal{Z}$ be any $(\mathcal{H},\mathcal{G}_2)$-hybrid environments and assume that

$$\Pr[\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}\text{''}] \overset{c}{\approx} 0 . \tag{3.1}$$

By Eq. 3.1 and Lemma 3.6 on page 101 we have that

$$\Pr[\mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}\text{''}] \overset{c}{\approx} 0 . \tag{3.2}$$

Since $\mathcal{IO}(\mathcal{H})$ is assumed to be PPT we have that $\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$ is PPT. So, using Eq. 3.2 and the second premise we get that the following holds:

$$\Pr[\mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_1\text{''} \cup \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_2\text{''}] \overset{c}{\approx} 0 \tag{3.3}$$

$$\mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]} \overset{c}{\approx} \mathrm{HYB}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\pi,\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]} . \tag{3.4}$$

Notice that to apply the second premise we need $\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$ to be an environment. Since $\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$ outputs other values than 0 and 1, formally it is not an environment. However, $\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]$ only has a finite domain D as the sets of symbols for reporting IO violations are assumed to be finite. Therefore we can use Lemma 2.3 on page 12 and apply the second premise for each of the environments $I^{(d)}(\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle])$, which applies $I^{(d)}$ to its final output value.

By Lemma 3.7 on the facing page and Eq. 3.4 we have that

$$\mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]} \overset{c}{\approx} \mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle}_{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\mathcal{S},\mathcal{Z}[\pi,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle]} . \tag{3.5}$$

By Eq. 3.3 and Eq. 3.5 we have that

$$\Pr[\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle}_{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\mathcal{S},\mathcal{Z}[\pi,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle]} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_1\text{''}] \overset{c}{\approx} 0 . \tag{3.6}$$

Since π and $\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle$ are assumed to be PPT, we have that $\mathcal{Z}[\pi,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle]$ is PPT, which by Eq. 3.6 and the first premise gives us that

$$\Pr[\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle}_{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\mathcal{S},\mathcal{Z}[\pi,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle]} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{H}\text{''}] \overset{c}{\approx} 0 \tag{3.7}$$

$$\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle}_{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\mathcal{S},\mathcal{Z}[\pi,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle]} \overset{c}{\approx} \mathrm{HYB}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle}_{\gamma,\mathcal{Z}[\pi,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle]} . \tag{3.8}$$

Again we applied Lemma 2.3 on page 12. Lemma 3.7 on the facing page and Eq. 3.8 imply that

$$\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle}_{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\mathcal{S},\mathcal{Z}[\pi,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle]} \overset{c}{\approx} \mathrm{HYB}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\pi[\gamma/\mathcal{G}_1],\mathcal{Z}} . \tag{3.9}$$

By Eq. 3.5 and Eq. 3.9 we have that

$$\mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]} \overset{c}{\approx} \mathrm{HYB}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\pi[\gamma/\mathcal{G}_1],\mathcal{Z}} . \tag{3.10}$$

Clearly $\Pr[\mathrm{HYB}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\pi[\gamma/\mathcal{G}_1],\mathcal{Z}} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_1\text{''} \cup \text{``}\mathtt{C} \rightsquigarrow \mathcal{G}_1\text{''}] = 0$, which by Eq. 3.10 on the preceding page gives us that

$$\Pr[\mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_1\text{''} \cup \text{``}\mathtt{C} \rightsquigarrow \mathcal{G}_1\text{''}] \overset{\mathrm{c}}{\approx} 0 \ ,$$

which by Lemma 3.6 on page 101 gives us that

$$\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} \overset{\mathrm{c}}{\approx} \mathrm{IDEAL}^{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T},\mathcal{Z}[\mathcal{S}/\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle]} \ . \tag{3.11}$$

Eq. 3.3 on the preceding page and Eq. 3.11 gives us that

$$\Pr[\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}_2\text{''}] \overset{\mathrm{c}}{\approx} 0 \ . \tag{3.12}$$

By Eq. 3.5 on the preceding page and Eq. 3.11 we have that

$$\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} \overset{\mathrm{c}}{\approx} \mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle}_{\langle\mathcal{G}_1,\mathcal{IO}(\mathcal{G}_1)\rangle,\mathcal{S},\mathcal{Z}[\pi,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle]} \ ,$$

which by Eq. 3.7 on the preceding page implies that

$$\Pr[\mathrm{IDEAL}^{\langle\mathcal{H},\mathcal{IO}(\mathcal{H})\rangle,\langle\mathcal{G}_2,\mathcal{IO}(\mathcal{G}_2)\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{F})\rangle,\mathcal{T}[\mathcal{S}/\mathcal{G}_1],\mathcal{Z}} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{H}\text{''}] \overset{\mathrm{c}}{\approx} 0 \ . \tag{3.13}$$

Eq.s. 3.5, 3.9, 3.11, 3.12 and 3.13 establish the lemma. □

3.5.11 Modeling Classes of Adversaries via Restricted IO

We have until now only considered adversaries which could adaptively corrupt all parties. In particular, we did not model static adversaries or adversaries that can only corrupt a bounded number of parties. We now use our notion of restricted IO to do this. Since an IO restriction is allowed to depend on the sequence of corruptions, we can use the IO restriction to define the corruption patterns under which the protocol is realized. We can define the static IO restriction $\mathcal{IO}_{\mathrm{static}}$ for an ideal functionality $\mathcal{F}$. If $\mathcal{IO}_{\mathrm{static}}$ receives an input of the form $(\mathtt{corrupt}, \cdot)$ (a corruption) after having received the first input of the form $(\mathtt{activate}, j, v)$ (a party activation), then $\mathcal{IO}_{\mathrm{static}}$ outputs ``$\mathtt{H} \rightsquigarrow \mathcal{F}$''. We also define the t-threshold corruption restriction $\mathcal{IO}_{\mathrm{thresh}}(t)$, to restrict the number of corruptions to a given threshold t of the parties. If $\mathcal{IO}_{\mathrm{thresh}}(t)$ receives an input of the form $(\mathtt{corrupt}, \cdot)$ more than t times, then it outputs ``$\mathtt{H} \rightsquigarrow \mathcal{F}$''. Consider now a protocol realizing $\langle\mathcal{F}, \mathcal{IO}_{\mathrm{static}}\rangle$ (respectively $\langle\mathcal{F}, \mathcal{IO}_{\mathrm{thresh}}(t)\rangle$) in the real-life model. This means that there exists an interface $\mathcal{S}$ such that for all environments $\mathcal{Z}$ it holds that *if* $\Pr[\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}_{\mathrm{static}}\rangle,\mathcal{S},\mathcal{Z}} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}\text{''}] = 0$ (respectively $\Pr[\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}_{\mathrm{thresh}}\rangle,\mathcal{S},\mathcal{Z}} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}\text{''}] = 0$), *then* $\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}_{\mathrm{static}}\rangle,\mathcal{S},\mathcal{Z}} \overset{\mathrm{c}}{\approx} \mathrm{REAL}_{\pi,\mathcal{Z}}$ (respectively $\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}_{\mathrm{thresh}}(t)\rangle,\mathcal{S},\mathcal{Z}} \overset{\mathrm{c}}{\approx} \mathrm{REAL}_{\pi,\mathcal{Z}}$). This means that we do not require that $\mathrm{IDEAL}_{\langle\mathcal{F},\mathcal{IO}\rangle,\mathcal{S},\mathcal{Z}} \overset{\mathrm{c}}{\approx} \mathrm{REAL}_{\pi,\mathcal{Z}}$ if $\mathcal{Z}$ corrupts a party during the execution (respectively corrupts more than t parties), except with negligible probability. This is exactly what we wanted to capture.

A subtlety arises in the hybrid models. We describe the subtly using $\mathcal{IO}_{\mathrm{static}}$ as an example. Consider the ideal process $\mathrm{IDEAL}^{\langle\mathcal{G},\mathcal{IO}(\mathcal{IO}_{\mathrm{static}})\rangle}_{\langle\mathcal{F},\mathcal{IO}(\mathcal{IO}_{\mathrm{static}})\rangle,\mathcal{T},\mathcal{Z}}$. If $\mathcal{Z}$ ever corrupts a party

during the execution, then both the IO restriction of $\mathcal{G}$ and the IO restriction of $\mathcal{F}$ are violated by a corruption. Therefore they will output ``$\mathtt{H} \rightsquigarrow \mathcal{G}$'' respectively ``$\mathtt{H} \rightsquigarrow \mathcal{F}$''. Which one is taken to be the output of the execution is essential. What we want is that if $\mathcal{IO}_{\text{static}}$ is violated by $\mathcal{Z}$, then $\text{IDEAL}^{\langle \mathcal{G}, \mathcal{IO}(\mathcal{IO}_{\text{static}})\rangle}_{\langle \mathcal{F}, \mathcal{IO}(\mathcal{IO}_{\text{static}})\rangle, \mathcal{T}, \mathcal{Z}}$ outputs ``$\mathtt{H} \rightsquigarrow \mathcal{F}$'', so that we do not require from the simulator that $\text{HYB}^{\langle \mathcal{G}, \mathcal{IO}(\mathcal{IO}_{\text{static}})\rangle}_{\pi, \mathcal{Z}} \overset{c}{\approx} \text{IDEAL}^{\langle \mathcal{G}, \mathcal{IO}(\mathcal{IO}_{\text{static}})\rangle}_{\langle \mathcal{F}, \mathcal{IO}(\mathcal{IO}_{\text{static}})\rangle, \mathcal{T}, \mathcal{Z}}$ when $\mathcal{IO}_{\text{static}}$ is violated. By inspection of corrupt in Fig. 3.20 on page 98 it can be seen that this is indeed the case as $\mathcal{F}$ is activated before $\mathcal{G}$.

In the following we will be using $\mathcal{IO}_{\text{thresh}}(t)$ so often that it is convenient to introduce the following terminology.

Definition 3.19 *Let $\langle \mathcal{F}, \mathcal{IO}(\mathcal{F})\rangle$ and $\langle \mathcal{G}, \mathcal{IO}(\mathcal{G})\rangle$ be ideal functionalities with IO restriction and let π be a hybrid protocol. We say that π t-realizes $\langle \mathcal{F}, \mathcal{IO}(\mathcal{F})\rangle$ in the $\langle \mathcal{G}, \mathcal{IO}(\mathcal{G})\rangle$-hybrid model if π realizes $\langle \mathcal{F}, \mathcal{IO}(\mathcal{F}) \wedge \mathcal{IO}_{\text{thresh}}(t)\rangle$ in the $\langle \mathcal{G}, \mathcal{IO}(\mathcal{G})\rangle$-hybrid model.*

3.5.12 The Significance of Restricted IO

We want to emphasis that restricted IO is primarily significant when efficiency is an issue; It is not known to change the picture when it comes to established possibility results for which tasks are realizable.

Restricted IO allows us to specified that a given functionality is only expected to work for a given restricted set of inputs, which in some cases allows to realize the functionality more efficiently. Restricted IO is however not known to allow us to realize any functionality that could not be realized without assuming the restricted IO restriction. As an example, in Section 3.8.3 we show how to specify a notion of universally composable proof of membership, which does not at the same time require realizations to be universally composable proofs of knowledge. This is significant from a practical point of view as universally composable proofs of membership find many applications and, with our current knowledge, can be realized much more efficiently than universally composable proofs of knowledge. However, as demonstrated by Canetti and Fischlin [CF01] universally composable proofs of knowledge can be realized for all NP relations. Therefore the notion of universally composable proofs of membership does not allow us to construct universally composable zero-knowledge proof systems for a larger class of languages.

It seems this could easily be the general picture, that restricted IO does not allow us to realize more ideal functionalities. Notice that for technical reasons the claim is actually false. Since the restricted IO behavior allows us to specify functionalities which are not PPT, the restricted IO behavior trivially allows us to realize a larger class of ideal functionalities since the prior formalization did not even allow to *specify* these functionalities. To be able to compare the models we therefore only consider functionalities which could be specified in both the model with restricted IO and the model without restricted IO. Therefore we only consider PPT IO restrictions $\mathcal{IO}$. To be able to compare the models we then consider realizing $\mathcal{F}_{|\mathcal{IO}}$ in the model without IO restriction vs. realizing $\langle \mathcal{F}, \mathcal{IO}\rangle$ in the model with IO restriction.

To be precise, for any ideal functionality $\mathcal{F}$ and PPT IO restriction $\mathcal{IO}$, let $\Pi(\mathcal{F}_{|\mathcal{IO}})$ denote the set of protocols π which realize $\mathcal{F}_{|\mathcal{IO}}$ in the model without IO restriction and let

$\Pi(\langle \mathcal{F}, \mathcal{IO} \rangle)$ denote the set of protocols π which realize $\mathcal{F}$ in the model with IO restriction. We clear have that $\Pi(\mathcal{F}_{|\mathcal{IO}}) \subseteq \Pi(\langle \mathcal{F}, \mathcal{IO} \rangle)$. The question is whether there exist $\mathcal{F}$ and PPT $\mathcal{IO}$ such that $\Pi(\mathcal{F}_{|\mathcal{IO}}) = \emptyset$ and $\Pi(\langle \mathcal{F}, \mathcal{IO} \rangle) \neq \emptyset$.

Notice that by Theorem 3.4 on page 93 at the level of individual protocols there is a difference in that more protocols might realize $\langle \mathcal{F}, \mathcal{IO} \rangle$. Theorem 3.4 gives us $\mathcal{F}$ for which $\Pi(\mathcal{F}_{|\mathcal{IO}}) \subsetneq \Pi(\langle \mathcal{F}, \mathcal{IO} \rangle)$. Indeed the existence of $\mathcal{F}$ and $\mathcal{IO}$ for which $\Pi(\langle \mathcal{F}, \mathcal{IO} \rangle) \setminus \Pi(\mathcal{F}_{|\mathcal{IO}}) \neq \emptyset$ is closely related to our motivation for introducing the notion of restricted IO: The set $\Pi(\langle \mathcal{F}, \mathcal{IO} \rangle) \setminus \Pi(\mathcal{F}_{|\mathcal{IO}})$ might contain superiorly efficient protocols. However for the example provided by Theorem 3.4 it is not the case that $\Pi(\mathcal{F}_{|\mathcal{IO}}) = \emptyset$, and it seems that it could be the case for all $\mathcal{F}$ that if $\Pi(\langle \mathcal{F}, \mathcal{IO} \rangle) \neq \emptyset$, then $\Pi(\mathcal{F}_{|\mathcal{IO}}) \neq \emptyset$.

The author does not know of any ideal functionality $\mathcal{F}$ and PPT IO restriction $\mathcal{IO}$ where $\Pi(\mathcal{F}_{|\mathcal{IO}}) = \emptyset$ and $\Pi(\langle \mathcal{F}, \mathcal{IO} \rangle) \neq \emptyset$. Even though this problem is not directly associated to the main motivation for introducing restricted IO, from a theoretical point of view it might be interesting to investigate whether such a system exists.

The answer to the problem depends on the model of computation considered, and we consider some models here. We note that the below treatment only covers some models of computation and is rather informal. The goal is to justify why restricted IO might not change the picture when it comes to possibility results, not to give a formal treatment. The primary reason being that the model of restricted IO was introduced exactly for the purpose of efficiency and modularity of proof, not to change the classes of realizable tasks. So, we do not consider the problem a fundamental problem to the issue of restricted IO.

3.5.12.1 The Model without Guaranteed Message Delivery

As demonstrated by Canetti, Lindell, Ostrovsky and Sahai [CLOS02] *any* PPT ideal functionality can be realized in the asynchronous model without guaranteed message delivery.[1] This makes the answer simple. Since all functionalities can already be realized, certainly no more functionalities can be realized under restricted IO.

3.5.12.2 The Model with Guaranteed Message Delivery

We can then add message delivery to the model. If we guarantee that messages are delivered in the real-life model and require from a realization that it too terminates and delivers its results, then no longer can all functionalities be realized. As discussed in Chapter 8, in this model there exist functionalities which can only be realized if less than a third of the parties are honest. This opens the possibility that there exist functionalities which cannot be realized with a dishonest third, but which can be realized with dishonest third under a restricted IO behavior. This means that the notion of restricted IO behavior might be interesting from the point of view of completeness results.

As an example we can consider the Byzantine agreement problem. This problem is discussed in great detail in Chapter 7, so here we only give an informal discussion. Consider the functionality which receives from each party P_i a bit b_i and outputs the majority of the

[1] In this model, in the real-life execution the messages are not guaranteed to be delivered, but on the other hand, in the ideal process, the functionality is not required to deliver messages either.

values. If the majority value is not defined, the functionality lets the adversary specify a bit c and outputs c to all parties. This functionality guarantees two important properties. First of all, all honest parties always output the same value and, second, if the honest parties are in majority and agree on the input, then the output is the common value of their inputs. As discussed in Chapter 7, this functionality can only be realized if less than a third of the parties is corrupted.[2] It can however be realized with more than a corrupted third under a specific IO behavior. Let n denote the number of parties and let t denote the number of corrupted parties. Without any restriction on the IO behavior we need that $t < n/3$. If we consider the restricted IO behavior that at least h of the honest parties always input a common value, where $h > n/2$, then the functionality can be realized as long as $t < n/2$. To see this consider the following protocol: Each party P_i sends its bit b_i to all parties. Then all parties wait until there exists a value $b \in \{0, 1\}$ for which it has received $b = b_j$ from at least $\lfloor n/2 \rfloor + 1$ parties P_j, including itself. Then it outputs b. Notice that since $2(\lfloor n/2 \rfloor + 1) > n$ the value b, and thus the protocol, is well-defined. Since, by the restricted IO behavior, at least $\lfloor n/2 \rfloor + 1$ honest parties have the same input value, and all honest parties send their input to all other parties, all parties will indeed receive at least $\lfloor n/2 \rfloor + 1$ identical values. Therefore the protocol terminates, by the guarantee that all sent messages are delivered.

It seems that this allows us to realize a functionality that could not be realized without restricted IO. This is however not the case, as we should compare to realizing the restricted functionality: We could have obtained the same result without considering restricted IO, by specifying a functionality which breaks down unless $h > n/2$ honest parties input the same value. We can even consider a slightly stronger formulation of such an ideal functionality: It works as the BA functionality, but it is only required to guarantee agreement when at least $\lfloor n/2 \rfloor + 1$ honest parties input the same value. I.e., consider the following weak BA functionality: It receives from each party P_i a bit b_i and, if at least $\lfloor n/2 \rfloor + 1$ of the honest parties input a common value b, then output b to all parties. Otherwise, let the adversary specify an output value c_j for each party P_j and output c_j to P_j. It is easy to see that the protocol described above also realizes this weak BA functionality.

3.6 Correctness of a Protocol

In this section we define a notion of correctness of a protocol in the UC model and we prove that for a particular class of functionalities correctness implies security. This will make some proofs simpler in the following sections.

Consider a hybrid protocol π and an ideal functionality $\mathcal{G}$. For any adversary $\mathcal{Z}$ the execution $\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}$ results in some output from the honest parties P_i in π. Informally we say that $\pi[\mathcal{G}]$ is a correct realization of $\mathcal{F}$ if $\mathcal{F}$ results in the same outputs from the honest parties when run in the same environment. Recall that when an ideal functionality $\mathcal{F}$ is run, besides receiving inputs for honest parties, it also expects an input $v_{\mathcal{F}}$ on the SIT. This value models inputs from the corrupted parties and non-deterministic choices of $\mathcal{F}$. To compare the execution of π to that of $\mathcal{F}$ we therefore say that for every possible execution of π, there

[2] Unless we assume a synchronous network and assume that the parties are able to sign messages. Here we do not assume either.

Algorithm $\mathrm{HYB}^{\mathcal{G},(V,\mathcal{F})}_{\pi,\mathcal{Z}}(k,r,z)$

initialize:
As in Fig. 3.4 on page 76, but with an initial input $r_{\mathcal{F}}$ and the initialization of $\mathcal{F}$ with $(k, r_{\mathcal{F}})$.

environment activation:
As environment activation in Fig. 3.4.

party activation:
On $(\texttt{activate}, i, x_{i,r}, \{m_{j,i,r-1}\}_{j\in C})$, first input (i, x_i) to $\mathcal{F}$. If $\mathcal{F}$ outputs $w \in$ ``$\texttt{H} \rightsquigarrow \mathcal{F}$'', then terminate with output $w$. Then proceed as in party activation in Fig. 3.4. If $\mathcal{G}$ outputs $w \in$ ``$\texttt{H} \rightsquigarrow \mathcal{G}$'', then terminate with output w. Notice that the party activation in Fig. 3.4 defines an output value $y_{i,r}$ from P_i.

corrupt:
On $(\texttt{corrupt}, i)$, first input $(\texttt{corrupt}, i)$ to $\mathcal{F}$. If $\mathcal{F}$ outputs $w \in$ ``$\texttt{H} \rightsquigarrow \mathcal{F}$'', then terminate with output $w$. Then proceed as in corrupt in Fig. 3.4. If $\mathcal{G}$ outputs $w \in$ ``$\texttt{H} \rightsquigarrow \mathcal{G}$'', then terminate with output w.

end round:
On $(\texttt{end round}, v_{\mathcal{G}})$, first proceed as in in end round in Fig. 3.4. If $\mathcal{G}$ outputs $w \in$ ``$\texttt{C} \rightsquigarrow \mathcal{G}$'', then terminate with output w. Then input the state of $\mathrm{HYB}^{\mathcal{G},(V,\mathcal{F})}_{\pi,\mathcal{Z}}(k,r,z)$, except for the state of $\mathcal{F}$ and $\mathcal{Z}$, to V to generate a value $v_{\mathcal{F}}$. Then input $(\texttt{end round}, v_{\mathcal{F}})$ to $\mathcal{F}$. If $\mathcal{F}$ outputs $w \in$ ``$\texttt{C} \rightsquigarrow \mathcal{F}$'', then terminate with output w. Otherwise, (when $\mathcal{F}$ outputs $\{y^{\mathcal{F}}_{i,r}\}_{i\in[n]}$) let $\{y_{i,r}\}_{i\in H}$ be the values defined in party activation. If $y_{i,r} \neq y^{\mathcal{F}}_{i,r}$ for any $i \in H$, then terminate the execution with output `incorrect`.

Figure 3.22: The $\mathcal{G}$-hybrid model with an ideal functionality for comparison.

exists efficiently computable inputs $v_{\mathcal{F}}$ to $\mathcal{F}$ such that π and $\mathcal{F}$ have the same outputs for all honest parties. Formally we require that there exists a PPT algorithm V which given a state of π and $\mathcal{G}$ right after an `end round` command outputs a value $v_{\mathcal{F}}$ with the property that it makes $\mathcal{F}$ output the same as π did in the previous round. In more detail, we consider the execution $\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}$ and extend the execution to include a copy of $\mathcal{F}$, which is activated on the same input sequence as π. At each `end round` command in $\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}$ we then apply V to the state of π and $\mathcal{G}$ to compute a $v_{\mathcal{F}}$. Then $v_{\mathcal{F}}$ is input to $\mathcal{F}$. We then require that the outputs generated by $\mathcal{F}$ are identical to those of π. We denote this execution by $\mathrm{HYB}^{\mathcal{G},(V,\mathcal{F})}_{\pi,\mathcal{Z}}(k,r,z)$. The details are given in Fig. 3.22.

Definition 3.20 *We say that π correctly realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if there exists a PPT algorithm V such that for all environment $\mathcal{Z}$ it holds that* if

$$\Pr[\mathrm{HYB}^{\mathcal{G},(V,\mathcal{F})}_{\pi,\mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{F}\text{''}] \overset{\mathrm{c}}{\approx} 0 \ ,$$

then

$$\Pr[\mathrm{HYB}^{\mathcal{G},(V,\mathcal{F})}_{\pi,\mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{G}\text{''} \cup \text{``}C \rightsquigarrow \mathcal{F}\text{''} \cup \{\mathit{incorrect}\}] \overset{\mathrm{c}}{\approx} 0 \ .$$

According to Definition 3.20, π correctly realizing $\mathcal{F}$ means that in any environment which uses π according to the IO restriction of $\mathcal{F}$ it holds that π uses $\mathcal{G}$ according to the IO

restriction of $\mathcal{G}$ and that the outputs of π are identical to those of $\mathcal{F}$, except with negligible probability. The condition that $\text{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})}$ does not output a element from ``$\mathtt{C} \rightsquigarrow \mathcal{F}$'' is a restriction on $V$, that it uses $\mathcal{F}$ according to its IO restriction. Notice that if $\text{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})}$ outputs an element from ``$\mathtt{C} \rightsquigarrow \mathcal{G}$'' because $\mathcal{Z}$ violates the IO restriction of $\mathcal{G}$, then in particular $\text{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})}$ does not output `incorrect`. So, if $\mathcal{Z}$ violates the IO restriction of $\mathcal{G}$ there are no requirements on π or V.

In Definition 3.15 on page 99 we defined a notion of realizing an ideal functionality. To distinguish this notion from that of correct realization we sometimes say that π securely realizes $\mathcal{F}$ when π realizes $\mathcal{F}$ according to Definition 3.15.

3.6.1 Arguing Secure Realization via Correct Realization

Notice that correctly realizing an ideal functionality is by far enough to securely realize it according to Definition 3.15 on page 99. The reason for this is that the algorithm V, in computing $v_{\mathcal{F}}$, has access to the entire state of π, which in particular includes all the inputs of the honest parties. This is more information than an interface is normally given. If however $\mathcal{F}$ has the property that it leaks all inputs on the SOT, then we can prove that a correct realization of $\mathcal{F}$ is also a secure realization.

Definition 3.21 *We say that an ideal functionality* outputs all inputs on the SOT. *If on each input of the form (i, x) the ideal functionality writes x on the SOT.*

Theorem 3.6 *Let $\mathcal{F}$ and $\mathcal{G}$ be ideal functionalities with IO restriction and let π be a hybrid protocol. If π correctly realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model and $\mathcal{F}$ outputs all inputs on the SOT, then π securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model.*

Proof. Since π correctly realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model there exists a PPT algorithm V such that for all hybrid environments $\mathcal{Z}$ we have that if

$$\Pr[\text{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}\text{''}] \overset{\text{c}}{\approx} 0 \ , \tag{3.14}$$

then

$$\Pr[\text{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{G}\text{''} \cup \text{``}\mathtt{C} \rightsquigarrow \mathcal{F}\text{''} \cup \{\mathtt{incorrect}\}] \overset{\text{c}}{\approx} 0 \ . \tag{3.15}$$

Consider the interface $\mathcal{S}_{\pi,\mathcal{G},V}$ in Fig. 3.23 on the following page and consider the ideal process $\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G}}$. We define an alternative version $\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G},\mathtt{incorrect}}$ which is defined exactly as $\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G}}$, except that if after an `end round` command the $y_{i,r}$ values defined in party activation in $\mathcal{S}_{\pi,\mathcal{G},V}$ differ from the $y_{i,r}$ values output by $\mathcal{F}$ in $\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G}}$, then the output of $\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G},\mathtt{incorrect}}$ is `incorrect`. It is straight-forward to verify that

$$\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G},\mathtt{incorrect}} = \text{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})} \ . \tag{3.16}$$

This follows from the observation that all values are passed identically between the entities $\mathcal{G}$, $\mathcal{F}$, π, V and $\mathcal{G}$ in both and that all values are checked according to the same IO restrictions. The only difference between $\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G},\mathtt{incorrect}}$ and $\text{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})}$ is that in $\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G},\mathtt{incorrect}}$,

Interface $\mathcal{S}_{\pi,\mathcal{G},V}$

The interface runs internally a copy of π and $\mathcal{G}$ on the inputs of $\mathcal{Z}$. This is possible because all inputs from $\mathcal{Z}$ to $\mathcal{F}$ in $\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}$ are revealed to $\mathcal{S}_{\pi,\mathcal{G},V}$ on the SOT of $\mathcal{F}$. In detail the simulation proceeds as follows:

initialize:

On input $(k, r_{\mathcal{S}})$ use a section of $r_{\mathcal{S}}$ as the random bits $r_1, \ldots, r_n \in \{0,1\}^*$ used by the parties and use a section as the random bits $r_{\mathcal{G}}$ for $\mathcal{G}$. For $i, j \in [n]$, let $m_{i,j,0} = \epsilon$, let $m_{i,i,0} = (k, r_i)$ and let $t_{i,-1} = \epsilon$. The interface then runs a copy of $\text{HYB}^{\mathcal{G},(V,\mathcal{F})}_{\pi,\mathcal{Z}}$, where $\mathcal{F}$ and $\mathcal{Z}$ are the ideal functionality respectively the environment in $\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}$ and $\mathcal{G}$ is the core functionality of the ideal functionality with IO restriction $\langle \mathcal{G}, \mathcal{IO}(\mathcal{G}) \rangle$. This is done as described below.

party activation:

On $(\texttt{activate}, i, v_{\mathcal{F}}, \{m_{j,i,r-1}\}_{j\in C})$, let $x_{i,r} = v_{\mathcal{F}}$. Values $\{m_{j,i,r-1}\}_{j\in H}$ and $t_{i,r-1}$ were defined in the previous round. Add these to $\{m_{j,i,r-1}\}_{j\in C}$ and compute $(\{m_{i,j,r}\}_{j\in[n]}, y_{i,r}, s_{i,r}) = P_i(\{m_{j,i,r-1}\}_{j\in[n]}, x_{i,r}, t_{i,r-1})$. Input $(i, s_{i,r})$ to $\mathcal{G}$ and receive $v_{\mathcal{G}}$ on the SOT of $\mathcal{G}$. Output $(\{m_{i,j,r}\}_{j\in[n]\setminus\{i\}}, v_{\mathcal{G}})$ to $\mathcal{Z}$.

corrupt:

On $(\texttt{corrupt}, i, v_{\mathcal{F}})$, input $(\texttt{corrupt}, i)$ on the SOT of $\mathcal{G}$ and receive a value $v_{\mathcal{G}}$ on the SOT of $\mathcal{G}$. Then output $(r_i, v_{\mathcal{G}})$. Set $C = C \cup \{i\}$.

end round:

On $(\texttt{end round}, v_{\mathcal{G}})$, input $(\texttt{end round}, v_{\mathcal{G}})$ to $\mathcal{G}$ to generate $\{t_{i,r}\}_{i\in[n]}$.

Then input the state of $\text{HYB}^{\mathcal{G},(V,\mathcal{F})}_{\pi,\mathcal{Z}}$, except for the state of $\mathcal{F}$ and $\mathcal{Z}$ (which is not accessible by $\mathcal{G}$), to V to receive a value $v_{\mathcal{F}}$. Then output $v_{\mathcal{F}}$ and receive the input $\{y_{i,r}\}_{i\in C}$. Then output $\{t_{i,r}\}_{i\in C}$. The values $\{t_{i,r}\}_{i\in H}$ are used as input for the honest parties in the next round. Set $r = r + 1$.

Figure 3.23: The interface $\mathcal{S}_{\pi,\mathcal{G},V}$ used in the proof of Theorem 3.6 on the page before.

the values $y_{i,r}$ given to $\mathcal{Z}$ comes from $\mathcal{F}$ and that in $\text{HYB}^{\mathcal{G},(V,\mathcal{F})}_{\pi,\mathcal{Z}}$, the values $y_{i,r}$ given to $\mathcal{Z}$ comes from π. If however these values ever differ, then the output of both experiments is `incorrect`. Consider then any environment $\mathcal{Z}$ where

$$\Pr[\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}} \in \text{``H} \rightsquigarrow \mathcal{F}\text{''}] \overset{c}{\approx} 0 \ . \tag{3.17}$$

We prove the lemma by proving that

$$\Pr[\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}} \in \text{``H} \rightsquigarrow \mathcal{G}\text{''}] \overset{c}{\approx} 0 \ . \tag{3.18}$$

and

$$\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}} \overset{c}{\approx} \text{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}} \ . \tag{3.19}$$

Since $\text{IDEAL}^{\mathcal{G},\texttt{incorrect}}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}$ and $\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}$ are identical until $\text{IDEAL}^{\mathcal{G},\texttt{incorrect}}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}$ outputs `incorrect` we have that

$$\Pr[\text{IDEAL}^{\mathcal{G},\texttt{incorrect}}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}} \in \text{``H} \rightsquigarrow \mathcal{F}\text{''}] \leq \Pr[\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}} \in \text{``H} \rightsquigarrow \mathcal{F}\text{''}] \ . \tag{3.20}$$

By Eq.s 3.16, 3.17 and 3.20 we get Eq. 3.14 on page 111 and can therefore conclude Eq. 3.15 on page 111. By Eq. 3.15 we have that $\Pr[\mathrm{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})} = \texttt{incorrect}] \overset{c}{\approx} 0$, which by Eq. 3.16 implies that

$$\Pr[\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G},\texttt{incorrect}} = \texttt{incorrect}] \overset{c}{\approx} 0 \ . \tag{3.21}$$

Since $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G},\texttt{incorrect}}$ and $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G}}$ are identical until $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G},\texttt{incorrect}}$ outputs `incorrect`, Eq.s 3.16 and 3.21 imply that

$$\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}_{\pi,\mathcal{G},V},\mathcal{Z}}^{\mathcal{G}} \overset{c}{\approx} \mathrm{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})} \ . \tag{3.22}$$

By Eq.s 3.14 and 3.15 we have that

$$\Pr[\mathrm{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})} \in \text{``}\texttt{H} \rightsquigarrow \mathcal{F}\text{''} \cup \text{``}\texttt{C} \rightsquigarrow \mathcal{F}\text{''} \cup \{\texttt{incorrect}\}] \overset{c}{\approx} 0 \ .$$

Since the experiments $\mathrm{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})}$ and $\mathrm{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G}}$ are identical until $\mathrm{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})}$ outputs a value from $\text{``}\texttt{H} \rightsquigarrow \mathcal{F}\text{''} \cup \text{``}\texttt{C} \rightsquigarrow \mathcal{F}\text{''} \cup \{\texttt{incorrect}\}$ this implies that

$$\mathrm{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G},(V,\mathcal{F})} \overset{c}{\approx} \mathrm{HYB}_{\pi,\mathcal{Z}}^{\mathcal{G}} \ . \tag{3.23}$$

By Eq.s 3.15 and 3.22 we get Eq. 3.18 on the preceding page, and by Eq.s 3.22 and 3.23 we get Eq. 3.19 on the facing page. □

3.7 Rewinding

We now describe a mechanism for capturing a weaker form of security, where the simulator is allowed to rewind the execution. Rewinding is a technique which has been used to prove many protocols secure, in particular many zero-knowledge protocols. The main tool which is always used is to end the simulation if it gets stuck and then try again from an earlier state. If the simulation has a probability $1/p$ of succeeding in one run, this approach will take an expected p trials. Therefore such simulations will at best be *expected* PPT, which we keep in mind when we make our definition. It is well-known that proving protocols secure using rewinding is problematic in many ways, in particular such protocols in general have poor compositional properties. Running l simulations in parallel, each with a probability of successful simulation of $\frac{1}{2}$ say, will give a protocol with a probability of success which can be as low as 2^{-l}, which soon renders the rewinding technique useless. Protocols proved secure with rewinding do compose in sequence, even though one should even be careful in that case, see Goldreich and Krawczyk [GK90].

Our notion of 'UC security with rewinding' does not circumvent these well-known problems. Indeed we cannot reprove the general composition theorem for the new notion. We will however see that protocols which are only 'UC secure with rewinding' can be used as sub-protocols to build protocols which are fully UC secure. The value of the new notion therefore comes from the fact that it allows us to use 'UC secure with rewinding' protocols as sub-protocols in UC secure protocols in a modular manner.

In Chapter 5 we present a zero-knowledge proof system which we only know how to prove secure using rewinding. We will however use the zero-knowledge proof system in every

subsequent chapter to build universally composable protocols. Used carefully rewinding proofs can play as prominent a role in the UC framework as in other, weaker frameworks. This is our main motivation for defining rewinding security within the UC framework.

Consider the usual ideal process $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$. We consider an extension which we denote by $\text{IDEAL}^{\curvearrowleft}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and call the simulation with rewinding. It runs exactly as $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, except that we allow $\mathcal{S}$ to use a special rerun control tape to rewind the execution. The simulator can output two types of commands, $(\texttt{store}, cid)$ and $(\texttt{rerun-from}, cid)$. In response to a $(\texttt{store}, cid)$ the current configuration of $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ is stored under the configuration id cid; We do not store the state of $\mathcal{S}$ though as $\mathcal{S}$ is allowed to maintain state between the reruns. The stored configuration includes the state of IO restrictions if such are present. In response to a $(\texttt{rerun-from}, cid)$ command, $\mathcal{S}$ is run in the configuration stored under cid. If no such configuration is stored, the command is ignored. If $\mathcal{Z}$ outputs $(\texttt{guess}, b)$, then $\mathcal{S}$ is given a special input indicating that $\mathcal{Z}$ terminated, in response to which it must either rerun from a stored configuration or let the game end. In the later case $\text{IDEAL}^{\curvearrowleft}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ terminates with output b.[3] If an IO restriction is ever violated, then $\text{IDEAL}^{\curvearrowleft}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ immediately terminates with the appropriate output — in particular $\mathcal{S}$ is not given the possibility of more reruns. We require that $\mathcal{S}$ does an *expected* polynomial number of reruns.

Definition 3.22 *Let $\mathcal{S}$ be a hybrid interface, let π be a protocol for the $\mathcal{G}$-hybrid model and let $\mathcal{F}$ be an ideal functionality. We say that $\mathcal{S}$ can produce $\pi[\mathcal{G}]$ with rewinding given $\mathcal{F}$, written $\overset{\curvearrowleft}{\mathcal{S}} : \mathcal{F} \triangleleft \pi[\mathcal{G}]$, if for all $\mathcal{G}$-hybrid environments $\mathcal{Z}$ where $\Pr[\text{IDEAL}^{\curvearrowleft,\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{F}\text{''}]$ it holds that $\Pr[\text{IDEAL}^{\curvearrowleft,\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{G}\text{''}] \overset{c}{\approx} 0$ and $\text{IDEAL}^{\curvearrowleft,\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \text{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}$.*

Definition 3.23 *Let π be a protocol for the $\mathcal{G}$-hybrid model and let $\mathcal{F}$ be an ideal functionality. We say that π realizes $\mathcal{F}$ with rewinding in the $\mathcal{G}$-hybrid model if there exists an expected PPT $\mathcal{G}$-hybrid interface $\mathcal{S}$ such that $\overset{\curvearrowleft}{\mathcal{S}} : \mathcal{F} \triangleleft \pi[\mathcal{G}]$.*

3.8 Some Functionalities

In this section we describe some conventions used in the context of the UC model, and we specify some ideal functionalities.

3.8.1 Conventions

If in the description of an ideal functionality we write "let the adversary specify an output round" (to specify when to output a value v), we mean the following: As part of the value output on the SOT, output $(\texttt{output round}, oid)$ for a unique output round id $oid \in \{0,1\}^*$. If then in a later round a value $(oid, \texttt{term})$ is input on the SIT, then output v. This allows us to specify functionalities without guaranteed termination.

Following [CLOS02] we say that an n-party functionality is well formed if it outputs its entire internal state when n parties are corrupted. In [CLOS02] it is also required that the functionality do not depend on which parties are corrupted to call it well-formed. Here we

[3] Of course $\mathcal{S}$ does not see b when it makes its decision.

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

The functionality runs with parties $P_1, \ldots, P_n$ and is parametrized by a round complexity r for a transmission. The functionality proceeds as follows:

secure message transmission:

On input (mid, P_j, m) from P_i, where $m \in \{0,1\}^*$, output $(mid, P_j, |m|)$ on the SOT.

After r rounds, output (mid, P_i, m) to P_j.

If P_i is corrupted and $(mid, P_i, P_j, \texttt{fail})$ is input on the SIT then drop all messages from P_i to P_j with message id mid.

If P_i is corrupted and (mid, P_i, P_i, m) is input on the SIT, then output (mid, i, m) to P_j.[a]

incorrect inputs:

If an honest party ever uses the same message id mid twice, then output `fail` on the SOT and break down.

[a] Notice that the environment has immediate delivery. This models a rushing environment.

Figure 3.24: The secure message transmission functionality $\mathcal{F}_{\text{SMT}}$.

drop this requirement as we will need to consider functionalities depending on the set of corrupted parties.

3.8.2 The Secure-Channels Model

In Fig. 3.24 a functionality for secure message transmission is given. It allows the party to transmit a message while only leaking the length of the message. In Chapter 4 we discuss how to realize $\mathcal{F}_{\text{SMT}}$ under various assumptions.

3.8.3 Zero-Knowledge Proofs

In Fig. 3.25 on the next page an ideal functionality for zero-knowledge proof of knowledge is given. If two parties, the prover P_i and the verifier P_j, agree on an instance x, then the functionality allows P_i to prove to P_j that it knows w such that $(x, w) \in R$ for some relation R, without leaking the witness w. The functionality requires that when an instance x is input by an honest party, then x is from the instance language S from which instances are supposed to be drawn. This means that a realization only has to be secure for such inputs. Since it cannot necessarily be checked in PPT whether $x \in S$ we have to capture this condition via the IO behavior. This is done by letting the IO restriction output ``$\texttt{H} \rightsquigarrow \mathcal{F}_{\text{ZK-PK}}$'' if $x \notin S$, intuitively meaning that the honest parties are considered to have misused the functionality if an honest party ever inputs $x \notin S$. Notice that because the functionality outputs x on the SOT along with the identity of the party that input x it is indeed possible for the IO restriction to check the condition.[4]

In Fig. 3.26 on page 117 a functionality for zero-knowledge proof of membership is given, which does not at the same time require proof of knowledge. The difference between a zero-

[4] The IO restriction is not restricted computationally, but is only given access to the SOT and the SIT.

Functionality $\mathcal{F}^R_{\text{ZK-PK}}$

The functionality runs with n parties $P_1, \ldots, P_n$. It is parametrized by a round complexity r and by a PPT binary relation $R \subseteq S \times \{0,1\}^*$.[a] All input values are output on the SOT, except the witness w.

initialization:

If all honest parties input `init` in the same round, then in a round specified by the adversary, output `ready` to all parties. Until then ignore all inputs.

activation:

If in the same round P_j inputs (pid, P_i, x) and P_i is corrupted or P_i is honest and inputs (pid, P_j, x, w) for $(x, w) \in R$, then after r rounds output (pid, b) to P_j, where $b \in \{0,1\}$ is determined as follows: If P_i is corrupted in the output round, then $b = 1$ iff a value (pid, P_i, P_j, w') for which $(x, w') \in R^{\mathtt{xtr}}$ was input on the SIT before round r. If P_i is honest in the output round, then $b = 1$ (we already required that $(x, w) \in R$ when P_i is honest).

IO restriction:

The IO restriction $\mathcal{IO}_{\text{ZK}}$ is defined by requiring that for all x input by honest parties it holds that $x \in S$. If this is not the case, then output ``H $\rightsquigarrow \mathcal{F}_{\text{ZK-PK}}$''. Notice that this is a legal specification as the value (pid, x) is revealed on the SOT and so the restriction can be checked given access to just the ST.

[a] This also captures the case with several such relations $R_1, \ldots, R_l$, using the definition $((i, x), w) \in R \equiv (x, w) \in R_i$.

Figure 3.25: The zero-knowledge proof of knowledge functionality for PPT binary relation $R \subseteq S \times \{0,1\}^*$.

knowledge proof of membership and a zero-knowledge proof of knowledge is that the prover only proves that $x \in L(R)$, i.e. there exists $w \in \{0,1\}^*$ such that $(x, w) \in R$. The prover does not convince the verifier that he *knows* such a witness. We formalize the loosening that the prover does not need to know the witness by allowing the environment to simply tell whether to accept the proof from a corrupted party or not, restricted to specifying acceptance for $x \in L(R)$ of course, to guarantee soundness. The essential difference between $\mathcal{F}_{\text{ZK-PK}}$ and $\mathcal{F}_{\text{ZK-PM}}$ is that the corrupted parties (inputting via the SIT) does not have to provide a witness to the functionality to prove that $x \in L(R)$. We still give the witness to the honest parties though, as the witness is needed in the protocols realizing the ideal functionality to obtain efficient realizations, or actually to obtain realizations at all. As for the condition $x \in S$ discussed above, the condition $x \in L(R)$ cannot necessarily be checked in PPT.[5] Therefore the IO restriction of $\mathcal{F}_{\text{ZK-PM}}$ is used to check the condition. If the adversary[6] ever instructs $\mathcal{F}_{\text{ZK-PM}}$ to accept an instance $x \notin L(R)$, then $\mathcal{F}_{\text{ZK-PM}}$ outputs ``C $\rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$'', intuitively meaning that the corrupted parties misused the functionality. We note that the IO restriction of $\mathcal{F}_{\text{ZK-PM}}$ can indeed check $x \in L(R)$ as the IO restriction is not computationally bounded.

[5] If it could, then the language and the realization of $\mathcal{F}_{\text{ZK-PM}}$ would be trivial.

[6] By the adversary we mean the entity inputting on the SIT. I.e. the environment in the hybrid model execution and the simulator in the ideal process.

Functionality $\mathcal{F}^{R}_{\text{ZK-PM}}$

The functionality runs with n parties $P_1, \ldots, P_n$. It is parametrized by a round complexity r and by a PPT binary relation $R \subseteq S \times \{0,1\}^*$.[a] All input values are output on the SOT, except the witness w.

initialization:

If all honest parties input `init` in the same round, then in a round specified by the adversary, output `ready` to all parties. Until then ignore all inputs.

activation:

If in the same round P_j inputs (pid, P_i, x) and P_i is corrupted or P_i is honest and inputs (pid, P_j, x, w) for $(x, w) \in R$, then after r rounds output (pid, b) to P_j, where $b \in \{0,1\}$ is determined as follows: If P_i is corrupted in the output round, then $b = 1$ iff a value $(pid, P_i, P_j, 1)$ was input on the SIT before round r. If P_i is honest in the output round, then $b = 1$.

IO restriction:

The IO restriction $\mathcal{IO}_{\text{ZK-PM}}$ is defined by requiring that for all x input by honest parties it holds that $x \in S$. If this is not the case, then output ‘‘`H` $\rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$’’. Furthermore we require that when $(pid, P_i, P_j, 1)$ is input on the SIT and P_j input (pid, P_i, x) and P_j is honest, then $x \in L(R)$. If this is not the case, then output ‘‘`C` $\rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$’’. Notice that this is a legal specification as the value (pid, x) is revealed on the SOT and so the restriction can be checked given access to just the ST.

[a] See the footnote in Fig. 3.25 on the facing page.

Figure 3.26: The zero-knowledge proof of membership functionality for PPT binary relation $R \subseteq S \times \{0,1\}^*$.

Since the way that soundness is guaranteed might seem somewhat peculiar, we take a closer look at how the IO restriction guarantees soundness. Assume that P_j is honest and P_i is corrupted. Consider the situation where P_j inputs (pid, P_i, x) for $x \notin L(R)$ (possibly because P_i sent x to P_j in some outer protocol and claimed that $x \in L(R)$) and where at the same time the entity that inputs on the SIT inputs $(pid, P_i, P_j, 1)$ on the SIT to make $\mathcal{F}_{\text{ZK-PM}}$ accept the proof (i.e. to make $\mathcal{F}_{\text{ZK-PM}}$ output $(pid, 1)$ to P_j). This is perfectly allowable by the description of activation. However, in that case the IO behavior of $\mathcal{F}_{\text{ZK-PM}}$ will output ‘‘`C` $\rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$’’ and the execution will terminate. Thereby the entity that inputs on the SIT will be 'penalized' by the rules of the given execution, whether ideal or hybrid. This way of enforcing soundness might seem unnecessarily complicated. However, since the bit $x \overset{?}{\in} L(R)$ cannot (necessarily) be computed in PPT we cannot simply let this check be part of the code in activation. We know of no approach to defining UC proof of membership which does not also imply proof of knowledge which does not require that the functionality be able to perform some arbitrary (as opposed to PPT) computation to check $x \in L(R)$.

Remark 3.1 *The above definition of zero-knowledge proof of membership can be used to exemplify how the symbol ‘‘`C` $\rightsquigarrow \cdot$’’ is given the indented meaning. Recall that ‘‘`C` $\rightsquigarrow \cdot$’’ is intended to restrict the corrupted parties. Since the input from the corrupted parties to*

an ideal functionality comes from different entities in different contexts this means that the symbol plays dual roles, restricting the simulator in one context and helping the simulator in others. The reason being that in the ideal process the corrupted parties receive inputs from the simulator (or in other words, the simulator inputs on the SIT of $\mathcal{F}_{\text{ZK-PM}}$) and thus ``$C \rightsquigarrow \cdot$'' restricts the simulator — since the ideal process is compared to the real-life execution which never outputs ``$C \rightsquigarrow \cdot$'' the simulator is restricted to behave such that the ideal functionality never outputs ``$C \rightsquigarrow \cdot$''. But in the $\mathcal{F}_{\text{ZK-PM}}$-hybrid model, the corrupted parties receive inputs from the environment (or in other words, the environment inputs on the SIT of $\mathcal{F}_{\text{ZK-PM}}$) and so ``$C \rightsquigarrow \cdot$'' restricts the environment — whenever the ideal functionality outputs ``$C \rightsquigarrow \cdot$'' the experiment ends with output ``$C \rightsquigarrow \cdot$'' and the environment has no further chances of distinguishing.

Taking the definition of zero-knowledge proof of membership as an example, it is the fact that $\mathcal{F}_{\text{ZK-PM}}$ outputs ``$C \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$'' when a false statement is accepted which forces the simulator to only letting $\mathcal{F}_{\text{ZK-PM}}$ output 1 for true statements. If the simulator lets $\mathcal{F}_{\text{ZK-PM}}$ output 1 for a false statement, then $\mathcal{F}_{\text{ZK-PM}}$, and thus the ideal process, outputs ``$C \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$'', and since the ideal process is compared to the real-life execution which never outputs ``$C \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$'' the simulation fails. Since the simulator is restricted to letting $\mathcal{F}_{\text{ZK-PM}}$ output 1 for true statements it follows that it cannot simulate a protocol which outputs 1 for a false statement. This guarantees the soundness of the protocol. This shows how ``$C \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$'' restricts the simulator in the ideal process. On the other hand, in the $\mathcal{F}_{\text{ZK-PM}}$-hybrid model the symbol ``$C \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$'' restricts the environment to using the functionality correctly. If the environment lets $\mathcal{F}_{\text{ZK-PM}}$ accept a false statement, then $\mathcal{F}_{\text{ZK-PM}}$ outputs ``$C \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$'' and the execution is terminated and the output of the hybrid execution will be ``$C \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$''. Since the exact same thing happens in the ideal process to which the hybrid execution is to be compared, this means that the compared executions are identical whenever the environment lets $\mathcal{F}_{\text{ZK-PM}}$ accept a false statement. Thus the behavior of the simulator does not matter when the environment lets $\mathcal{F}_{\text{ZK-PM}}$ accept false statements. This shows how ``$C \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$'' helps the simulator when $\mathcal{F}_{\text{ZK-PM}}$ occurs in a hybrid model, and shows why the simulator in a $\mathcal{F}_{\text{ZK-PM}}$-hybrid model can simulate under the assumption that the $\mathcal{F}_{\text{ZK-PM}}$ functionality only accepts true statements.

The notion of universally composable zero-knowledge proof of membership (which does not at the same time requiring proof of knowledge) finds applications in many settings and can, with our current knowledge, be realized much more efficiently than universally composable zero-knowledge proofs of knowledge. The notion seems to be hard to define without the use of arbitrary restricted IO behavior (or some other mechanism for specifying ideal functionalities which are not PPT). We think that this in itself justifies the trouble of establishing the restricted IO notion for the UC model. We realize $\mathcal{F}_{\text{ZK-PM}}$ in Chapter 5 and use it several times in subsequent chapters. In Chapter 5 we also realize $\mathcal{F}_{\text{ZK-PK}}^{R}$ with rewinding.

3.8.4 The Broadcast Model

We define the broadcast model to be the $(\mathcal{F}_{\text{BA}}, \mathcal{F}_{\text{broadcast}})$-hybrid model, where $\mathcal{F}_{\text{BA}}$ is given in Fig. 3.17 on page 91 and $\mathcal{F}_{\text{broadcast}}$ is the broadcast functionality given in Fig. 3.27 on

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

The functionality runs with parties $P_1, \ldots, P_n$ and is parametrized by a round complexity r. All inputs to the functionality are output on the SOT.

broadcast:

If P_i inputs $(\texttt{broadcast}, baid, m)$ then after r rounds output $(\texttt{broadcast}, P_i, baid, m)$ to all parties. If P_i is corrupted and $(baid, \texttt{fail})$ is input on the SIT, then drop all messages with id $baid$. If P_i is corrupted and $(\texttt{broadcast}, P_i, baid, m)$ is input on the SIT, then output $(\texttt{broadcast}, P_i, baid, m)$ to all parties.[a]

incorrect inputs:

If an honest party uses the same $baid$ twice, then output `fail` on the SOT and break down.

[a] Notice that the environment has immediate delivery. This models a rushing environment.

Figure 3.27: The broadcast functionality.

this page. A broadcast simply lets the parties agree on what message a given broadcaster sent, even if the broadcaster maliciously tries to make the parties accept different message. Byzantine agreement and broadcast are important tools in secure multiparty computation as they allow the parties to act consistently.

In Chapter 7 we show how to implement the broadcast model. Of $\mathcal{F}_{\text{BA}}$ and $\mathcal{F}_{\text{broadcast}}$ it is enough to realize one, as there exists simple reductions between them. In Chapter 7 we therefore only look at realizing $\mathcal{F}_{\text{BA}}$. The reason for looking at $\mathcal{F}_{\text{BA}}$ instead of $\mathcal{F}_{\text{broadcast}}$ is that most efficient broadcast protocols also solve the BA problem, and though the functionalities are equivalent under PPT reduction, the reduction of $\mathcal{F}_{\text{BA}}$ to $\mathcal{F}_{\text{broadcast}}$ requires n applications of $\mathcal{F}_{\text{broadcast}}$, whereas the reduction in the other direction can be done with one application of $\mathcal{F}_{\text{BA}}$: Given a broadcast channel, BA can be implemented by all parties broadcasting their input and then taking majority over the received values — this of course require that a majority of the parties is honest. Given a BA functionality broadcast can be implemented by the broadcaster sending the input to all parties and then running a BA on the values received by the parties.

3.8.5 Signatures

In Fig. 3.28 on the following page an ideal functionality for signature schemes is given and in Fig. 3.29 on page 121 the natural realization based on a digital signature scheme is given in the broadcast model. We now prove that if the signature scheme is secure then $\pi_{\text{sig}}^{(gen,sig,ver)}$ is indeed a realization of $\mathcal{F}_{\text{sig}}$. A proof of the a similar theorem appears in [CR03], for a somewhat different ideal functionality.[7] The proof idea is the same.

[7] The main difference is that our formalization guarantees universal verifiability, i.e that if some party has accepted a signature, then all parties to which the signature is transfered will also accept the signature. Universal verifiability is essential in some contexts, e.g. when using digital signatures for realizing broadcast and Byzantine agreement, which we will do in Chapter 7. Another difference is that our functionality does not have an explicit command for verifying a signature; Instead we have a command for transferring signatures. This is so mainly because our UC model does not allow for so-called immediate functionalities. Therefore

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

The functionality runs with parties $P_1, \ldots, P_n$. All inputs are output on the SOT.

initialize:

If all honest parties input `init` in the same round, then proceed as follows:

1. Initialize a dictionary S mapping from messages m and indices $i \in \{1, \ldots, n\}$ into indices $\{1, \ldots, n\}$. Initially $S(m, i) = \emptyset$. *We think of $S(m, i)$ as the indices of those parties P_j from which P_i has a signature on m, either because P_j signed m and sent the signature to P_i or because some other party transfered such a signature to P_i.*

   Let C denote the set of corrupted parties. We have the convention that for all $i \in C$ and all $m \in \{0,1\}^*$ it holds that $S(m, i) = C \cup \bigcup_{l \in H} S(m, l)$. *So, the corrupted parties knows all generated signatures and have exchanged signatures on all messages under their own keys.*
2. In a round specified by the adversary, output `ready` to all parties. Until having output `ready`, ignore all inputs.

sign:

If P_i inputs $(\texttt{sign}, j, m)$, where $j \in [n]$, then let $S(m, j) = S(m, j) \cup \{i\}$ and output $(\texttt{transfered}, i, i, m)$ to P_j, unless P_i was honest when inputting $(\texttt{sign}, j, m)$ and was corrupted during the round. In that case, do nothing.

In the following we use 'input $(\texttt{sign}, m)$' as a shorthand for 'input $(\texttt{sign}, j, m)$ for all $j \in [n]$'.

transfer:

If P_i inputs $(\texttt{transfer}, l, j, m)$, where $l \in S(m, i)$, then let $S(m, j) = S(m, j) \cup \{l\}$ and output $(\texttt{transfered}, i, l, m)$ to P_j, unless P_i was honest when inputting $(\texttt{transfer}, l, j, m)$ and was corrupted during the round. In that case, do nothing.

incorrect inputs:

If in the first round where a honest party inputs a non-trivial value some honest party do not input `init`, then break down. Also break down if an honest party inputs `init` twice or inputs a value not of one of the above forms. On break down, output `fail` on the SOT.

Figure 3.28: The signature functionality.

Theorem 3.7 *Let (gen, sig, ver) be a digital signature scheme where the set of signable messages is $\{0,1\}^*$. If (gen, sig, ver) is existentially unforgeable under adaptive chosen-message attack, then $\pi_{\text{sig}}^{(gen,sig,ver)}$ realizes $\mathcal{F}_{\text{sig}}$ in the $\mathcal{F}_{\text{broadcast}}$-hybrid model.*

Proof. In the following we use π_{sig} to denote $\pi_{\text{sig}}^{(gen,sig,ver)}$. By Theorem 3.6 on page 111 it is sufficient to prove that π_{sig} correctly realizes $\mathcal{F}_{\text{sig}}$ in the $\mathcal{F}_{\text{broadcast}}$-hybrid model. So, we have to prove that there exists a PPT algorithm V such that for all environment $\mathcal{Z}$ it holds that if

$$\Pr[\text{HYB}_{\pi_{\text{sig}}, \mathcal{Z}}^{\mathcal{F}_{\text{broadcast}}, (V, \mathcal{F}_{\text{sig}})} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}_{\text{sig}}\text{''}] \overset{c}{\approx} 0 \ ,$$

a verification command would necessarily have to take at least one round, which would add an unnecessary (round) complexity to the functionality.

Protocol $\pi_{\text{sig}}^{gen,sig,ver}$

The protocol runs in the $\mathcal{F}_{\text{broadcast}}$-hybrid model with parties $P_1, \ldots, P_n$. The party P_i proceeds as follows:

initialize:

In the very first round (where k and r_i is provided), proceed as follows:

1. Generate $(v, s) \leftarrow gen(1)$ and input $(\texttt{broadcast}, v)$ to $\mathcal{F}_{\text{broadcast}}$.
2. In round r after having input `init`, where r is the round complexity of $\mathcal{F}_{\text{broadcast}}$, if $\mathcal{F}_{\text{broadcast}}$ outputs a value $(\texttt{broadcast}, P_j, v_j)$, then store this value. For P_j where no $(\texttt{broadcast}, P_j, v_j)$ was received, let $v_j = \epsilon$.

The `init` inputs to P_i is handled as follows:

- On an input `init` after $\mathcal{F}_{\text{broadcast}}$ has delivered an output, simply output `ready` and until having output `ready`, ignore all inputs but the first `init` input.
- On an input `init` before round $\mathcal{F}_{\text{broadcast}}$ has delivered an output, wait until the round where $\mathcal{F}_{\text{broadcast}}$ delivers an output and then proceed as if having received `init` in that round.

sign:

On input $(\texttt{sign}, j, m)$, compute $\sigma \leftarrow sig(s, m)$ and send $(\texttt{transfer}, i, m, \sigma)$ to P_j.

transfer:

1. On input $(\texttt{transfer}, l, j, m)$, if a value (l, m, σ) is stored, then send $(\texttt{transfer}, l, m, \sigma)$ to P_j.
2. On a message $(\texttt{transfer}, l, m', \sigma')$ from P_j, if $ver(v_l, m', \sigma') = 1$, then output $(\texttt{transfered}, j, l, m')$ and store (l, m', σ').

Figure 3.29: Realizing $\mathcal{F}_{\text{sig}}$ using a digital signature scheme (gen, sig, ver).

then

$$\Pr[\text{HYB}_{\pi_{\text{sig}},\mathcal{Z}}^{\mathcal{F}_{\text{broadcast}},(V,\mathcal{F}_{\text{sig}})} \in \text{``}\texttt{H} \rightsquigarrow \mathcal{F}_{\text{broadcast}}\text{''} \cup \text{``}\texttt{C} \rightsquigarrow \mathcal{F}_{\text{sig}}\text{''} \cup \{\texttt{incorrect}\}] \overset{\text{c}}{\approx} 0 \ .$$

It is straight forward to verify that π_{sig} uses $\mathcal{F}_{\text{broadcast}}$ correctly when π_{sig} is activated correctly. And, the corrupted parties cannot misuse $\mathcal{F}_{\text{sig}}$. It is therefore enough to construct V such that $\Pr[\text{HYB}_{\pi_{\text{sig}},\mathcal{Z}}^{\mathcal{F}_{\text{broadcast}},(V,\mathcal{F}_{\text{sig}})} = \texttt{incorrect}] \overset{\text{c}}{\approx} 0$. This is done below.

The algorithm V proceeds as follows: If in some round all honest parties receive input `init`, then V receives some output round id oid from $\mathcal{F}_{\text{sig}}$. When the broadcast of keys in π_{sig} ends, V inputs $(oid, \texttt{term})$ on the SIT of $\mathcal{F}_{\text{sig}}$ to make it output `ready` from all parties, as do the parties in the protocol π_{sig}. This is a perfect simulation of the initialization in the protocol. When $\mathcal{Z}$ delivers $(\texttt{transfer}, l, m', \sigma')$ from corrupted P_j to honest P_i, then proceed as follows: If $ver(v_l, \sigma', m') = 0$, then do nothing. If $ver(v_l, \sigma', m') = 1$, then P_i should output $(\texttt{transfered}, j, l, m')$ in the simulation, so V must make $\mathcal{F}_{\text{sig}}$ output $(\texttt{transfered}, j, l, m')$ too. If P_l is corrupted or P_l ever received the input $(\texttt{sign}, \cdot, m')$, then because P_j is corrupted and because of the convention $S(m', j) = C \cup \bigcup_{l \in H} S(m', l)$ we have that $l \in S(m, j)$.

Therefore V can input $(\texttt{transfer}, l, i, m')$ to $\mathcal{F}_{\text{sig}}$ on behalf of the corrupted P_j to make $\mathcal{F}_{\text{sig}}$ output $(\texttt{transfered}, j, l, m')$ to P_i, as desired. So, in that case the simulation is perfect. If P_l is honest and never received the input $(\texttt{sign}, \cdot, m')$, then give up the simulation and output, *"This is a forgery for v_l: (m', σ')"*. In this case the simulation fails.

It is straight-forward to verify that $\text{HYB}_{\pi_{\text{sig}}, \mathcal{Z}}^{\mathcal{F}_{\text{broadcast}}, (V, \mathcal{F}_{\text{sig}})} \neq \texttt{incorrect}$, unless V outputs *"This is a forgery for v_l: (m', σ')"* and terminates. It is therefore enough to prove that $p(k, z)$ is negligible, where $p(k, z)$ is the probability that V outputs *"This is a forgery for v_l: (m', σ')"* in $\text{HYB}_{\pi_{\text{sig}}, \mathcal{Z}}^{\mathcal{F}_{\text{broadcast}}, (V, \mathcal{F}_{\text{sig}})}(k, z)$.

Consider the following adversary A designed for running in $\text{FORGE}_{(gen, sig, ver), A}(k, z)$. It receives input (k, z, v). Then it starts running $\text{HYB}_{\pi_{\text{sig}}, \mathcal{Z}}^{\mathcal{F}_{\text{broadcast}}, (V, \mathcal{F}_{\text{sig}})}(k, z)$, except that it picks a uniformly random index $s \in [n]$ and sets $v_s = v$. Then it runs $\text{HYB}_{\pi_{\text{sig}}, \mathcal{Z}}^{\mathcal{F}_{\text{broadcast}}, (V, \mathcal{F}_{\text{sig}})}(k, z)$. Each time it would normally compute $\sigma \leftarrow sig(s_s, m)$, instead it outputs $(\texttt{sign}, m)$ to the forging game and receives $\sigma \leftarrow sig(s_s, m)$. If during the run of $\text{HYB}_{\pi_{\text{sig}}, \mathcal{Z}}^{\mathcal{F}_{\text{broadcast}}, (V, \mathcal{F}_{\text{sig}})}(k, z)$ it happens that V outputs *"This is a forgery for v_s: (m', σ')"*, then $ver(v_s, \sigma', m') = 1$ and P_s is honest and never received the input $(\texttt{sign}, \cdot, m')$, so V never output $(\texttt{sign}, m')$ to the forging game. Therefore, output $(\texttt{forgery}, m', \sigma')$ to make $\text{FORGE}_{(gen, sig, ver), A}(k, z)$ output 1. If P_s is ever corrupted, then A gives up the run of $\text{HYB}_{\pi_{\text{sig}}, \mathcal{Z}}^{\mathcal{F}_{\text{broadcast}}, (V, \mathcal{F}_{\text{sig}})}(k, z)$, as it does not know the internal state of P_s.

Now, $p(k, z)$ is the probability that C outputs *"This is a forgery for v_l: (m', σ')"* for some $i \in [n]$. Since P_l is honest when this happens and the execution of $\text{HYB}_{\pi_{\text{sig}}, \mathcal{Z}}^{\mathcal{F}_{\text{broadcast}}, (V, \mathcal{F}_{\text{sig}})}(k, z)$ is independent of s in the view of $\mathcal{Z}$ until P_s is corrupted and A picked $s \in [n]$ uniformly at random, it follows that the probability that *"This is a forgery for v_s: (m', σ')"* is output is at least $\frac{p(k,z)}{n}$. So, $\Pr[\text{FORGE}_{(gen, sig, ver), A}(k, z) = 1] \geq \frac{p(k,z)}{n}$. So, $\Pr[\text{HYB}_{\pi_{\text{sig}}, \mathcal{Z}}^{\mathcal{F}_{\text{broadcast}}, (V, \mathcal{F}_{\text{sig}})}(k, z) = \texttt{incorrect}] \leq n \Pr[\text{FORGE}_{(gen, sig, ver), A}(k, z) = 1]$. Since $\text{FORGE}_{(gen, sig, ver), A} \overset{\text{c}}{\approx} 0$ by the existential unforgeability under adaptive chosen-message attack of (gen, sig, ver), and n is polynomial in k, it follows from Lemma 2.1 on page 11 that $\Pr[\text{HYB}_{\pi_{\text{sig}}, \mathcal{Z}}^{\mathcal{F}_{\text{broadcast}}, (V, \mathcal{F}_{\text{sig}})} = \texttt{incorrect}] \overset{\text{c}}{\approx} 0$, as desired. □

In Chapter 7 we prove that for $t < (n-1)/2$ the $\mathcal{F}_{\text{broadcast}}$-functionality can be realized in the $\mathcal{F}_{\text{sig}}$-hybrid model, and we discuss a well-known result that $\mathcal{F}_{\text{broadcast}}$ cannot be t-realized in the real-life model unless $t \geq (n-1)/3$. This shows that the use of $\mathcal{F}_{\text{broadcast}}$ in π_{sig} is necessary when $t \geq n/3$. Notice that π_{sig} only uses one broadcast per party. Therefore realizing $\mathcal{F}_{\text{broadcast}}$ using π_{sig} is not an absurdity. In practice the initial broadcast could be realized using a public-key infrastructure (PKI) which allows the parties to retrieve the public-key of some other party, along with a transferable certificate. Basically the initial rounds until $\mathcal{F}_{\text{broadcast}}$ has output the broadcast values to all parties models some kind of setup of the network, and the actual computation does not begin until $\mathcal{F}_{\text{broadcast}}$ delivered outputs. After this point $\mathcal{F}_{\text{broadcast}}$ is never used again. For later use we give protocols with this property a name.

Definition 3.24 *A protocol with a setup functionality $\mathcal{I}$ for the $\mathcal{G}$-hybrid model is a protocol π for the $(\mathcal{G}, \mathcal{I})$-hybrid model, where π is of the following form: In the very first round of the*

protocol (where k and r_i is supplied to P_i), the party P_i inputs `init` *to $\mathcal{I}$. Then P_i waits until $\mathcal{I}$ has output a value and until this happens, P_i ignores all inputs, except for the first* `init` *input. If P_i receives an* `init` *input while waiting for $\mathcal{I}$, then, in the round where $\mathcal{I}$ delivers an output, P_i proceeds as if having received* `init` *in that round. After $\mathcal{I}$ has delivered the first output to P_i, P_i never inputs anything but the empty string to $\mathcal{I}$ and ignores all outputs from $\mathcal{I}$. We require from the ideal functionality $\mathcal{I}$ that it delivers the first output to all parties in the same round when activated in the same round by all honest parties. We call $\mathcal{I}$ the initializing functionality of π and we call the round where $\mathcal{I}$ delivers outputs to all parties the actual start of the protocol.*

3.8.6 Threshold Signatures

In Fig. 3.30 on the following page an ideal functionality for threshold signature is given. A threshold signature has the defining quality that the ability to sign is shared between the parties and that a signature on a message m is not generated unless at least c parties allowed to signed it. We say that c parties gave a signature share and we call c the construction threshold. The 2 in the name of $\mathcal{F}_{\text{2T-sig}}$ refers to the functionality being a so-called dual-threshold signature functionality, because it allows to meaningfully set the corruption threshold lower than $c-1$. We contrast this to the functionality $\mathcal{F}_{\text{T-sig}}$ which is defined exactly like $\mathcal{F}_{\text{2T-sig}}$, except that we have the convention that if *any* honest party input $(\texttt{sign}, j, m)$, then $T(m,i) = 1$ for all corrupted parties P_i. I.e. as soon as just one honest party allowed to sign m, all corrupted parties receive a signature on m. In particular, if there is a maximal set of t corrupted parties, then as soon as $t+1$ parties allowed to sign m, at least one honest party did so, and therefore all corrupted parties obtain a signature on m. It is therefore not meaningful to set the construction threshold to $c > t+1$. For $\mathcal{F}_{\text{T-sig}}$ we therefore always assume that $c = t+1$. The dual-threshold signature functionality is named so following [Sho00].

One trivial way of realizing the dual-threshold signature functionality is to use a standard signature scheme for each party and then let a threshold signature consist of c individual signatures. Seeing c signature on m, it follows that at least c parties must have signed m.

Theorem 3.8 *For all $0 < c \leq n$ the protocol $\pi^c_{\text{tsig-triv}}$ realizes $\mathcal{F}^c_{\text{2T-sig}}$ in the $\mathcal{F}_{\text{sig}}$-hybrid model with signing round complexity* 1*.*

Proof. By Theorem 3.6 on page 111 it is sufficient to prove that $\pi^c_{\text{tsig-triv}}$ correctly realizes $\mathcal{F}^c_{\text{2T-sig}}$ in the $\mathcal{F}_{\text{sig}}$-hybrid model. So, we have to prove that there exists a PPT algorithm V such that for all environment $\mathcal{Z}$ it holds that if

$$\Pr[\text{HYB}^{\mathcal{F}_{\text{sig}},(V,\mathcal{F}^c_{\text{2T-sig}})}_{\pi^c_{\text{tsig-triv}},\mathcal{Z}} \in \text{``}\texttt{H} \rightsquigarrow \mathcal{F}^c_{\text{2T-sig}}\text{''}] \overset{\text{c}}{\approx} 0 \ ,$$

then

$$\Pr[\text{HYB}^{\mathcal{F}_{\text{sig}},(V,\mathcal{F}^c_{\text{2T-sig}})}_{\pi^c_{\text{tsig-triv}},\mathcal{Z}} \in \text{``}\texttt{H} \rightsquigarrow \mathcal{F}_{\text{sig}}\text{''} \cup \text{``}\texttt{C} \rightsquigarrow \mathcal{F}^c_{\text{2T-sig}}\text{''} \cup \{\texttt{incorrect}\}] \overset{\text{c}}{\approx} 0 \ .$$

The algorithm V is defined as follows: In each activation, where $\mathcal{Z}$ inputs $v_{\mathcal{F}_{\text{sig}}} = \texttt{ready}$ to $\mathcal{F}_{\text{sig}}$, V outputs $v_{\mathcal{F}^c_{\text{2T-sig}}} = \texttt{ready}$. If $\mathcal{Z}$ inputs a $v_{\mathcal{F}_{\text{sig}}}$ value to $\mathcal{F}_{\text{sig}}$ with the input $(\texttt{sign}, j, m)$ to corrupted P_i, then V specifies the input $(\texttt{sign}, j, m)$ to corrupted P_i in its $v_{\mathcal{F}^c_{\text{2T-sig}}}$ value.

Functionality $\mathcal{F}_{\text{2T-sig}}^{gen}$

The functionality runs with parties $P_1, \ldots, P_n$. It is parameterized by a signing round complexity r and a construction threshold c. All inputs are output on the SOT.

initialize:

If all honest parties input `init` in the same round, then proceed as follows:

1. Initialize a dictionary SS mapping from messages m and indices $i \in \{1, \ldots, n\}$ into indices $\{1, \ldots, n\}$. Initially $SS(m, i) = \emptyset$. Furthermore, let T be a dictionary mapping from indices and messages into $\{0, 1\}$. Initially $T(m, i) = 0$. *We think of $SS(m, i)$ as the indices of those parties P_j from which P_i has a signature share on m and we think of $T(m, i)$ as indicating whether P_i has a signature on m.*

   We have the convention that for all $i \in [n]$ and all $m \in \{0, 1\}^*$, if $|SS(m, i)| \geq c$, then $T(m, i) = 1$.

   Let C denote the set of corrupted parties. We have the convention that for all $i \in C$ and all $m \in \{0, 1\}^*$ it holds that $SS(m, i) = C \cup \bigcup_{l \in H} SS(m, l)$.
2. In a round specified by the adversary, output `ready` to all parties. Until having output `ready`, ignore all inputs.

sign:

If P_i inputs $(\texttt{sign}, j, m)$, where $j \in [n]$, then let $SS(m, j) = SS(m, j) \cup \{i\}$, unless P_i was honest when inputting $(\texttt{sign}, j, m)$ and was corrupted within r rounds. In that case, do nothing.

If during this $T(m, j) = 1$ then output $(\texttt{signed}, m)$ to P_j after r rounds. The output round is specified by the adversary.

In the following we use 'input $(\texttt{sign}, m)$' as a shorthand for 'input $(\texttt{sign}, j, m)$ for all $j \in [n]$'.

transfer:

If P_i inputs $(\texttt{transfer}, j, m)$, where $T(m, i) = 1$, then let $T(m, j) = 1$ and output $(\texttt{transfered}, i, m)$ to P_j, unless P_i was honest when inputting $(\texttt{transfer}, j, m)$ and was corrupted during the round. In that case, do nothing.

incorrect inputs:

If in the first round were a honest party inputs a non-trivial value some honest party do not input `init`, then break down. Also break down if an honest party inputs `init` twice or inputs a value not of one of the above forms. On break down, output `fail` on the SOT.

Figure 3.30: The dual-threshold signature functionality.

If $\mathcal{Z}$ inputs a $v_{\mathcal{F}_{\text{sig}}}$ value to $\mathcal{F}_{\text{sig}}$ with the input $(\texttt{transfer}, l, j, m)$ to corrupted P_i for all $l \in L \subseteq S(m, i)$, where $|L| = c$, then V specifies the input $(\texttt{transfer}, j, m)$ to P_i in its $v_{\mathcal{F}_{\text{2T-sig}}^c}$ value. All other types of correctly formed inputs to corrupted parties in $\mathcal{F}_{\text{sig}}$ are ignored. If $\mathcal{Z}$ give an input which is not of the correct form to a corrupted party in $\mathcal{F}_{\text{sig}}$, then V will just output $v_{\mathcal{F}_{\text{2T-sig}}^c} = \epsilon$.

Clearly V is a PPT algorithm. We argue that V maintains the invariant in $\text{HYB}_{\pi_{\text{tsig-triv}}^c, \mathcal{Z}}^{\mathcal{F}_{\text{sig}}, (V, \mathcal{F}_{\text{2T-sig}}^c)}$

Protocol $\pi_{\text{tsig-triv}}$

The protocol runs with parties $P_1, \ldots, P_n$ in the $\mathcal{F}_{\text{sig}}$-hybrid model. The protocol is parameterized by a construction threshold c. Party P_i proceeds as follows:

initialize:

On input `init`, input `init` to $\mathcal{F}_{\text{sig}}$ and wait for the output `ready`. Then output `ready`. Until having output `ready`, ignore all inputs.

Let $S(m, i) = \emptyset$.

sign:

On input $(\texttt{sign}, j, m)$, input $(\texttt{sign}, j, m)$ to $\mathcal{F}_{\text{sig}}$.

On output $(\texttt{transfered}, j, j, m')$, let $S(m, i) = S(m, i) = \cup\{j\}$. If during this it happens that $|S(m', i)| \geq c$, then output $(\texttt{signed}, m)$.

transfer:

On input $(\texttt{transfer}, j, m)$, where $|S(m, i)| \geq c$, let L be some subset of $S(m, i)$ of size c and input $(\texttt{transfer}, l, j, m)$ to $\mathcal{F}_{\text{sig}}$ for $l \in L$.

If there ever arrives exactly c transfers from some P_l in the same round, then output $(\texttt{transfered}, l, m')$. If less than c or more than c transfers arrive, then ignore these.

Figure 3.31: Realizing $\mathcal{F}_{\text{2T-sig}}$ using a signature functionality $\mathcal{F}_{\text{sig}}$.

that as long as $\mathcal{Z}$ only activates honest parties P_i on inputs of the correct form and only specifies inputs on behalf of corrupted parties in $\mathcal{F}_{\text{sig}}$ of the correct form it holds that $|S(m, i)| \geq c$ iff $T(m, i) = 1$. But first we argue that this is enough to prove the theorem. It is straightforward to verify that as long as $\mathcal{Z}$ only activates honest parties P_i on inputs of the correct form and only specifies inputs on behalf of corrupted parties in $\mathcal{F}_{\text{sig}}$ of the correct form, it will be the case that the parties of $\pi^c_{\text{tsig-triv}}$ only gives correct inputs to $\mathcal{F}_{\text{sig}}$ and that V only gives correct inputs on behalf of corrupted parties to $\mathcal{F}^c_{\text{2T-sig}}$. Therefore $\mathcal{Z}$ is always the first to violate an IO restriction. In particular

$$\Pr[\text{HYB}^{\mathcal{F}_{\text{sig}},(V,\mathcal{F}^c_{\text{2T-sig}})}_{\pi^c_{\text{tsig-triv}},\mathcal{Z}} \in \text{``}\texttt{H} \rightsquigarrow \mathcal{F}_{\text{sig}}\text{''} \cup \text{``}\texttt{C} \rightsquigarrow \mathcal{F}^c_{\text{2T-sig}}\text{''}] = 0 \ .$$

It is therefore sufficient to prove that as long as $\mathcal{Z}$ only activates honest parties P_i on inputs of the correct form and only specifies inputs on behalf of corrupted parties in $\mathcal{F}_{\text{sig}}$ of the correct form it holds that

$$\Pr[\text{HYB}^{\mathcal{F}_{\text{sig}},(V,\mathcal{F}^c_{\text{2T-sig}})}_{\pi^c_{\text{tsig-triv}},\mathcal{Z}} = \texttt{incorrect}] = 0 \ .$$

We have to argue that honest parties in $\pi^c_{\text{tsig-triv}}$ and $\mathcal{F}^c_{\text{2T-sig}}$ have identical outputs. By the invariant we have that as long as $\mathcal{Z}$ only activates honest parties P_i on inputs of the correct form and only specifies inputs on behalf of corrupted parties in $\mathcal{F}_{\text{sig}}$ of the correct form it holds that $|S(m, i)| \geq c$ iff $T(m, i) = 1$. We argue that this means that $\pi^c_{\text{tsig-triv}}$ and $\mathcal{F}^c_{\text{2T-sig}}$ deliver identical outputs. There are two types of output, $(\texttt{signed}, m)$ and $(\texttt{transfered}, i, m)$. Assume first that honest P_i in $\pi^c_{\text{tsig-triv}}$ outputs $(\texttt{signed}, m)$ in round r. This means that in round r it happened for the first time that $|S(m, i)| \geq c$. In particular $|S(m, i)| < c$ in round $r - 1$. By the invariant this means that $T(m, i) = 0$ in round $r - 1$ and $T(m, i) = 1$ in round

r. By inspection of $\mathcal{F}_{\text{2T-sig}}^{c}$ this means that $\mathcal{F}_{\text{2T-sig}}^{c}$ output $(\texttt{signed}, m)$ in round r. Assume then that $\mathcal{F}_{\text{2T-sig}}^{c}$ output $(\texttt{signed}, m)$ in round r on behalf of honest P_i. By inspection of $\mathcal{F}_{\text{2T-sig}}^{c}$ this happens when $T(m,i) = 0$ in round $r-1$ and $T(m,i) = 1$ in round r. By the invariant this implies that $|S(m,i)| < c$ in round $r-1$ and $|S(m,i)| \geq c$ in round r. By inspection of $\pi_{\text{tsig-triv}}^{c}$, this implies that P_i output $(\texttt{signed}, m)$ in round r. Assume then that honest P_i in $\pi_{\text{tsig-triv}}^{c}$ outputs $(\texttt{transfered}, j, m)$ in round r. This means that P_j got the input $(\texttt{transfer}, i, m)$ and that $|S(m,j)| \geq c$. By the invariant this means that $T(m,j) = 1$. Since P_j gets the input $(\texttt{transfer}, j, m)$ in $\pi_{\text{tsig-triv}}^{c}$ iff P_j gets the input $(\texttt{transfer}, j, m)$ in $\mathcal{F}_{\text{2T-sig}}^{c}$ it follows by inspection of $\mathcal{F}_{\text{2T-sig}}^{c}$ that P_i output $(\texttt{transfered}, j, m)$ in $\mathcal{F}_{\text{2T-sig}}^{c}$, as desired. Assume then that honest P_i output $(\texttt{transfered}, j, m)$ in $\mathcal{F}_{\text{2T-sig}}^{c}$. By inspection of $\mathcal{F}_{\text{2T-sig}}^{c}$ this implies that P_j got the input $(\texttt{transfer}, i, m)$ and that $T(m,j) = 1$. By the invariant this means that $|S(m,j)| \geq c$. Since P_j gets the input $(\texttt{transfer}, j, m)$ in $\pi_{\text{tsig-triv}}^{c}$ iff P_j gets the input $(\texttt{transfer}, j, m)$ in $\mathcal{F}_{\text{2T-sig}}^{c}$ it follows by inspection of $\pi_{\text{tsig-triv}}^{c}$ that P_i will receive c transfers from P_j and will then output $(\texttt{transfered}, j, m)$, as desired.

We proceed to prove the invariant. To allow an easier argument we prove a slightly stronger invariant. We prove for $\text{HYB}_{\pi_{\text{tsig-triv}}^{c}, \mathcal{Z}}^{\mathcal{F}_{\text{sig}}, (V, \mathcal{F}_{\text{2T-sig}}^{c})}$ that as long as $\mathcal{Z}$ only activates honest parties P_i on inputs of the correct form and only specifies inputs on behalf of corrupted parties in $\mathcal{F}_{\text{sig}}$ of the correct form it holds that $|S(m,i)| \geq c$ iff $T(m,i) = 1$ and that if $|SS(m,i)| < c$, then $SS(m,i) = S(m,i)$. This is trivially the case in the beginning, where both dictionaries are uninitialized. If at some point all honest parties are activated on input $\texttt{init}$ by $\mathcal{Z}$, then $\mathcal{F}_{\text{2T-sig}}^{c}$ initializes SS. And, in $\pi_{\text{tsig-triv}}^{c}$ all honest parties input $\texttt{init}$ to $\mathcal{F}_{\text{sig}}$, which then initializes S. By inspection of $\mathcal{F}_{\text{2T-sig}}^{c}$ and $\mathcal{F}_{\text{sig}}$ it can be seen that SS and S are initialized identically.

We then consider what happens on the various commands from $\mathcal{Z}$. Consider first the command $(\texttt{corrupt}, i)$. By inspection of $\mathcal{F}_{\text{sig}}$ and $\mathcal{F}_{\text{2T-sig}}^{c}$ we get for all $m \in \{0,1\}^*$ that $SS(m,i) = C \cup \bigcup_{l \in H} SS(m,l)$ and $S(m,i) = C \cup \bigcup_{l \in H} S(m,l)$. Assume first that for $|SS(m,l)| < c$ for all $l \in H$. Then by the invariant $SS(m,l) = S(m,l)$ for all $l \in H$ and therefore $SS(m,i) = S(m,i)$, which clearly implies that the invariant is maintained.

Consider then the activation of a honest party P_i on $(\texttt{sign}, j, m)$. In $\pi_{\text{tsig-triv}}^{c}$, the party P_i inputs $(\texttt{sign}, j, m)$ to $\mathcal{F}_{\text{sig}}$, which makes the update $S'(m,j) = S(m,j) \cup \{i\}$. In $\mathcal{F}_{\text{2T-sig}}^{c}$ we also get the update $SS'(m,j) = SS(m,j) \cup \{i\}$ on input $(\texttt{sign}, j, m)$. Furthermore, in both functionalities, if P_i is corrupted before the $\texttt{end round}$ command, then the update is not made. We prove that this maintains the invariant. Clearly so if no updates are made. Assume then that $SS'(m,j) = SS(m,j) \cup \{i\}$ and $S'(m,j) = S(m,j) \cup \{i\}$. Assume that $T'(m,j) = 1$. If $T(m,j) = 1$, then $|SS(m,j)| \geq c$ and therefore $|SS'(m,k)| \geq c$, as desired. If $T(m,j) = 0$, then $T'(m,j)$ was set to 1 because $SS(m,j) = c-1$ and $SS'(m,j) = c$. Since $SS(m,j) = c-1 < c$ we have, by the invariant, that $S(m,k) = SS(m,j)$. Since $SS'(m,j) = SS(m,j) \cup \{i\}$ and $S'(m,j) = S(m,j) \cup \{i\}$ this means that $S'(m,j) = SS'(m,j)$. In particular, $|S'(m,j)| = c$, as desired. If $|SS'(m,j)| < c$, we also have that $|SS(m,j)| < c$, which by the invariant means that $SS(m,j) = S(m,j)$. Since $SS'(m,j) = SS(m,j) \cup \{i\}$ and $S'(m,j) = S(m,j) \cup \{i\}$ this means that $SS'(m,j) = S'(m,j)$, as desired. Consider then the command $(\texttt{sign}, j, m)$ to corrupted P_i in $\mathcal{F}_{\text{sig}}$. By definition of V we have that $(\texttt{sign}, j, m)$ is given to the corrupted P_i in $\mathcal{F}_{\text{2T-sig}}^{c}$. The analysis then proceeds as for $(\texttt{sign}, j, m)$ to a

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

The functionality runs with parties $P_1, \ldots, P_n$ and is parametrized by a PPT generator gen.

initialize:

On input `init` from all honest parties, run $crs \leftarrow gen(k)$ and output crs on the SOT, let the adversary specify an output round, and in the specified round, output crs to all parties.

Figure 3.32: The common reference string functionality with generator gen.

honest party.

Consider then the activation of a honest party on $(\texttt{transfer}, j, m)$. In $\pi^c_{\text{tsig-triv}}$, if $|S(m,i)| < c$ nothing happens. By the invariant $|S(m,i)| < c$ implies that $T(m,i) = 0$. This means that in $\mathcal{F}^c_{\text{2T-sig}}$ nothing happens either. If $|S(m,i)| \geq c$, then P_i transfers c signatures to P_j. In particular $S'(m,j) = S(m,j) \cup L$. By the invariant $|S(m,i)| \geq c$ implies that $T(m,i) = 1$. So, in $\mathcal{F}^c_{\text{2T-sig}}$ we get that $T'(m,j) = 1$. Since $|S'(m,j) = S(m,j) \cup L| \geq |L| = c$ and $T'(m,j) = 1$, clearly the invariant is maintained.

Consider then the commands of the form $(\texttt{transfer}, l, j, m)$ to corrupted P_i in $\mathcal{F}_{\text{sig}}$. Let L be the indices $l \in S(m,i)$ for which $\mathcal{Z}$ input a $(\texttt{transfer}, l, j, m)$ command to P_i. If $|L| = c$, then, by definition, V inputs $(\texttt{transfer}, j, m)$ to $\mathcal{F}^c_{\text{2T-sig}}$. The analysis of this case proceeds as for input $(\texttt{transfer}, j, m)$ to honest P_i. If $|L| \neq c$, then V inputs nothing to the corrupted P_i in $\mathcal{F}^c_{\text{2T-sig}}$. This is sound as P_j in $\pi^c_{\text{tsig-triv}}$ will receive d transfers from P_i, for $d \neq c$, and will therefore ignore these. □

Notice that if the guaranteed number of honest parties is less than the construction threshold, i.e. $n - t < c$, then even if all honest parties sign the same message m they are not guaranteed to get a signature on m. Therefore $n \geq 2t+1$ is needed to guarantee signing. Furthermore, if $n \geq 2t+1$ and all honest parties sign m in round R, then all honest parties will receive a signature no later than in round $R+r$, where r is the signing round complexity of $\mathcal{F}_{\text{2T-sig}}$.

We have seen that we can realize the dual-threshold signature functionality given the signature functionality. The threshold signature functionality is in some respect weaker than what the realization using signatures give us. Given a threshold signature on m one knows that c parties signed m, but given c signatures from individual parties on m, one in addition knows *which* c parties signed m. This overkill in the realization has the price that a threshold signature has the size of c individual signatures. In Chapter 6 we will see that we can exploit the fact that a threshold signature does not have to carry c identities to construct a realization where a threshold signature has the size of *one* individual signature. This is indeed the very reason for looking at threshold signatures.

3.8.7 The Common Reference String Model

It was proved in [Can01a, CF01] that a number of important two-party functionalities, such as e.g commitment and zero-knowledge proof, cannot be realized in the real-life model by

Functionality $\mathcal{F}_{\text{PRS}}^{gen}$

The functionality runs with parties $P_1, \ldots, P_n$ and is parametrized by a PPT generator gen.

initialize:

In the first round, generate $K_i \leftarrow gen(k; t_i)$ for each party P_i and output K_i on the SOT. If P_i is ever corrupted, then output t_i on the SOT.

key request:

On input (rid, P_i) from P_j, output (rid, P_j, P_i) on the SOT. Let the adversary specify an output round, and in the specified round output (rid, K_i') to P_j, where K_i' is determined as follows: If P_i is corrupted, then proceed as follows: If the SIT contains a value (rid, K, t), then $K_i' = gen(k; t)$. Otherwise $K_i' = \perp$. If P_i is honest, then $K_i' = K_i$.

Figure 3.33: The private reference string functionality with generator gen.

a two-party protocol. Since it is clearly important for practical applications to have two-party protocols realizing these two-party functionalities we cannot just give up because of the bad news. The way out of the misery is to consider some realistic hybrid model where the functionalities can be realized. One such model is the common reference string (CRS) model, where we assume that that a common reference string with a prescribed distribution is available to the players. If a CRS is available, then two-player solutions do exist, as demonstrated in [CF01]. The common reference string model is discussed in more detail in [Can01b] and [CR03]. The CRS can be modeled using the ideal functionality in Fig. 3.32 on the preceding page.

When we say that a functionality can be realized in the CRS model we silently assume that it is for a CRS of size $O(k)$ bits. If a protocol requires a larger CRS, $\Theta(nk)$ bits say, we will mention this explicitly.

3.8.8 The Private Reference String Model

Another hybrid-model that we use a number of times is what we call the private reference string (PRS) model. As the CRS model it is a model which allows to realize some functionalities which cannot be realized in the bare model. It is different from the CRS model in that each party has its own 'reference string' and that no guarantee on the distribution of this string is guaranteed when the party is corrupted, even though we are guaranteed that it is at least of the right form. As discussed in Remark 3.2 on the next page one advantage of the PRS model over the CRS model is that it can be realized using the current Internet PKI environment.

As for the CRS model, the PRS model is parametrized by a generator gen, giving the distribution of the PRS's (of the honest parties). When initialized it generates a key $K_i \leftarrow gen$ for each honest party P_i, and when queried for the key of P_i it forwards K_i. To model that corrupted parties must at least choose a key of the correct form, we require that a corrupted party P_j specifies K_j by specifying $t_j \in \{0,1\}^*$ such that $K_j = gen(t_j)$. The ideal functionality is given in Fig. 3.33.

Notice that the keys forwarded for corrupted parties need not be the same for all parties or have any particular distribution. Furthermore, the keys of corrupted parties might depend on the keys of honest parties, as these are revealed on the SOT. All that is guaranteed is that the trapdoors for all keys are known by $\mathcal{F}_{\text{PRS}}$.

The functionality $\mathcal{F}_{\text{PRS}}$ has some similarity to a PKI, letting K be the public key and letting t be the private key, with the important difference from a PKI that the honest parties do not know t. This has some important implications as no protocol using $\mathcal{F}_{\text{PRS}}$ can 'misuse' it in a way revealing t. This will in many settings allow protocols to use the same $\mathcal{F}_{\text{PRS}}$ in a secure manner. At the time of writing, this is ongoing research and will not be discussed further here.

Remark 3.2 *In practice, the functionality Fig. 3.33 on the preceding page could be realized by having each party P_i generate a key $K_i \leftarrow gen(t_i)$, $t_i \leftarrow_R \{0,1\}^*$ and approach a certificate authority (CA) to register K_i. This would model the* initialize *command. The command* key request *would then be realized by P_j requesting the key from P_i, P_i sending the key K_i along with the certificate and P_j verifying the certificate. If the verification fails, P_j outputs $K_i = \perp$.*

The fact that corrupted parties can specify arbitrary t_j in Fig. 3.33 on the facing page then corresponds to the fact that we cannot enforce that corrupted parties generate their keys at random. The fact that $\mathcal{F}_{\text{PRS}}$ allows corrupted parties to return different keys in different requests corresponds to the fact that corrupted parties might register several keys.

The fact that $\mathcal{F}_{\text{PRS}}$ learns the random bits used to produce the registered keys means that $\mathcal{F}_{\text{PRS}}$ cannot be realized by just any CA. We need that the CA verifies that the key registrant (KR) actually knows the random bits used to generate the registered key.

This corresponds directly to what is known as a proof of Possession of Private key in current PKI lingo [CFS+03]. This denotes the practice that a key registrant (KR) which registers a public key pk must prove that he possesses the private key sk corresponding to pk.

Even though it is not necessarily the case in the current Internet PKI environment that a CA checks that the KR has PoP, it is however recommended that the certification practice statement (CPS) of the CA states [i]f and how the subject [KR] must prove possession of the companion private key for the public key being registered *[CFS+03]. Therefore one should be able to determine from the CPS of the CA and the certificate policy extension fields of an X.509 certificate whether and how the CA checked PoP.*

As discussed in [CFS+03, Note 4], [t]he subject [KR] may not need to prove to the CA that the subject has possession of the private key corresponding to the public key being registered if the CA generates the subject's key pair on the subject's behalf. *A CPS stating that the PoP is guaranteed by having the CA generate the key corresponds directly to our functionality $\mathcal{F}_{\text{PRS}}$.*

The PRS model could e.g. be instantiated to associate to each party a key of a trapdoor commitment scheme. In that case $K \leftarrow gen(t)$ would be the key generator of the trapdoor commitment scheme and t would be the trapdoor.[8] In that case $\mathcal{F}_{\text{PRS}}^{gen}$ would guarantee that

[8] We could e.g. have $K = (P, Q, g, h = g^x \bmod P)$ and $t = x$ for a prime P, an element g of prime order Q and uniformly random $x \in \mathbf{Z}_Q$. Then $\text{commit}_K(m) = h^m g^r \bmod P$ for uniformly random $r \in \mathbf{Z}_Q$ is unconditionally hiding and computationally binding for $m \in \mathbf{Z}_Q$, if computing discrete logarithms base g is

the key K of an honest party is random, so that commit_K is computational binding. Furthermore, since the hiding property is unconditional it holds for arbitrary keys, so commit_K is always unconditionally hiding. If therefore in a given protocol for the $\mathcal{F}_{\text{PRS}}^{gen}$-hybrid model a party P_i wants to commit to a value m to the party P_j, this can be done by P_i requesting the commitment key K_j of P_j and sending $c = \text{commit}_{K_j}(m)$ to P_j. Since commit_{K_j} is unconditionally hiding even if P_j is corrupted this is 'secure' for P_i. Furthermore, since K_j is correctly generated when P_j is honest, P_j is guaranteed that P_i can only open c to one value. This is an example where it does not matter that a corrupted P_j can chose K_j according to another distribution than *gen*. The randomness of K_j is only used to guarantee the computational binding of commit_{K_j} and the computation binding is in the interest of P_j only.[9] For a further discussion of setting up the private reference string, using e.g. a certificate authority, see Section 2.9.5.

It is easy to see that the PRS model can be realized in the CRS model, using a CRS consisting of the concatenation of the PRS of each party. Another approach would have each party P_i generate $K_i \leftarrow gen(t_i)$ and on request of the key send K_i to the requester and then prove knowledge of t_i. This proof of knowledge would however have to be a universally composable zero-knowledge proof of knowledge, which again requires e.g. the CRS model. But since zero-knowledge proof of knowledge can be realized with a short CRS, see e.g. [CF01], the later approach might be preferable. We note that the other direction holds too: The CRS model can be realized in the PRS model for an appropriate generator *gen* and considering a modified formulation of $\mathcal{F}_{\text{PRS}}$, which does not reveal t_i when P_i is corrupted. Again this is ongoing research and will not be discussed further here.

3.9 The Random-Oracle Model

In this section we describe how to model the random-oracle model within our model of secure protocols. Informally, the random-oracle model for a hash-function h is the model where h is replaced by a uniformly random function. The rational behind the random-oracle model is that by modeling primitives as DES, MD5 or SHA using the strong assumption that they (properly used/modified) behave like random functions, one can build efficient and secure protocols based on these primitives. The model has been used to argue the security of a number of constructions. Examples are the OAEP encryption mode for RSA [BR95, Sho01], the Fiat-Shamir heuristic [FS86], and the efficient Byzantine agreement protocol of Cachin, Kursawe, and Shoup [CKS00]. To distinguish the random-oracle devoid model from the random-oracle model, we will call it the first the complexity-theoretic model (CT model).

We define the RO model to be the real-life execution model where additionally the parties have access to a uniformly random function $H : \{0,1\}^k \to \{0,1\}^k$. This can be modeled within the framework in [Can01a] using a hybrid model with an ideal functionality for evaluating H. Because our formulation of the UC framework does not allow for immediate functionalities this approach is not immediately applicable, but we can take a similar ap-

hard. To guarantee that $\text{commit}_K(m) = h^m g^r \bmod P$ is unconditionally hiding it is enough that Q is prime and that the order of g in $\mathbf{Z}_P^*$ is Q and that $h \in \langle g \rangle_Q$. Therefore also keys registered by corrupted parties define unconditionally hiding trapdoor commitment schemes

[9] Assuming that no other party accepts commitments under K_j.

proach: A protocol for the random-oracle model is as usual n ITMs $\pi = (P_1, \ldots, P_n)$, but with the change that the machine P_i is an ITM with oracle access. I.e. we assume that it has an oracle input tape, an oracle output tape and an oracle state. An ITM with oracle access P is executed in the presence of some oracle O, also an ITM; We write P^O. When P enters the oracle state, the contents of the oracle output tape is written to the input tape of O and O is run to produce an output which is written to the oracle input tape of P. Then P resumes normal execution, i.e. its transition function is applied to the new state with the updated oracle input tape. The protocol will then simply be run while giving the parties access to a common uniformly random function H; We write $\mathrm{REAL}_{\pi^H,\mathcal{Z}}$. When $\mathcal{Z}$ corrupts a party P_i in π, then $\mathcal{Z}$ takes over the powers of P_i, e.g. it can now send messages on behalf of P_i. To be able to run P_i we need that $\mathcal{Z}$ has access to H. This access can be provide in two different ways. We can either let $\mathcal{Z}$ be an ITM with oracle access and then simply run $\mathcal{Z}$ with access to H. Another approach is to let $\mathcal{Z}$ access H through the command system by outputting commands of the form $(\texttt{RO-query}, x)$ in response to which it receives $H(x)$. The principal difference between the approaches is whether the interface is allowed to simulate the replies to $\mathcal{Z}$ in the ideal world, and the two choices lead to models with different strength, named the programmable random-oracle model (PRO model) respectively the non-programmable random-oracle model (NPRO model) in [Nie02a]

3.9.1 The Non-Programmable Random-Oracle Model

We describe the non-programmable random-oracle model in more detail. In this model, when $\mathcal{Z}$ wants to query H it outputs a value $(\texttt{RO-query}, x)$ and will be returned $H(x)$. This way of modeling the random oracle basically corresponds to an execution $\mathrm{HYB}^H_{\pi,\mathcal{Z}}$ in the hybrid model, where H is a functionality for evaluating a uniformly random function and where $\mathcal{Z}$ is allowed to request evaluations on the SIT of H.

The changes to the definitions of security are straight-forward. E.g., we say that π^H realizes $\mathcal{F}$ in the NPRO model if there exists an interface $\mathcal{S}$ such that for all environments $\mathcal{Z}$ it holds that $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{\mathrm{c}}{\approx} \mathrm{REAL}_{\pi^H,\mathcal{Z}}$ when H is chosen as a uniformly random function $H : \{0,1\}^k \to \{0,1\}^k$. Notice that no function H is present in $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, so along with all the other commands issued by $\mathcal{Z}$, the simulator must also simulate H. This is the reason why we call the random oracle programmable. When simulating, $\mathcal{S}$ is free to pick the replies $H(x)$ to the commands $(\texttt{RO-query}, x)$ from $\mathcal{Z}$ as it desires. We say that $\mathcal{S}$ can program the random-oracle H.

3.9.2 The Programmable Random-Oracle Model

We describe the programmable random-oracle model in more detail. In this model we let the environment $\mathcal{Z}$ and the interface $\mathcal{S}$ be ITMs with oracle access and we say that π^H realizes $\mathcal{F}$ in the NPRO model if there exists an interface $\mathcal{S}$ such that for all environments $\mathcal{Z}$ it holds that $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}^H,\mathcal{Z}^H} \overset{\mathrm{c}}{\approx} \mathrm{REAL}_{\pi^H,\mathcal{Z}^H}$, when H is chosen as a uniformly random function $H : \{0,1\}^k \to \{0,1\}^k$.

The reason for calling this model the *non-programmable* random-oracle model has to do with the power of the interface $\mathcal{S}$ when simulating. Assume that $\mathcal{S}$ runs internally a simulated copy of π. Consider again the programmable random-oracle model. In the execution

$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ the simulated copies of $P_1, \ldots, P_n$ will make oracle calls to evaluate H. Since π is being run inside the interface $\mathcal{S}$, it is free to choose the value $H(x)$ to return to P_i when P_i queries on x. Contrast this to the non-programmable random-oracle model, where the execution $\text{IDEAL}_{\mathcal{F},\mathcal{S}^H,\mathcal{Z}^H}$ is considered. Here $\mathcal{Z}$ has access to H and therefore has the opportunity of verifying that the observed communication is consistent with H. This is of course always the case in $\text{REAL}_{\pi^H,\mathcal{Z}^H}$, so in $\text{IDEAL}_{\mathcal{F},\mathcal{S}^H,\mathcal{Z}^H}$ the interface $\mathcal{S}$ is forced to simulate an execution of π which is consistent with the supplied H. This means that $\mathcal{S}$ is no longer free to define $H(x)$ as desired. The first formalization of the difference between the PRO and the NPRO models appears in [Nie02a]. The distinction between these RO models has e.g been used by Ishai, Kilian, Nissim and Petrank [IKNP03] in the context of extending oblivious transfers efficiently in the NPRO model.

3.9.3 Implementing the Random-Oracle

To obtain an efficient implementation of a protocol for the random-oracle model, the function H can be implemented using e.g. a cryptographic hash-function or in some cases using a so-called probabilist hash-function [Can97]. In the paper [BR93] where Bellare and Rogaway introduce the RO model it is said about the methodology of proving schemes secure in the random-oracle model and then instantiating the oracle with a carefully chosen function in practice that *'It is our* thesis *that this method, when carried out, leads to secure and efficient protocols. Indeed, protocols constructed under this paradigm have so far proved "secure" in practice. But we stress that all claims of provable security are claims made within the random-oracle model, and instantiating the oracle with h is only a heuristic whose success we trust from experience.'* This is an important point to observe. That π^H realizes $\mathcal{F}$ when H is a uniformly random function does not guarantee that π^h realizes $\mathcal{F}$ for a hash-function h. This point was seriously stressed in [CGH98] by Canetti, Goldreich and Halevi by constructing an encryption scheme which is secure in the RO model, but is not secure in the CT model, no matter the instantiation of the RO. Their scheme is constructed as to try to "detect" whether it is in the RO model or not, and then reveal the secret key if it is not in the RO model.

3.9.4 Separating the Models

Above we gave two different RO models. It turns that the choice between these two ways of modeling a hash-function using a random oracle is an essential one. Being able to program the oracle is a very powerful tool to have in the simulation, and it is used by almost all applications of the random-oracle model, including the above mentioned result from [BR95, Sho01,FS86,CKS00] — and it is used essentially, i.e. without the ability to program the oracle most of these results do not stand. In Sections 4.2 and 4.3 we give a formal proof that the models are essentially different. What we show there is that there exists a natural protocol problem which has a simple solution in the RO model and which has no solution in the NPRO model, namely the problem of constructing a non-interactive communication protocol secure against adaptive adversaries a.k.a. non-interactive non-committing encryption. This means that the PRO model is stronger than the NPRO model in terms of which problems can be solved. This separation result first appeared in [Nie02a].

3.9.4.1 Previous Separation Results.

Other examples of constructions secure in the RO model and not secure in weaker models were known prior to [Nie02a]. Most prominently is the mentioned result from [CGH98] that there exists an encryption scheme which is secure in the RO model, but is not secure in the CT model, no matter the instantiation of the RO. This result showed that the RO is stronger than the CT model in the sense that the exists realizations which are secure in the RO and which are not secure in the CT model.

One strength of the result from [CGH98] is that it is the semantic security of the encryption scheme that is violated in the CT model, whereas it in our example it is the less standard non-committing property that is violated. Their result thus establishes that even standard security properties do not carry over from the RO model to the CT model. Another strength of the result from [CGH98] is that their encryption scheme can be proved secure in the NPRO model, as they do not use the programmability of random-oracle. This means that their result separates the CT model and the NPRO model.[10]

Another separation result is that the Fiat-Shamir [FS86] methodology can be proved secure in the RO model. This can be viewed as being a separation result, as it is proved in [GK90] that in the CT model there does not exist zero-knowledge protocols for NP with less than four rounds unless NP is in BPP. And the above observation is indeed a separation if one insists that the RO must be realized using a fixed function h. However, it would anyway be naive to expect this to be more than 'heuristically secure', as noted by the initiators of the RO model in [BR93], where it is said that *'We stress that the protocol problem* Π *and protocol* P *must be "independent" of the hash function we are to use. It is easy to construct unnatural problems or protocols whose description and goals depend explicitly on* h *so that the protocol is secure in the RO model but fails when the RO is instantiated with the hash function. The notion of "independence" will not be formalized in this paper.'* Therefore the oracle should at least be implemented by a random function h drawn from some function family, say, by a trusted party, and handed to both the prover and the verifier. But h is then a de facto common random string and the existence of one-round zero-knowledge proofs is no longer ruled out [BFM88]. It does in particular not follow that in some preprocessing model there does not exist some kind of non-interactive instantiation of the RO which makes the methodology secure. The (in)security of the Fiat-Shamir methodology is investigated in great detail in [GT03] by Goldwasser and Tauman.

A recent separation result is that of Bellare, Boldyreva and Palacio [BBP03], where a uninstantiable scheme for IND-CPA-preserving asymmetric encryption is proved secure in the RO model. This separation result is more serious than previous results in that the proposed scheme is very natural and by inspection shows no signs of being constructed as to frustrated the RO model or being insecure for other reasons. This example seems to be

[10] The scheme from [CGH98] uses the non-interactive CS proofs of Micali [Mic00] for the RO model, and so for their non-interactive encryption scheme to be secure it should be verified that these CS proofs are secure in the NPRO model, which is done fairly straight-forwardly, as the proof in [Mic00] does not use the programmability of the oracle. We will however not prove this here, as under all circumstances, if the CS proofs for the RO model are replaced by interactive CS proofs (which can be constructed in the CT model under the assumption that collision resistant hash functions exits [Mic00]), then it is trivial to verify that the scheme in [CGH98] is secure in the NPRO model.

the first one seriously contradicting the thesis in [BR93] that the random oracle paradigm leads to secure schemes when one does not deliberately try to frustrate it. Their example does however not seem to be a separation between the NPRO model and the CT model.

More recently Maurer, Renner and Holenstein [MRH04] introduced a generalized notion of indistinguishability, called indifferentiability, along with its theory. This theory allows to demonstrate in a simple and precise manner the separation between the NPRO model and the CT model first demonstrated in [CGH98].

We mention one last separation result. In [Pas03] Pass formalized a notion of deniability of zero-knowledge proofs and proved that it is possible to obtain in the RO model but not in the CRS model. This in particular rules out realizing the deniable zero-knowledge in the model where a random hash-function drawn from a function family is given to all parties. As the separation result in Sections 4.2 and 4.3, the result of Pass provides a separation at the level of which problems can be realized.

To the best of our knowledge the only known results separating the NPRO model and the CT model is the results of [CGH98, MRH04]. This means that all known separation result between the NPRO model and the CT model to some extend is of the form anticipated by the initiators of the RO model: Though the scheme in [CGH98] does not depend on a single function or function family it uses essentially the description of the specific function or function family used, and [MRH04] does not provide separating schemes, but derives the separation result from a general theory of indifferentiability. It would be interesting to see if it is possible to construct examples of cryptographic schemes separating the NPRO model and the CT model, which is not of the form anticipated by the above quotation from [BR93].

3.9.4.2 The 'Right' Formulation

We have provided two formulations of the random-oracle model and claimed that we prove that they are essentially different. So, there seems to be a choice to be made. Which is the 'right' formulation? First of all, clearly the question does not make sense. But should one prefer any of the formulations, probably it would be the PRO model. From the separation between the PRO model and the NPRO model, one might get the idea that the NPRO model is 'safer' to use. However, we know from [CGH98] that the NPRO model is not sound, so the NPRO model is safer than the PRO model in the same way that jumping out the window on the tenth floor is safer than jumping out the window on the eleventh floor. With both models one has to keep in mind that *'[...] all claims of provable security are claims made within the random-oracle model, and instantiating the oracle with h is only a heuristic whose success we trust from experience.'* So, on that front little is obtained from using the NPRO model. A lot is lost however, as most interesting applications of the random-oracle model *needs* the programmability. So, since the motivation for introducing the RO model was to be able to model e.g. hash-functions in a way strong enough to prove more schemes secure, and none of the models are sound, the PRO model is clearly preferable. For any problem where it is not provably *necessary* to use a RO model one should of course prefer not to use it at all.

3.10 Multi-Round Functionalities

When specifying an ideal functionality $\mathcal{G}$ it is often convenient to specify it such that it takes an input from each party in each round and then delivers the desired results to all parties in the same round. This makes the functionality easy to use. However, in many cases this results in a functionality $\mathcal{G}$ which cannot be realizes because the desired result cannot be computed by any protocol in one round. Therefore one will have to specify a functionality $\mathcal{G}'$ which takes an input from each party and then in some subsequence round, either given by some round-complexity parameter or specified by the adversary, delivers the desired results to the parties. Such multi-round ideal functionalities are however less convenient to use as a helping functionality in a hybrid model, because the protocol using the multi-round ideal functionality $\mathcal{G}'$ must constantly deal with the fact that $\mathcal{G}'$ might be delayed. In this section we therefore establish a general framework which allows to prove a protocol secure in the $\mathcal{G}$-hybrid model and then transform it, using a generic transformation, into a secure protocol for the $\mathcal{G}'$-hybrid model.

As mentioned, many functionalities $\mathcal{G}$ which delivers the result in one round cannot be realized. We can however still take a protocol π for the $\mathcal{G}$-hybrid model and replace calls to $\mathcal{G}$ with calls to a protocol γ which realizes $\mathcal{G}$ except that the protocol uses multiple rounds for computing the results — during the rounds until γ delivers a result the parties in π just wait. We then obtain a protocol which except for the round complexity has the same input-output behavior as π. We will formalize what it means for this to be secure and then prove that indeed it is. We do this in three steps. First we show how to transform any functionality $\mathcal{F}$ into a multi-round functionality MultiRound($\mathcal{F}$), where MultiRound($\mathcal{F}$) is a functionality which basically runs $\mathcal{F}$, except that each original round of $\mathcal{F}$ is allowed to take a number of rounds determined by the adversary. Then we show how to change a protocol π for the $\mathcal{G}$-hybrid model into a protocol MultiRound(π) for the MultiRound($\mathcal{G}$)-hybrid model. We then show that if π realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, then MultiRound(π) realizes MultiRound($\mathcal{F}$) in the MultiRound($\mathcal{G}$)-hybrid model.

3.10.1 Multi-Round Functionalities

In the following we will need to talk about rounds where the parties, and the functionalities, do not output any value. Since our framework requires all parties, and all functionalities, to send a value to all parties in all rounds, we add a value $\perp \notin \{0,1\}^*$ to the messages that can be send and introduce the convention that $\perp$ means 'no value'.

Definition 3.25 *Let $V \in \{0,1\}^*$ and let $(\cdot,\cdot) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be the encoding from Section 2.1. Parse V as $(b, V') \leftarrow V$ for $b_i \in \{0,1\}$ and $V' \in \{0,1\}^*$. If $b = 0$ then we say that V is a* trivial value *and if $b = 1$, then we say that V is a* non-trivial value*. We consider all trivial values $(0, V')$ as an encoding of the symbol $\perp \notin \{0,1\}^*$ used to indicate* no value *and we consider the non-trivial value $(1, V')$ as an encoding of $V' \in \{0,1\}^*$.*

Using this convention we can then define a multi-round version MultiRound($\mathcal{F}$) of any functionality $\mathcal{F}$. This functionality is given in Fig. 3.34 on the next page. A value with $b = 0$ in end round results in a trivial activation, which is not meant to activate $\mathcal{F}$. We call the

Functionality MultiRound($\mathcal{F}$)

The functionality MultiRound($\mathcal{F}$) runs with parties $P_1, \ldots, P_n$ and proceeds as follows:

initialization:
On input (k, r) input (k, r) to $\mathcal{F}$.

party activation:
On input (i, x_i), parse x_i as (b_i, x_i'). If $b_i = 0$, then output $(0, x_i)$ on the SOT; Otherwise, input (i, x_i') to $\mathcal{F}$ to receive a value v on the SOT of $\mathcal{F}$. Then output $(1, v)$ on the SOT.

end round:
On input $(\texttt{activate}, v)$, parse v as (b, v'). If $b = 0$, then output $(0, \epsilon)$ to all parties. If $b = 1$, then input $(\texttt{activate}, v')$ to $\mathcal{F}$ to receive the outputs $\{y_i\}_{i\in[n]}$. Then output $(1, y_i)$ to P_i in this round.

Figure 3.34: The multi-round version of a synchronous ideal functionality.

Party MultiRound(P_i)

The algorithm MultiRound(P_i) can be consider an ITM saving its state in $m_{i,i,r}$. We describe the party MultiRound(P_i) using ITM terminology. It proceeds as follows:

1. Receive (k, r_i). Let $r \leftarrow 0$ be a round counter. Let $m_{j,i,0} = \epsilon$ for $j \in [n]$, let $m_{i,i,0} = (k, r_i)$ and let $s_{i,0} = \epsilon$. Wait until the first round where $\mathcal{Z}$ inputs a value of the form $(1, x_{i,1})$.
2. Let $r \leftarrow r + 1$. Then run
$$(\{m_{i,j,r}\}_{j\in[n]}, y_{i,r}, t_{i,r}) = P_i(\{m_{j,i,r-1}\}_{j\in[n]}, x_{i,r}, s_{i,r-1})$$
and output $(\{m_{i,j,r}\}_{j\in[n]}, y_{i,r}, (1, t_{i,r}))$. *I.e. in intuitive terms, send* $m_{i,j,r}$ *to* P_j *and input* $(1, t_{i,r})$ *to* MultiRound($\mathcal{G}$).
3. In the next round, save the incoming messages $\{m_{j,i,r}\}_{j\in[n]}$ and wait until the first round where MultiRound($\mathcal{G}$) outputs a message $(1, s_{i,r})$. In the rounds until this happens, output $(0, \epsilon)$ to $\mathcal{Z}$, input $(0, \epsilon)$ to MultiRound($\mathcal{G}$) and send ϵ to all parties, and if $\mathcal{Z}$ inputs anything in these rounds, then simply ignore. In the round where MultiRound($\mathcal{G}$) outputs $(1, s_{i,r})$, output $(1, y_{i,r})$ to $\mathcal{Z}$. Then wait until the first round where $\mathcal{Z}$ inputs a value of the form $(1, x_{i,r+1})$ and goto Step 2.

Figure 3.35: MultiRound(P_i).

activations which results in $\mathcal{F}$ being activated the non-trivial activations. Notice that if all parties get to input exactly one non-trivial value to MultiRound($\mathcal{F}$) between each non-trivial activation, then $\mathcal{F}$ is activated on a sequence of inputs which is possible when $\mathcal{F}$ is run in the $\mathcal{F}$-hybrid model or the ideal process for $\mathcal{F}$.

3.10.2 Multi-Round Protocols

We then present a transformation of a $\mathcal{G}$-hybrid protocol into a corresponding MultiRound($\mathcal{G}$)-hybrid protocol. Let $\pi = (P_1, \ldots, P_n)$ be a protocol for the broadcast model, i.e. P_i is a PPT algorithm

$$(\{m_{i,j,r}\}_{j\in[n]}, y_{i,r}, s_{i,r}) = P_i(\{m_{j,i,r-1}\}_{j\in[n]}, x_{i,r}, t_{i,r-1}) \ ,$$

where $t_{i,r-1}$ is the output from $\mathcal{G}$ in the previous round and $s_{i,r}$ is the input to $\mathcal{G}$ in the next round. The party MultiRound(P_i) is given in Fig. 3.35 on the facing page, and we let MultiRound(π) = (MultiRound(P_1), ..., MultiRound(P_n)).

3.10.3 Multi-Round IO Behaviors

Since a multi-round functionality MultiRound($\mathcal{F}$) requires that all parties inputs non-trivial inputs between each non-trivial activation, we have to put a restriction on the IO behavior.

Definition 3.26 *The non-overlapping synchronous IO behavior $\mathcal{IO}_{\text{MultiRound}}$ is defined as follows:*

- *For the IO behavior of the honest parties we have the following restrictions: On input (i, x_i), i.e. input x_i to party P_i, parse x_i as (b_i, x_i') for $b_i \in \{0, 1\}$. If in some round some honest party receives an input of the form $(1, \cdots)$ and some honest party receives an input of the form $(0, \cdots)$, then output* `fail` *as soon as this is detected. Furthermore, if some honest party receives an input of the form $(1, \cdots)$ in two rounds $r_1 < r_2$ and P_i did not output a value of the form $(1, \ldots)$ in any of the rounds $r_1, \ldots, r_2 - 1$, then output* `fail` *when this is detected, i.e. in round r_2. Notice that because* MultiRound($\mathcal{F}$) *outputs the bit b_i in the SOT this IO behavior can be checked given just the values on the SIT and SOT, as required.*

- *For the IO behavior on the ST we have the following restrictions: If in two rounds values of the form $v = (1, v')$ are input on the SIT without all honest parties having output a value of the form $(1, v)$ on the SOT in between these two rounds, then output* `fail`*. Notice that an honest party outputting a value of the form $(1, v)$ means that it got a non-trivial input, see Fig. 3.34 on the preceding page, and notice that this IO behavior can be checked given just the values on the SIT and SOT, as required.*

The $\mathcal{IO}_{\text{MultiRound}}$ behavior basically requires from the honest parties that MultiRound($\mathcal{F}$) is activated in synchronous rounds, and it requires from the entity inputting on the SIT that MultiRound($\mathcal{F}$) is not (non-trivially) activated until all honest parties have been given a fresh non-trivial input. Notice that it is not essential for the correctness of MultiRound($\mathcal{F}$) that the honest parties receive (non-trivial) inputs in synchronous rounds. This part of the non-overlapping synchronous IO behavior is for ensuring the correctness of the protocol MultiRound(π).

As for an ideal functionality and a protocol we can define a multi-round variant of an IO behavior $\mathcal{IO}$. We can consider the IO behavior MultiRound($\mathcal{IO}$), which can be defined via Fig. 3.34 on page 136, by just considering $\mathcal{IO}$ an ITM to which we apply the

MultiRound construction in Fig. 3.34. We let $\mathcal{IO}' = \text{MultiRound}(\mathcal{IO})$ denote the construction in Fig. 3.34 on page 136 applied to $\mathcal{IO}$ — with the one difference that in initialization only k is given to an IO behavior — and we then define $\text{MultiRound}(\mathcal{IO}) = \mathcal{IO}' \wedge \mathcal{IO}_{\text{MultiRound}}$. Intuitively this means that $\text{MultiRound}(\mathcal{IO})$ requires that a given IO sequence IO is according to $\mathcal{IO}_{\text{MultiRound}}$. This allows to defined a sound IO sequence which $\mathcal{F}$ would see if $\text{MultiRound}(\mathcal{F})$ saw IO, and $\text{MultiRound}(\mathcal{IO})$ then requires that this pruned IO sequence is in accordance with $\mathcal{IO}$. For an ideal-functionality with (a possibly empty) IO restriction, $\langle \mathcal{F}, \mathcal{IO} \rangle$, we then define the multi-round version to be the core-functionality $\text{MultiRound}(\mathcal{F})$ with IO restriction $\text{MultiRound}(\mathcal{IO})$.

Definition 3.27 $\text{MultiRound}^{\mathcal{IO}}(\langle \mathcal{F}, \mathcal{IO} \rangle) = \langle \text{MultiRound}(\mathcal{F}), \text{MultiRound}(\mathcal{IO}) \rangle$.

3.10.4 The Multi-Round Compilation Theorem

We then prove that if π realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, then $\text{MultiRound}^{\mathcal{IO}}(\pi)$ realizes $\text{MultiRound}^{\mathcal{IO}}(\mathcal{F})$ in the $\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})$-hybrid model.

Notice that if a protocol $\text{MultiRound}(\pi)$ is run in an environment $\mathcal{Z}$ with non-overlapping synchronous IO behavior, then basically π is run on the input sequence from $\mathcal{Z}$ which is labeled $(1, \cdots)$. All parties $\text{MultiRound}(P_i)$ are activated with an input of the form $(1, x_{i,r})$ in the same round, so all parties $\text{MultiRound}(P_i)$ execute Step 2 in the same round, i.e. execute the parties P_i in the same round. Then the messages of P_i are sent and the value for $\mathcal{G}$ is given to $\text{MultiRound}(\mathcal{G})$. Therefore all the parties $\text{MultiRound}(P_i)$ will receive a non-trivial value from $\text{MultiRound}(\mathcal{G})$ in the same round and will output $(1, y_{i,r})$ to $\mathcal{Z}$ in the same round. By $\mathcal{IO}_{\text{MultiRound}}$ the environment only inputs values of the form $(0, \cdots)$ during these rounds and $\text{MultiRound}(P_i)$ outputs $(0, \epsilon)$. In a subsequent round all parties $\text{MultiRound}(P_i)$ are activated with an input of the form $(1, x_{i,r+1})$ in the same round (under $\mathcal{IO}_{\text{MultiRound}}$), so the parties simultaneously go to Step 2, and another round of π is executed. Basically $\mathcal{Z}$ is running π, but stretched so that in between rounds of π the protocol $\text{MultiRound}(\pi)$ and $\mathcal{Z}$ are exchanging empty messages, we call these rounds IO-silent rounds. Therefore, as long as $\mathcal{Z}$ has the $\mathcal{IO}_{\text{MultiRound}}$ IO behavior it cannot obtain anything against $\text{MultiRound}(P_i)$ which it could not have obtained against π. We formalize this as follows. From $\mathcal{Z}$ with the $\mathcal{IO}_{\text{MultiRound}}$ IO behavior we can construct an environment $\text{UnMultiRound}(\mathcal{Z})$, which is basically $\mathcal{Z}$ with all the IO-silent rounds pruned. The only difference between running π in $\text{UnMultiRound}(\mathcal{Z})$ and running $\text{MultiRound}(\pi)$ in $\mathcal{Z}$ will be the IO-silent rounds. Therefore, letting $\text{UnMultiRound}(\mathcal{Z})$ simulate these IO-silent rounds[11] to $\mathcal{Z}$ we will have that $\text{HYB}^{\mathcal{G}}_{\pi,\text{UnMultiRound}(\mathcal{Z})} = \text{HYB}^{\text{MultiRound}(\mathcal{G})}_{\text{MultiRound}(\pi),\mathcal{Z}}$. The exact details of $\text{UnMultiRound}(\mathcal{Z})$ are given in Fig. 3.36 on the facing page.

Lemma 3.9 *For all environments $\mathcal{Z}$, all $k \in \mathbf{N}$, all $z \in \{0,1\}^*$ and all random bits r it holds that*

$$\text{HYB}^{\mathcal{G}}_{\pi,\text{UnMultiRound}(\mathcal{Z})}(k, r, z) = \text{HYB}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}(\pi),\mathcal{Z}_{|\mathcal{IO}_{\text{MultiRound}}}}(k, r, z) \ ,$$

where $\mathcal{Z}_{|\mathcal{IO}_{\text{MultiRound}}}$ is the environment which itself checks whether it violates $\mathcal{IO}_{\text{MultiRound}}$ and outputs 1 in case this happens.

[11] The IO-silent rounds are simulated until $\mathcal{Z}$ non-trivially activates $\text{MultiRound}(\mathcal{G})$.

Environment UnMultiRound($\mathcal{Z}$)

initialize:

Receive (k, z) and input (k, z) to $\mathcal{Z}$. Let $O = 0$, let $I_i = 0$ for $i \in [n]$ and let $H = [n]$.

environment activation:

Activate $\mathcal{Z}$ to obtain a command c. If the output is $(\texttt{guess}, b)$, then output c. Otherwise the commands are handled as described below.

party activation:

If $c = (\texttt{activate}, i, x_{i,r}, \{m_{j,i,r-1}\}_{j\in C})$, then parse $x_{i,r}$ as $(b_i, x'_{i,r})$ and do one of the following:

1. If $b_i = 0$ or $I_i = O + 1$ (meaning that P_i already received $O + 1$ inputs and only output O times), then simulate to $\mathcal{Z}$ that P_i outputs $(0, \epsilon)$, sends ϵ to all parties, and inputs ϵ to $\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})$ (Formally, hand $v_{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})} = (0, \epsilon)$ and $\{m_{i,j,r} = \epsilon\}_{j\in[n]\setminus\{i\}}$ to $\mathcal{Z}$). *This simulates an activation of Step 3 in Fig. 3.35 on page 136. Then go to environment activation.*
2. If $b_i = 1$ and $I_i = O$ (meaning the P_i completed O rounds and $\mathcal{Z}$ gave a non-trivial input), then let $I_i \leftarrow I_i + 1$ and output $(\texttt{activate}, i, x'_{i,r}, \{m_{j,i,r-1}\}_{j\in C})$. *In* $\text{HYB}^{\mathcal{G}}_{\pi,\text{UnMultiRound}(\mathcal{Z})}$ *the party* P_i *is then activated to compute* $(\{m_{i,j,r}\}_{j\in[n]}, y_{i,r}, t_{i,r}) = P_i(\{m_{j,i,r-1}\}_{j\in[n]}, x_{i,r}, s_{i,r-1})$. *Then* $t_{i,r}$ *is given to* $\mathcal{G}$ *which either results in the IO restriction of* $\mathcal{G}$ *producing an error or a value* $v_{i,r}$ *on the SOT of* $\mathcal{G}$. *In the last case* UnMultiRound($\mathcal{Z}$) *receives* $v_{i,r}$ *along with* $\{m_{i,j,r}\}_{i\in[n]\setminus\{i\}}$. Input $(1, v_{i,r})$ to $\mathcal{Z}$ to simulate an activation of $\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})$ and hand $\{m_{i,j,r}\}_{i\in[n]\setminus\{i\}}$ to $\mathcal{Z}$ to simulate the sending of these messages in $\text{HYB}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}(\pi),\mathcal{Z}}$. This simulates an activation of Step 2 in Fig. 3.35 on page 136.

Figure 3.36(a) (cont. in Fig. 3.36(b) on the following page): UnMultiRound($\mathcal{Z}$)

Proof. It is straight-forward to verify that as long as $\mathcal{Z}$ maintains $\mathcal{IO}_{\text{MultiRound}}$, then

$$\text{HYB}^{\mathcal{G}}_{\pi,\text{UnMultiRound}(\mathcal{Z})}(k, r, z) = \text{HYB}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}(\pi),\mathcal{Z}_{|\mathcal{IO}_{\text{MultiRound}}}}(k, r, z)$$

and that if $\mathcal{Z}$ violates $\mathcal{IO}_{\text{MultiRound}}$, then

$$\text{HYB}^{\mathcal{G}}_{\pi,\text{UnMultiRound}(\mathcal{Z})}(k, r, z) = 1 = \text{HYB}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}(\pi),\mathcal{Z}_{|\mathcal{IO}_{\text{MultiRound}}}}(k, r, z) \ .$$

This verification is elaborated on as part of the construction in Fig. 3.36. □

We have now argued that if MultiRound($\mathcal{Z}$) is required to stick to $\mathcal{IO}_{\text{MultiRound}}$, then it cannot obtain anything against MultiRound(π) which it could not have obtained against π. We are however not satisfied with this way of arguing. We want to capture the idea in terms of realizing $\text{MultiRound}^{\mathcal{IO}}(\mathcal{F})$. For this we need yet a lemma:

Lemma 3.10 *For all hybrid interfaces* $\mathcal{T}$*, there exists a hybrid interface* MultiRound($\mathcal{T}$) *such that for all functionalities* $\mathcal{F}$ *and all hybrid interfaces* $\mathcal{Z}$ *it holds that*

$$\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\text{UnMultiRound}(\mathcal{Z})} = \text{IDEAL}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}^{\mathcal{IO}}(\mathcal{F}),\text{MultiRound}(\mathcal{T}),\mathcal{Z}_{|\mathcal{IO}_{\text{MultiRound}}}} \ .$$

Environment UnMultiRound($\mathcal{Z}$)

corrupt:

If $c = (\texttt{corrupt}, i)$, then output $(\texttt{corrupt}, i)$, and receive the random bits r_i used in MultiRound(P_i) to run P_i, and hand r_i to $\mathcal{Z}$. Furthermore, hand ϵ to $\mathcal{Z}$ to simulate the (empty) value from MultiRound$^{\mathcal{IO}}(\mathcal{G})$ to $\mathcal{Z}$ when P_i is corrupted. *Notice that if an IO restriction is violated in $\mathcal{G}$ or $\mathcal{F}$ it results in the same output as when broken in* MultiRound$^{\mathcal{IO}}(\mathcal{G})$ *or* MultiRound$^{\mathcal{IO}}(\mathcal{F})$.

end round:

If $c = (\texttt{end round}, v)$, then parse v as (b, v'), and do one of the following:

1. If $I_i \neq I_j$ for honest P_i and P_j, then $\mathcal{IO}_{\text{MultiRound}}$ was violated by $\mathcal{Z}$, so terminate with output $(\texttt{guess}, 1)$. Otherwise, let I be the common value of the I_i for $i \in H$.
2. If $b = 0$ or $I = O$, then give the values $\{y_{i,r} = \epsilon\}_{i \in H}$ to $\mathcal{Z}$ to simulate the output of the parties in Step 3 in Fig. 3.35 on page 136 when MultiRound$^{\mathcal{IO}}(\mathcal{G})$ does not output a value. Then go to environment activation.
3. If $b = 1$ and $I = O + 1$, then let $O \leftarrow O + 1$ and output $(\texttt{end round}, v')$ and receive the output values $\{y_{i,I}\}_{i \in H}$ of honest parties and receive the output v on the SOT of $\mathcal{G}$. Then input $\{(1, y_{i,I})\}_{i \in H}$ to $\mathcal{Z}$ to simulate the outputs of MultiRound(P_i), and input $(1, v)$ to $\mathcal{Z}$ to simulate the output of MultiRound$^{\mathcal{IO}}(\mathcal{G})$. Notice that in HYB$^{\mathcal{G}}_{\pi,\text{UnMultiRound}(\mathcal{Z})}$ the functionality $\mathcal{G}$ has received an input $t_{i,I} \in \{0,1\}^*$ from all honest parties and is now activated on a value $v' \in \{0,1\}^*$, resulting in $\mathcal{G}$ outputting a value $s_{i,I}$ to all parties. *From the view of $\mathcal{Z}$, believing it runs the parties* MultiRound(P_i)*, these parties should be in Step 3 in Fig. 3.35 on page 136 and should just have received* $(1, s_{i,I})$ *from* MultiRound$^{\mathcal{IO}}(\mathcal{G})$. *This should result in the party* MultiRound(P_i) *outputting* $(1, y_{i,I})$*, as simulated correctly above.*

Figure 3.36(b) (cont. from Fig. 3.36(a) on the preceding page): UnMultiRound($\mathcal{Z}$)

Proof. The idea behind the construction of MultiRound($\mathcal{T}$) is as follows: In

$$\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\text{UnMultiRound}(\mathcal{Z})}$$

the interface $\mathcal{T}$ sees $\mathcal{F}$ and simulates some view to UnMultiRound($\mathcal{Z}$). Then UnMultiRound($\mathcal{Z}$) adds some IO-silent rounds to this view, depending on when $\mathcal{Z}$ non-trivially activates (the simulated view of) MultiRound$^{\mathcal{IO}}(\mathcal{G})$. The IO-silent rounds output by $\mathcal{Z}$ are pruned by UnMultiRound($\mathcal{Z}$) and only the non-trivial inputs are relayed to $\mathcal{F}$. In

$$\text{IDEAL}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}^{\mathcal{IO}}(\mathcal{F}),\text{MultiRound}(\mathcal{T}),\mathcal{Z}_{|\mathcal{IO}_{\text{MultiRound}}}}$$

the functionality MultiRound$^{\mathcal{IO}}(\mathcal{F})$ actually has IO-silent rounds. Since $\mathcal{T}$ does not expect to see IO-silent rounds, these are pruned by MultiRound($\mathcal{T}$) and the remaining view, of $\mathcal{F}$, is presented to $\mathcal{T}$. Then $\mathcal{T}$ produces the same view as in

$$\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\text{UnMultiRound}(\mathcal{Z})} \ .$$

To this view $\text{MultiRound}(\mathcal{T})$ then adds simulated IO-silent rounds as do $\text{MultiRound}(\mathcal{Z})$ in

$$\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\text{UnMultiRound}(\mathcal{Z})} \ .$$

This will then be the view presented to $\mathcal{Z}$ in

$$\text{IDEAL}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}^{\mathcal{IO}}(\mathcal{F}),\text{MultiRound}(\mathcal{T}),\mathcal{Z}_{|\mathcal{IO}_{\text{MultiRound}}}} \ .$$

By construction the data-flows in

$$\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\text{UnMultiRound}(\mathcal{Z})}$$

and

$$\text{IDEAL}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}^{\mathcal{IO}}(\mathcal{F}),\text{MultiRound}(\mathcal{T}),\mathcal{Z}_{|\mathcal{IO}_{\text{MultiRound}}}}$$

are identical, the only difference being that the code for adding and pruning IO-silent rounds are placed in different entities in the two. □

Theorem 3.9 *If π realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, then* $\text{MultiRound}(\pi)$ *realizes* $\text{MultiRound}^{\mathcal{IO}}(\mathcal{F})$ *in the* $\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})$*-hybrid model.*

Proof. By the premise of the theorem we have that there exists a hybrid interface $\mathcal{T}$ such that for all environments $\mathcal{Z}$ where

$$\Pr[\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\text{UnMultiRound}(\mathcal{Z})} \in \text{``H} \rightsquigarrow \mathcal{F}\text{''}] \overset{\text{c}}{\approx} 0 \tag{3.24}$$

it holds that

$$\Pr[\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\text{UnMultiRound}(\mathcal{Z})} \in \text{``H} \rightsquigarrow \mathcal{G}\text{''}] \overset{\text{c}}{\approx} 0 \tag{3.25}$$

and

$$\text{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\text{UnMultiRound}(\mathcal{Z})} \overset{\text{c}}{\approx} \text{HYB}^{\mathcal{G}}_{\pi,\text{UnMultiRound}(\mathcal{Z})} \ . \tag{3.26}$$

We then claim that for all hybrid environments $\mathcal{Z}$ were

$$\Pr[\text{IDEAL}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}^{\mathcal{IO}}(\mathcal{F}),\text{MultiRound}(\mathcal{T}),\mathcal{Z}} \in \text{``H} \rightsquigarrow \text{MultiRound}^{\mathcal{IO}}(\mathcal{F})\text{''}] \overset{\text{c}}{\approx} 0 \tag{3.27}$$

it holds that

$$\Pr[\text{IDEAL}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}^{\mathcal{IO}}(\mathcal{F}),\text{MultiRound}(\mathcal{T}),\mathcal{Z}} \in \text{``H} \rightsquigarrow \text{MultiRound}^{\mathcal{IO}}(\mathcal{G})\text{''}] \overset{\text{c}}{\approx} 0 \tag{3.28}$$

and

$$\text{IDEAL}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}^{\mathcal{IO}}(\mathcal{F}),\text{MultiRound}(\mathcal{T}),\mathcal{Z}} \overset{\text{c}}{\approx} \text{HYB}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}(\pi),\mathcal{Z}} \ , \tag{3.29}$$

proving the theorem. That $\text{MultiRound}(\pi)$ uses $\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})$ according to $\mathcal{IO}_{\text{MultiRound}}$ under $\mathcal{IO}_{\text{MultiRound}}$ is straight-forward to verify. So, it is enough to prove that

$$\Pr[\text{IDEAL}^{\text{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\text{MultiRound}^{\mathcal{IO}}(\mathcal{F}),\text{MultiRound}(\mathcal{T}),\mathcal{Z}} \in \text{``H} \rightsquigarrow \mathcal{G}\text{''}] \overset{\text{c}}{\approx} 0 \tag{3.30}$$

and Eq. 3.29 on the preceding page. From Eq. 3.27 on the page before and Lemma 3.10 on page 140 we get Eq. 3.24 on the page before which allows us to conclude Eq. 3.25 on the preceding page and Eq. 3.26 on the page before. From Eq. 3.25 on the preceding page and Lemma 3.10 on page 140 we get Eq. 3.30. By Lemma 3.9 on page 138, Eq. 3.26 on the preceding page and Lemma 3.10 on page 140, in that order, we have that

$$\begin{aligned}\mathrm{HYB}^{\mathrm{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\mathrm{MultiRound}(\pi),\mathcal{Z}_{|\mathcal{IO}_{\mathrm{MultiRound}}}} &= \mathrm{HYB}^{\mathcal{G}}_{\pi,\mathrm{UnMultiRound}(\mathcal{Z})} \\ &\overset{\mathrm{c}}{\approx} \mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{T},\mathrm{UnMultiRound}(\mathcal{Z})} \\ &= \mathrm{IDEAL}^{\mathrm{MultiRound}^{\mathcal{IO}}(\mathcal{G})}_{\mathrm{MultiRound}^{\mathcal{IO}}(\mathcal{F}),\mathrm{MultiRound}(\mathcal{T}),\mathcal{Z}_{|\mathcal{IO}_{\mathrm{MultiRound}}}},\end{aligned}$$

from which Eq. 3.29 on the preceding page easily follows. □

3.11 Staggered Functionalities

In the previous section we showed how to compile single-round protocols and single-round ideal functionalities into multi-round protocols respectively single-round ideal functionalities and we showed a theorem about the maintenance of security. The motivation was that some single-round ideal functionalities cannot be realized. In this section we take the approach a step further and consider ideal functionalities which in addition to using several rounds might not deliver the output to all parties in the same round. The motivation here is that some functionalities can be realized much more efficiently, both in terms of communication complexity and round complexity, if allowed to deliver the output in different rounds, as compared to when the output must be delivered to all parties in the same round. We call an ideal-function which might deliver the output to different parties in different rounds a staggered termination ideal functionality and say that it has staggered termination. One prominent example is the BA problem, were it is known that any protocol realizing the multi-round version of $\mathcal{F}_{\mathrm{BA}}$ and which always delivers the output to all parties in the same round must have a worst-case round complexity of $t+1$, where t is the number of tolerated corrupted parties, independent of how many parties are actually corrupted. In particular, the protocol will have runs in which all parties were honest and in which $t+1$ rounds were used. If on the other hand the protocol is allowed to deliver the output in different rounds, then there exist protocols with worst-case round complexity $f+2$, where f is the actual number of corrupted parties.[12] This makes a huge difference when only a few parties are actually corrupted, which is typically the case for most real-life system in typical use, as a high corruption level would generally only be observed under some exceptional event as e.g. the presence of an intruder in the network.

3.11.1 Staggered Termination and Staggered Activation Go Hand-In-Hand

The dual of staggered termination is staggered activation, where we allow that for a given activation of the ideal functionality that different parties provide inputs in different rounds.

[12] These results are discussed in much more detail in Chapter 7.

Functionality Stagger($\mathcal{F}, l$)

The functionality Stagger($\mathcal{F}, l$) runs with parties $P_1, \ldots, P_n$ and proceeds as follows:

initialization:
On input (k, r) input (k, r) to $\mathcal{F}$.

party activation:
On input (i, x_i), parse x_i as (b_i, x_i'). If $b_i = 0$, then output (i, x_i) on the SOT; Otherwise, input (i, x_i') to $\mathcal{F}$ to receive a value v on the SOT of $\mathcal{F}$. Then output $(1, v)$ on the SOT.

end round:
On input $(\texttt{activate}, v)$, parse v as $(b, v', S_0, \ldots, S_{l-1})$, where $S_0, \ldots, S_{l-1}$ are disjoint sets for which $H = \cup_{i=0}^{l-1} S_i$. If $b = 0$, then output $(0, \epsilon)$ to all parties.[a] If $b = 1$, then input $(\texttt{activate}, v')$ to $\mathcal{F}$ to receive the outputs $\{y_i\}_{i \in [n]}$. Then for $i = 0, \ldots, l-1$, in round $r + i$ (the current round being indexed r) output $(1, y_i)$ to P_i for all $i \in S_i$ and output $(0, \epsilon)$ to P_i for $i \in H \setminus S_i$.

[a] A value with $b = 0$ is a trivial activation which is not meant to activate $\mathcal{F}$.

Figure 3.37: The staggered version of a synchronous ideal functionality.

When reactive protocols[13] are considered it makes little sense to allow staggered termination if we do not allow staggered activation. The reason is that if the parties have a sequence of activations of the protocol, where the i'th activation depends on the output of the $(i-1)$'th activation, then in the cases where the termination of the $(i-1)$'th activation has staggered termination the i'th *activation* will also be staggered. Unless of course the parties run a protocol to synchronize before the next i'th activation. But as a corollary of the above-mentioned results about BA protocols it can be seen that synchronizing has a worst-case round complexity of $t + 1$, even when no parties are corrupted. Therefore, resynchronizing after each activation would be inefficient and, furthermore, it would make it useless to allow staggered termination as the resynchronization done before each call of the protocol could equally well have been done by the sub-protocol to avoid staggering in the first place. Therefore, staggered termination and staggered activation must go hand-in-hand. We reflect this in the transformation of ideal functionalities by allowing the same staggering gap in activation and termination.

3.11.2 Staggered Functionalities

Given any functionality $\mathcal{F}$ we can define a version of $\mathcal{F}$, called Stagger($\mathcal{F}, l$), which allows parties to input up to l rounds apart and returns outputs such that two honest parties receive their outputs at most l rounds apart.

We define the notion of non-overlapping staggered execution of a synchronous functionality. What we keep in mind is a given functionality $\mathcal{F}$. It receives in each round r from each party P_i a value x_i and delivers back a value y_i to P_i. The non-overlapping staggered

[13] A reactive protocol is one which might be called several times by the parties to solve several instances of the problem that the protocol solves.

execution of $\mathcal{F}$ is a functionality where each synchronous round of $\mathcal{F}$ is stretched into several rounds and where activation and termination of the round is allowed to have a l-round staggering gap. The functionality collects inputs from parties during its own synchronous rounds — again we introduce the notion that an input of the form $(0, \cdot)$ corresponds to no input and $(1, v)$ corresponds to input v. Each time all parties have given a non-trivial input, $\mathcal{F}'$ activates $\mathcal{F}$ on that round of inputs and outputs the values produced to the parties, allowing the adversary to specify in which round and allowing the adversary to specify a l-round staggering. In the rounds where $\mathcal{F}'$ outputs no value, it will formally output a value $(0, \cdot)$, and to output v, it outputs $(1, v)$. We require from the parties that they input the r'th round of inputs to $\mathcal{F}'$ during l consecutive rounds, and that this is done after the $(r-1)$'th round of outputs have been given by $\mathcal{F}'$. We denote this functionality by Stagger($\mathcal{F}$). The details are given in Fig. 3.37 on the next page and the described IO behavior is defined formally below.

Notice that the parties receive output at most l rounds apart. If in particular $l = 0$, then $S_0 = H$ and all parties receive outputs in round r.

Definition 3.28 *The non-overlapping l-round staggering IO behavior $\mathcal{IO}_{\text{stagger}}(l)$ is defined as follows:*

- *For the IO behavior of the honest parties we have the following restrictions: Initially, let $I_i \leftarrow O_i \leftarrow 0$ for all parties. On input (i, x_i), i.e. input x_i to party P_i, parse x_i as (b_i, x_i') for $b_i \in \{0, 1\}$. If $b_i = 1$, then let $I_i \leftarrow I_i + 1$. And in the same way, each time P_i outputs a value of the form $(1, y_i)$, let $O_i \leftarrow O_i + 1$. If ever $I_i > O_i + 1$ for honest P_i, then output* `fail`*. This defines the non-overlapping IO behavior. Furthermore, initially, let $r \leftarrow 0$ and increase r by 1 each time all honest parties received an input, trivial or not. For $I \in \mathbf{N}$, let $r(I)$ be the first round, r, in which some honest party had $I_i = I$. If in the round $r(I) + l$ it holds that $I_j < I$ for some honest party P_j, then output* `fail`*. This defines the l-round staggering IO behavior. Notice that because* MultiRound($\mathcal{F}$) *outputs the bit b_i on the SOT, this IO behavior can be checked given just the values on the SIT and SOT, as required.*

- *For the IO behavior on the ST we have the following restrictions: If in two rounds values of the form $v = (1, v')$ are input on the SIT without all honest parties having output a value of the form $(1, v)$ on the SOT in between these two rounds, then output* `fail`*. Notice that an honest party outputting a value of the form $(1, v)$ means that it got a non-trivial input, see Fig. 3.37, and notice that this IO behavior can be checked given just the values on the SIT and SOT, as required.*

Notice that Stagger($\mathcal{F}$) is actually identical to MultiRound($\mathcal{F}$) from Fig. 3.34 on page 136 except for the staggered termination, and notice that the l-round staggering IO behavior is not essential for Stagger($\mathcal{F}$). As long as all parties have received a new non-trivial input before each non-trivial activation, the execution of $\mathcal{F}$ inside Stagger($\mathcal{F}$) will be sound.[14] Again the IO behavior is specified to make it possible to actually realize Stagger($\mathcal{F}$).

[14] Here, by sound we mean that each honest party is given exactly one input between each activation of the functionality.

Functionality $\mathcal{F}_{\mathrm{ST}}(l)$

The functionality runs with parties $P_1, \ldots, P_n$ and proceeds as follows:

send:

If a party inputs $(stid, \{m_{i,j}\}_{j\in[n]})$ in round r,[a] then let $r_i(stid) = r$, let $s_i(stid) = r_i(stid) + 2(l+1)$ and in round $s_i(stid)$ output $(stid, \{m'_{j,i}\}_{i\in[n]})$ to P_i, where $m'_{j,i}$ is determined as follows:

- If P_j was honest in round $r_i(stid)+l+1$ and input $(stid, \{m_{j,k}\}_{k\in[n]})$ in round $r_j(stid)$ and $|r_j(stid) - r_i(stid)| \leq l$, then let $m'_{j,i} = m_{j,i}$.
- If P_j was honest in round $r_i(stid) + l + 1$ and the above case does not apply, then let $m_{j,i} = \epsilon$.
- If P_j was corrupted in round $r_i(stid)+l+1$ then interpret part of the value on the SIT in round $r_i(stid) + l + 1$ as a message $m_{j,i}$ and let $m'_{j,i} = m_{j,i}$.

incorrect inputs:

If an honest party ever uses the same staggered transmission id *stid* twice, then break down.

[a] The corrupted parties input on the SIT.

Figure 3.38: The Staggered Transmission functionality.

Protocol $\pi_{\mathrm{ST}}(l)$

The protocol runs with parties $P_1, \ldots, P_n$ in the real-life model, and proceeds as follows:

send:

On input $(stid, \{m_{i,j}\}_{j\in[n]})$, let $r_i(stid) = r$, let $s_i(stid) = r_i(stid) + 2(l+1)$ and send $(stid, m_{i,j})$ to P_j. In round $s_i(stid)$ output $(stid, \{m'_{j,i}\}_{i\in[n]})$, where $m'_{j,i}$ is defined as follows: If a message of the form $(stid, m_{j,i})$ was received from P_j in one of the rounds $r_i(stid) - l + 1, \ldots, r_i(stid) + l + 1$, then let $m'_{j,i} = m_{j,i}$. Otherwise, let $m'_{j,i} = \epsilon$.

Figure 3.39: The Staggered Transmission protocol.

3.11.3 Staggered Protocols

We then present the staggered version, Stagger(π), of a protocol π, but first we will present a simple functionality $\mathcal{F}_{\mathrm{ST}}(l)$ from [LLR02b] for sending messages between parties with a known staggering gap l. The functionality is given in Fig. 3.38. A realization of $\mathcal{F}_{\mathrm{ST}}(l)$ is given in Fig. 3.39.

Theorem 3.10 *$\pi_{\mathrm{ST}}(l)$ realizes $\mathcal{F}_{\mathrm{ST}}(l)$.*

Proof. By Theorem 3.6 on page 111 it is enough to prove that the behavior of the protocol $\pi_{\mathrm{ST}}(l)$ is a possible[15] behavior of the ideal functionality $\mathcal{F}_{\mathrm{ST}}(l)$. This is straight-forward to verify: All values of the form $(stid, m_{j,i})$ sent in rounds $r_i(stid) - l, \ldots, r_i(stid) + l$ are

[15] And computable in PPT.

received in rounds $r_i(stid) - l + 1, \ldots, r_i(stid) + l + 1$ and are therefore considered by P_i, exactly as in $\mathcal{F}_{\mathrm{ST}}(t)$.[16] Assume on the other hand that P_i considers a message $(stid, m_{j,i})$. Then this message was sent by an honest party in rounds $r_i(stid) - l, \ldots, r_i(stid) + l$ or sent by the environment no later than in round $r_i(stid)+l+1$.[17]. Therefore such a message would be consider by $\mathcal{F}_{\mathrm{ST}}(l)$ when coming from honest P_j and $\mathcal{F}_{\mathrm{ST}}(l)$ can be forced to consider it from a corrupted P_j by inputting $m_{j,i}$ on the SIT of $\mathcal{F}_{\mathrm{ST}}(l)$ in round $r_i(stid) + l + 1$, were $m_{j,i}$ is known. □

An essential observation, from [LLR02b], about $\mathcal{F}_{\mathrm{ST}}(l)$ is the following: Assume that the parties are synchronized except for an l-round staggering gap and that the parties are simulating a synchronous protocol using $\mathcal{F}_{\mathrm{ST}}(l)$ for message transmission. In detail, assume that party P_i computes round R of the synchronous protocol in round $r_i(R)$, of the staggered protocol. This is done by sending the messages, $m_{i,j,R}$ of round R of the synchronous protocol via $\mathcal{F}_{\mathrm{ST}}(l)$ with $stid = R$. Since by assumption $|r_i(R) - r_j(R)| \leq l$ for all honest parties P_i and P_j the output of $\mathcal{F}_{\mathrm{ST}}(l)$ to P_i in round $s_i(R)$ will contain all the messages $m_{j,i,R}$ sent be honest parties. Therefore P_i can compute the round $(R + 1)$ messages, $m_{i,j,R+1}$, of the synchronous protocol and send these in round $r_i(R + 1) = s_i(R)$. Equally important, let $r_{last}(R) = \max_{i \in H}\{r_i(R)\}_{i \in H}$ and let $r_{first}(R) = \min_{i \in H}\{r_i(R)\}$. Then the last honest party computed round R messages in round $r_{last}(R)$ and the first honest party sent round $(R+1)$ messages in round $r_{first}(R+1)$. By the assumption of at most l-round staggering, we have that $r_{first}(R+1)+l \geq r_{last}(R+1) \geq r_{last}(R)+2(l+1)$, so $r_{first}(R+1) > r_{last}(R)+l+1$. In words, the first message from a honest party in round $R+1$ is sent more than $l+1$ rounds after the last message from a honest party in round R. We say that a round R message is consider in round r if an honest party at some point receives $(R, \{m_{j,i}\})$ from $\mathcal{F}_{\mathrm{ST}}(l)$, where $m_{j,i} \neq \epsilon$ and $(R, m_{j,i})$ was input to $\mathcal{F}_{\mathrm{ST}}(l)$ in round r. We let $c_{last}(R)$ be the last round in which a round R message was considered. Since a party only considers round R messages received no later than in round $r_i(R) + l + 1$ it follows that $r_{last}(R) + l + 1 \geq c_{last}(R)$. In words, the last message of round R is considered by an honest party no later than $l+1$ rounds after the last message of round R was sent by any party, honest or corrupted. Combining the two observations we get that $r_{first}(R + 1) > r_{last}(R) + l + 1 \geq c_{last}(R)$. So, the first message from a honest party in round $R+1$ is sent at least one round after the last message of round R was considered by an honest party. This in particular implies that the environment must decide on all message to send in round R before it sees any message sent in round $R+1$. This is essential, and sufficient, for the security of the synchronous protocol to be maintained.

We are then ready to describe the protocol Stagger(π). Let $\pi = (P_1, \ldots, P_n)$ be a protocol for the $\mathcal{G}$-hybrid model, i.e. P_i is a PPT algorithm

$$(m_{i,1,r}, \ldots, m_{i,n,r}, y_{i,r}, s_{i,r}) = P_i(m_{1,i,r-1}, \ldots, m_{n,i,r-1}, x_{i,r}, t_{i,r-1}) \ ,$$

where $t_{i,r-1}$ is the output from $\mathcal{G}$ in the previous round and $s_{i,r}$ is the input to $\mathcal{G}$ in the next round. The party Stagger(P_i) is given in Fig. 3.40, and we let $\mathrm{Stagger}(\pi, l) = (\mathrm{Stagger}(P_1), \ldots, \mathrm{Stagger}(P_n))[\pi_{\mathrm{ST}}(l)/\mathcal{F}_{\mathrm{ST}}(l)]$.

[16] The environment might have sent messages from corrupted parties as late as in round $r_i(stid) + l + 1$ as it is rushing. Still these messages are consider by P_i.

[17] And it might have been sent that late as the environment is rushing

Party Stagger(P_i)

The algorithm specifying party Stagger(P_i) can be considered an ITM saving its state in $m_{i,i,r}$. We describe the party Stagger(P_i) using ITM terminology. The party is for the $(\mathcal{F}_{\mathrm{ST}}(l), \mathrm{Stagger}(\mathcal{G}))$-hybrid model and proceeds as follows.

1. Receive (k, r_i). Let $R \leftarrow 0$ be a round counter. Let $m_{j,i,0} = \epsilon$ for $j \in [n]$, let $m_{i,i,0} = (k, r_i)$ and let $t_{i,0} = \epsilon$. Wait until the first round where $\mathcal{Z}$ inputs a value of the form $(1, x_{i,1})$. In all rounds send ϵ to all parties — all communication will take place over $\mathcal{F}_{\mathrm{ST}}(l)$.
2. Let $R \leftarrow R + 1$. Then run

$$(\{m_{i,j,R}\}_{j\in[n]}, y_{i,R}, s_{i,R}) = P_i(\{m_{j,i,R-1}\}_{j\in[n]}, x_{i,R}, t_{i,R-1})$$

   and input $(R, \{m_{i,j,R}\}_{j\in[n]})$ to $\mathcal{F}_{\mathrm{ST}}(l)$ and input $(1, s_{i,R})$ to Stagger($\mathcal{G}$).
3. Then wait until the first round where $\mathcal{F}_{\mathrm{ST}}(l)$ have output $(R, \{m_{j,i,R}\}_{j\in[n]})$ and Stagger($\mathcal{G}$) have output a message of the form $(1, t_{i,R})$. In the rounds until both events have taken place output $\perp$[a] to $\mathcal{Z}$ and input $\perp$ to Stagger($\mathcal{G}$), and if $\mathcal{Z}$ inputs anything in these rounds, then simply ignore. When $(R, \{m_{j,i,R}\}_{j\in[n]})$ and $t_{i,R}$ have been received, output $(1, y_{i,R})$ to $\mathcal{Z}$, wait until the first round where $\mathcal{Z}$ inputs a value of the form $(1, x_{i,R+1})$, and goto Step 2.

[a] We use the notation $\perp = (0, \epsilon)$.

Figure 3.40: Stagger(P_i).

3.11.4 Staggered IO Behaviors

As for an ideal functionality and a protocol we can define a staggered variant of an IO behavior $\mathcal{IO}$. We can consider the IO behavior Stagger($\mathcal{IO}$), which can be defined via Fig. 3.37 on page 144, by considering $\mathcal{IO}$ an ITM. We let $\mathcal{IO}' = \mathrm{Stagger}(\mathcal{IO})$ denote the construction in Fig. 3.37 on page 144 applied to $\mathcal{IO}$ — with the one difference that in initialization only k is given to an IO behavior — and we then define $\mathrm{Stagger}(\mathcal{IO}, l) = \mathcal{IO}' \wedge \mathcal{IO}_{\mathrm{stagger}}(l)$. Intuitively this means that $\mathrm{Stagger}(\mathcal{IO}, l)$ requires that a given IO sequence IO is according to $\mathcal{IO}_{\mathrm{stagger}}(l)$, which allows to defined a sound IO sequence which $\mathcal{F}$ would see if Stagger($\mathcal{F}$) saw IO, and Stagger($\mathcal{IO}$) then requires that this pruned IO sequence is in accordance with $\mathcal{IO}$.

Definition 3.29 $\mathrm{Stagger}^{\mathcal{IO}}(\langle \mathcal{F}, \mathcal{IO} \rangle, l) = \langle \mathrm{Stagger}(\mathcal{F}), \mathrm{Stagger}(\mathcal{IO}, l) \rangle$.

3.11.5 The Staggered Compilation Theorem

We will now prove that if π realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, then $\mathrm{Stagger}(\pi, l)$ realizes $\mathrm{Stagger}^{\mathcal{IO}}(\mathcal{F}, l)$ in the $\mathrm{Stagger}^{\mathcal{IO}}(\mathcal{G}, l)$-hybrid model. We will also relate the communication complexity (the total number of bits sent over the point-to-point lines) and the round complexity of the two protocols. Since the environment might refuse to let $\mathrm{Stagger}^{\mathcal{IO}}(\mathcal{G}, l)$ ever terminate and might never input the next round of non-trivial inputs, we will not count the rounds where all parties received messages from all parties for the round they are simulating

and are only waiting for outputs from $\text{Stagger}^{\mathcal{IO}}(\mathcal{G}, l)$ or the next round of non-trivial inputs. These rounds are completely inactive and we therefore call this notion of round complexity the active-round complexity.

Lemma 3.11 *If π realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, then* $\text{Stagger}(\pi, l)$ *realizes* $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}, l)$ *in the* $\text{Stagger}^{\mathcal{IO}}(\mathcal{G}, l)$*-hybrid model. Furthermore, the total number of bits communicated by* $\text{Stagger}(\pi, l)$ *on a given sequence of non-trivial inputs and random bits $(r_1, \ldots, r_n)$ for the parties $(P'_1, \ldots, P'_n)$ is the same as for π on the same sequence of inputs and random bits $(r_1, \ldots, r_n)$ for the parties $(P_1, \ldots, P_n)$ and the active-round complexity is the same up to a constant factor $3(l+1)$.*

Proof. Using that the parties are at most l rounds apart and that the last activated party uses $2(l+1)$ rounds to simulate one round of π, the claims about the complexity is easy to verify. By Theorem 3.5 on page 100 and Theorem 3.10 on page 145 it is therefore enough to prove that $(\text{Stagger}(P_1), \ldots, \text{Stagger}(P_n))$ realizes $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}, l)$ in the $(\mathcal{F}_{\text{ST}}(l), \text{Stagger}^{\mathcal{IO}}(\mathcal{G}, l))$-hybrid model. The proof follows the proof of Theorem 3.9 on page 141. The most important difference between $\text{Stagger}(P_i)$ and $\text{MultiRound}(P_i)$ is that $\text{Stagger}(P_i)$ can be l-rounds away from the other parties $\text{Stagger}(P_j)$ in the protocol, as it starts executing on the first non-trivial input from $\mathcal{Z}$, which is allowed to arrive with l-round staggering according to $\mathcal{IO}_{\text{stagger}}(l)$. We have however argued above that when the messages of the synchronous protocol is sent using $\mathcal{F}_{\text{ST}}(l)$, then a l-round staggering can be tolerated. That the construction of $\text{Stagger}(\pi)$ is sound therefore follows using a simple inductive argument that the parties always have at most l-round staggering unless $\mathcal{Z}$ violated $\mathcal{IO}_{\text{stagger}}(l)$. Assume that for a given value of R the parties reach Step 2 in Fig. 3.40 on the page before with at most l-round staggering, the basis of the induction being $R = 1$. Then they call $\mathcal{F}_{\text{ST}}(l)$ and $\text{Stagger}^{\mathcal{IO}}(\mathcal{G}, l)$. Let $r_{first}(R)$ ($r_{last}(R)$) be the first (last) round where a party received outputs from both $\mathcal{F}_{\text{ST}}(l)$ and $\text{Stagger}^{\mathcal{IO}}(\mathcal{G}, l)$; Let $r_{first,\mathcal{G}}(R)$ ($r_{last,\mathcal{G}}(R)$) be the first (last) round where a party received output from $\text{Stagger}^{\mathcal{IO}}(\mathcal{G}, l)$; And let $r_{first,\mathcal{F}_{\text{ST}}}(R)$ ($r_{last,\mathcal{F}_{\text{ST}}}(R)$) be the first (last) round where a party received output from $\mathcal{F}_{\text{ST}}(l)$; Because $\text{Stagger}^{\mathcal{IO}}(\mathcal{G}, l)$ and $\mathcal{F}_{\text{ST}}(l)$ maintain l-round staggering we have that $r_{last,\mathcal{G}}(R) \leq r_{first,\mathcal{G}}(R) + l$ and $r_{last,\mathcal{F}_{\text{ST}}}(R) \leq r_{first,\mathcal{F}_{\text{ST}}}(R)$. Since a party received output from both $\mathcal{G}$ and $\mathcal{F}_{\text{ST}}(l)$ in round $r_{first}(R)$ we have that $\max\{r_{first,\mathcal{G}}(R), r_{first,\mathcal{F}_{\text{ST}}}(R)\} \leq r_{first}(R)$. Since a party received a value from $\mathcal{G}$ or $\mathcal{F}_{\text{ST}}(l)$ in round $r_{last}(R)$ we have that $r_{last}(R) = \max\{r_{last,\mathcal{G}}(R), r_{last,\mathcal{F}_{\text{ST}}}(R)\}$. Combining these observations we get that $r_{last}(R) = \max\{r_{last,\mathcal{G}}(R), r_{last,\mathcal{F}_{\text{ST}}}(R)\} \leq \max\{r_{first,\mathcal{G}}(R) + l, r_{first,\mathcal{F}_{\text{ST}}}(R) + l\} = \max\{r_{first,\mathcal{G}}(R), r_{first,\mathcal{F}_{\text{ST}}}(R)\} + l \leq r_{first}(R) + l$. Therefore the parties output $(1, y_{i,r})$ to $\mathcal{Z}$ with l-round stagger, so the parties are not the first to violate $\mathcal{IO}_{\text{stagger}}(l)$ and $\mathcal{Z}$ must input the next round of non-trivial inputs with l-round stagger to not violate $\mathcal{IO}_{\text{stagger}}(l)$. Therefore the parties go to Step 2 with at most l-round stagger, which establishes the induction step. □

We then consider the case with several ideal functionalities. For this purpose we prove a lemma.

Lemma 3.12 *There exists a protocol γ realizing* $\text{Stagger}^{\mathcal{IO}}([\mathcal{G}_1, \ldots, \mathcal{G}_m], l)$ *(with no communication) in the* $[\text{Stagger}^{\mathcal{IO}}(\mathcal{G}_1, l), \ldots, \text{Stagger}^{\mathcal{IO}}(\mathcal{G}_m, l)]$*-hybrid model.*

Proof. When a party in the protocol γ is given an input $(x_1, \ldots, x_m)$ for the ideal functionality $\text{Stagger}^{\mathcal{IO}}([\mathcal{G}_1, \ldots, \mathcal{G}_m], l)$ it inputs $(x_1, \ldots, x_m)$ to $[\text{Stagger}^{\mathcal{IO}}(\mathcal{G}_1, l), \ldots, \text{Stagger}^{\mathcal{IO}}(\mathcal{G}_m, l)]$. Then it waits until it received outputs y_i from all $\text{Stagger}^{\mathcal{IO}}(\mathcal{G}_i, l)$. In the round where the last y_i was received it outputs $(y_1, \ldots, y_m)$. The security of this protocol is a simple consequence of the fact that parallel composition of l-stagger protocols yield l-stagger protocols. Let r_{last} be the last round where a party outputs. Then some $\text{Stagger}^{\mathcal{IO}}(\mathcal{G}_i, l)$ output in that round. Therefore all other parties received their outputs from $\text{Stagger}^{\mathcal{IO}}(\mathcal{G}_i, l)$ no earlier than in round $r_{last} - l$. So, no party output earlier than in round $r - l$, which proves that the l-round staggering is maintained. □

Lemma 3.13 *If π realizes $\mathcal{F}$ in the $(\mathcal{G}_1, \ldots, \mathcal{G}_m)$-hybrid model, then there exists a protocol π' realizing $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}, l)$ in the $(\text{Stagger}^{\mathcal{IO}}(\mathcal{G}_1, l), \ldots, \text{Stagger}^{\mathcal{IO}}(\mathcal{G}_m, l))$-hybrid model. Furthermore, the total number of bits communicated by π' on a given sequence of non-trivial inputs and random bits $(r_1, \ldots, r_n)$ for the parties $(P'_1, \ldots, P'_n)$ is the same as for π on the same sequence of inputs and random bits $(r_1, \ldots, r_n)$ for the parties $(P_1, \ldots, P_n)$ and the active-round complexity is the same up to a constant factor $3(l + 1)$.*

Proof. The claims for the complexity follows as for Lemma 3.11 on the facing page and the rest of the claim follows from Lemmas 3.11 and 3.12 using the composition theorem. □

We consider a final extension of the result to consider protocols with initializing functionalities. Formally a protocol for the $(\mathcal{G}_1, \ldots, \mathcal{G}_m)$-hybrid model with initializing functionality $\mathcal{I}$ is just a protocol for the $(\mathcal{G}_1, \ldots, \mathcal{G}_m, \mathcal{I})$-hybrid model, so we know how to compile it into the $(\text{Stagger}^{\mathcal{IO}}(\mathcal{G}_1, l), \ldots, \text{Stagger}^{\mathcal{IO}}(\mathcal{G}_m, l), \text{Stagger}^{\mathcal{IO}}(\mathcal{I}, l))$-hybrid model. We can however do better and compile to the $(\text{Stagger}^{\mathcal{IO}}(\mathcal{G}_1, l), \ldots, \text{Stagger}^{\mathcal{IO}}(\mathcal{G}_m, l), \mathcal{I}))$-hybrid model. The reason is that $\mathcal{I}$ is only called in the very first round, which is still guaranteed to be synchronous. Consider in particular the protocol where the parties simultaneously input `init` to $\mathcal{I}$ in the very first activation of the parties (the round where k and r_i is supplied to P_i) and then waits for $\mathcal{I}$ to output a value, at which point they start running the protocol of Lemma 3.13. Since $\mathcal{I}$, by definition of an initializing functionality, delivers outputs to all honest parties in the same round (which we called the actual start of the protocol), it is straight-forward to verify that:

Theorem 3.11 *If π realizes $\mathcal{F}$ in the $(\mathcal{G}_1, \ldots, \mathcal{G}_m)$-hybrid model with initializing functionality $\mathcal{I}$, then there exists a protocol π' realizing $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}, l)$ in the $(\text{Stagger}^{\mathcal{IO}}(\mathcal{G}_1, l), \ldots, \text{Stagger}^{\mathcal{IO}}(\mathcal{G}_m, l))$-hybrid model with initializing functionality $\mathcal{I}$. Furthermore, the total number of bits communicated by π' on a given sequence of non-trivial inputs and random bits $(r_1, \ldots, r_n)$ for the parties $(P'_1, \ldots, P'_n)$ is the same as for π on the same sequence of inputs and random bits $(r_1, \ldots, r_n)$ for the parties $(P_1, \ldots, P_n)$ and the active-round complexity is the same up to a constant factor $3(l + 1)$.*

Part II

Basic Protocols

CHAPTER 4

The Secure-Channels Model

I don't like to commit myself about heaven and hell – you see, I have friends in both places.
— Mark Twain

4.1 Introduction

One way of constructing a secure protocol for the cryptographic model is to take a protocol which is secure in the secure-channels model, where secure channels are assumed, and then compile this protocol for the cryptographic model by adding encryption to the channels. This can significantly simplify the design process as the underlying protocol can be designed under the assumption that the adversary does not see the communication between honest parties. As it turns out, the advantage of using the secure-channels model is greatest when adaptive security is considered as a number of technical problems regarding invertible sampling vanishes in the secure channels model. When adapting an adaptively secure secure-channels protocol for the cryptographic model, the goal is to replace the secure channels of the by open channels using non-committing encryption (NCE), which by definition is an adaptively secure protocol for encrypting communication in a multiparty setting, i.e. an adaptively secure realization of the secure message transmission functionality $\mathcal{F}_{\text{SMT}}$. In this chapter we look at realizations of $\mathcal{F}_{\text{SMT}}$ and we look at a lower bounds on the round complexity of such realizations.

4.1.1 Non-Interactive NCE is Impossible

To realize $\mathcal{F}_{\text{SMT}}$, assume that we let each party P_i have a private key for a public-key encryption scheme, secure against adaptively chosen ciphertext attack, where the public key pk_i is known by all other parties. If we encrypt all communication to P_i under pk_i (including in the messages the identity of the sender), then we will have a statically secure implementation, see [Can01b]. However, no encryption scheme exists for which this protocol is *adaptively* secure. This follows from a general result that no non-interactive communication protocol is adaptively secure — throughout the dissertation we will let non-interactive communication protocol denote a communication protocol with the property that after a pre-processing

phase which might involve interaction (e.g. the receiver sending a public key to the sender), the sender can send an unbounded number of bits to the receiver without there being any communication from the receiver to the sender. The result that non-interactive NCE is impossible appears to have been folklore for some time; A formalization and a proof first appeared in [Nie02a]. We note that the impossibility result holds in much weaker models than the one from Chapter 3. E.g., even if we give the parties the possibility of reliably erasing their internal state, still non-interactive NCE is impossible. We will prove this result in Section 4.3. This result have been used by Canetti, Halevi and Katz [CHK03] to argue the necessity of key-evolving in non-interactive forward-secure public-key encryption.

4.1.2 Interactive NCE is Tricky

The above impossibility result tells us that any non-committing encryption protocol must be interactive. So, let us consider a two-move protocol. Here the receiver P_R of the $\mathcal{F}_{\text{SMT}}$ functionality sends the first message and the sender P_S of the $\mathcal{F}_{\text{SMT}}$ functionality sends the second message. Using the terminology of public-key encryption schemes, we can think of the first message as a public-key pk, of which the receiver of the $\mathcal{F}_{\text{SMT}}$ functionality knows the secret key sk, and the second message as an encryption $c \leftarrow E_{pk}(m)$ of the message m to be sent. This protocol is identical to the solution sketched above, except that the receiver can now generate a new key-pair (pk, sk) for each transfer. Small as the difference might seem, it turns out that this approach allows to realize $\mathcal{F}_{\text{SMT}}$ w.r.t. adaptive adversaries. The solution however turns out to be more involved than one might expect — no standard encryption scheme is known for which this protocol is adaptively secure.

To see what the problem with standard encryption schemes is, consider the environment $\mathcal{Z}$ which activates P_S with an arbitrary message m and then corrupts no party but just waits for P_R to send pk to P_S and for P_S to send a ciphertext c to P_R. Then the environment corrupts P_R (before c arrives) to learn sk (in general sk just denotes an internal state of P_R which would allow to decipher c). This is possible even in a model where the parties can reliably erase data, as P_R must keep the value of sk at least until c arrives. By the security of the encryption scheme the environment will observe that $m = D_{sk}(c)$ except possibly with negligible probability. Now, by the definition of security there should exist a simulator $\mathcal{S}$ such that the simulator executed in the ideal process for $\mathcal{F}_{\text{SMT}}$ with the same environment $\mathcal{Z}$ produces an output indistinguishable from what $\mathcal{Z}$ just observed. But, in the ideal process the simulator does not see m during the execution as long as both parties are uncorrupted, and the simulator must therefore generate c given just $|m|$. Then on the corruption of P_R, the simulator sees m and computes sk to give to the environment. Since the definition of security requires that the environment cannot tell the difference between the real-life execution and the simulation it follows that $m = D_{sk}(c)$ except with negligible probability. Since no secret key can make c decrypt to two different values, both with probability close to 1, this means that there exists an injective map from messages m to secret keys $sk_{c,m}$ for which $m = D_{sk_{c,m}}(c)$. Such a map does not exist for standard encryption schemes which typically has unique decryption given the public-key pk, which was already shown to the environment. Another consequence of the existence of this map, and its being injective, is that the length of the messages sent under a fixed value of sk cannot exceed the length of

sk — this is the clue to the impossibility result for non-interactive NCE mentioned above.

4.1.3 NCE for the Erasure Model

These issues were first solved by Beaver and Haber in [BH92]. Their solution is to set up the protocol such that P_R gets the opportunity of *erasing* sk before the encryption of m arrives. In their protocol P_R sends to P_S the public key pk. Then P_S generates a uniformly random message p and sends $c = E_{pk}(p)$ to P_R. Then P_R computes $p = D_{sk}(c)$ *and erases everything except* p. When m later becomes known to P_R he computes $c' = p \oplus m$, where $\oplus$ denotes bitwise xor, and sends c' to P_R who can then compute $m = p \oplus c'$. Since at no point P_R knows both sk and the encryption c' of m, the attacker cannot obtain both, which preempts the problem that for fixed c' there should be an injective map from sk to messages m. Since sk is deleted a new key-pair must be generated for each message — or each time P_S has sent a total of $|p|$ bits. This means that the scheme is interactive.

4.1.4 NCE for the Non-Erasure Model

The solution described above depends essentially on the use of erasure. However, in many settings, trusting the parties to be able to erase parts of their state might be unrealistic, due to e.g. physical limitations on erasure and weak operating systems. The first adaptively secure realization of $\mathcal{F}_{\text{SMT}}$ in the non-erasure model is by Canetti Canetti, Feige, Goldreich and Naor [CFGN96] who coin the term non-committing encryption scheme. They construct a solution based on what they call common domain trapdoor systems, which is basically a trapdoor system, where one given a permutation can sample a uniformly random permutation with the same domain (along with its trapdoor). They show how to build non-committing encryption based on the RSA and the DDH assumption. Their scheme has expansion factor $\Omega(k^2)$, i.e. the communication complexity of their scheme is $\Omega(k^2)$ bits per plaintext bit to be communicated. The protocols based on the RSA and DH assumptions are the most efficient in terms of rounds. They are both two-round protocols, which is optimal. In the first round the receiver sends a public key and in the second round the sender sends an encryption of the message under the public key. The protocol based on common-domain trapdoor systems uses an interactive protocol to set up the public keys and uses three more rounds. Later Beaver [Bea97] proposed a simpler and more efficient scheme based on the DDH assumption. His scheme has expansion factor $\Theta(k)$ and is a three-round protocol.

4.1.4.1 Oblivious Instance Generation

An important tool in both protocols is to be able to generate instances of hard problems in two ways. The first method generates $(x, s) \leftarrow gen(r)$, where x is a random instance and s is a solution. The second method generates $x \leftarrow \widetilde{gen}(r)$, where x is a random instance and the random bits used by $\widetilde{\mathcal{G}}(r)$ will not help in finding the solution to x.[1] If the random bits used by $\widetilde{gen}$ does not help in finding a solution to the generated random instance we call $\widetilde{gen}$ an oblivious generator. We will define this notion formally later.

[1] Therefore letting $\widetilde{gen}$ work by first running gen and then outputting just x will not do as the random bits used by $\widetilde{gen}$ to run gen allows to find a solution s to x by rerunning gen.

The usefulness of oblivious generation is in general that if we in the protocol specifies that the parties should generate a random instance obliviously, then in the simulation the simulator is free to generate the instance using the normal generator. If the honest party is later corrupted the simulator hands to the adversary only the instance, which is what the adversary would have seen in a real execution. This allows for the simulator to run the honest parties in a semi-honest way, while at the same time learning solutions to hard instances, which the adversary will have a hard time computing given just the instances. This might potentially give the simulator the necessary advantage that allows it to cheat the adversary. Oblivious generation has the same effect as when the protocol generates the instances along with a solution, but *erases* the solution right after the generation. Also there will the simulator get the solution while the adversary will not. In this sense oblivious generation replaces erasure.

In [CFGN96] the normal generation is the generation of a trapdoor permutation along with its trapdoor. The oblivious generation generates a permutation without learning the corresponding trapdoor. In [Bea97] the normal generation generates a pair (g^a, a), where g is a fixed generator of the quadratic residues in $\mathbf{Z}_p$ for an appropriate prime p. Here $x = g^a$ is the instance and the solution is the g-logarithm of x. The oblivious generation then generates x by drawing a uniformly random quadratic residue.

4.1.4.2 Sketch of the NCE Protocols

The protocols [Bea97, CFGN96] have some common structure. They consider the situation, where one bit is sent. The sender picks a bit c, generates two instances x_0 and x_1 such that x_c is generated along with a solution and such that x_{1-c} is generated obliviously. The bit c is in some sense the bit to be sent: In [CFGN96] it is set to be the bit to be sent and in [Bea97] it is picked uniformly random and later used to one-time-pad encrypt the bit to be sent. The simulators ability to learn both solutions and therefore not decide on c until the corruption takes place is what allows for the simulation.

In fact one of the generators in [CFGN96] is a sub-protocol having the same effect as the above described oblivious generators. This protocol is used to realize $\mathcal{F}_{\text{SMT}}$ assuming only trapdoor permutations. The goal is to let a party R (who is to receive a message from S) generate two permutations f_0 and f_1 such that he learns the trapdoor of f_c and does not know the trapdoor of f_{1-c} (the permutations are to be used by S as the public key of R). The protocol is an n-party protocol with $n > 2$ and is secure as long as one party remains honest. First each party P_i generates two permutations along with their trapdoors, (g_0^i, t_0^i) and (g_1^i, t_1^i). All permutations are generated to have the same domain. Then all parties send (g_0^i, g_1^i) to R, which by the common domain assumption can define two permutations $f_0 = g_0^1 \circ g_0^2 \circ \cdots \circ g_0^n$ and $f_1 = g_1^1 \circ g_1^2 \circ \cdots \circ g_1^n$. Now R should learn the trapdoor of f_c without learning the trapdoor of f_{1-c}. He does this by engaging in a 1-out-of-2-OT with each of the other parties — in an (1-out-of-2) oblivious transfer (OT) the sender inputs two values v_0 and v_1, the receiver inputs a bit c, the sender learns nothing about c and the receiver only learns v_c. The parties offer t_0^i and t_1^i and R chooses to learn t_c^i. We can assume that along with the trapdoors are sent the random bits used to generate them such that R can validate their correctness. Let $A \subset \{1, \ldots, n\}$ denote the subset corresponding to correct trapdoors.

Then R composes f_i of g_i^j for $j \in A$ and sends A as part of the public key to S. Obviously, if just one trapdoor t_i^j of the composed trapdoor t_i is unknown to a party, then f_i cannot be inverted by that party. Therefore the protocol achieves the desired goals as long as just one party remains honest. The OT-scheme used is actually only secure against the sender learning the bit c. It is such that the cheating receiver can learn both trapdoors without being detected. This gives a simulator the ability to learn both trapdoors as in the case where an oblivious generator had been used. Thereby the protocol fully replaces the role of oblivious generation.

4.1.5 Some New NCE Schemes

In Section 4.4 we present a new non-committing encryption scheme for the non-erasure model. It can be based on any simulatable public-key encryption scheme, which is an ordinary IND-CPA secure public-key encryption scheme with some additional oblivious generation properties. More precisely it should be possible to generate a public key without learning the corresponding private key and it should be possible to generate a ciphertext without learning the corresponding plaintext. The scheme is a three-round protocol and in the expectation communicates 4 public keys and 4 encryptions per plaintext bit. Based on encryption schemes with key and message sizes linear in k this gives an expansion factor of $\Theta(k)$ as for the DDH scheme in [Bea97].

We provide two examples of non-committing encryption schemes, based on the DDH assumption and the RSA assumption. Our protocol instantiated with the DDH scheme is similar to the DDH scheme from [Bea97] and our general protocol may be viewed as a generalization of Beaver's DDH scheme to the more general assumption of existence of IND-CPA secure encryption schemes with oblivious generators. Our protocol instantiated with the RSA scheme is completely new. Its improvement over the previously best known RSA scheme from [CFGN96] is by a factor of k in the expansion factor.

The RSA simulatable encryption scheme is build using the following general methodology. Call a trapdoor permutation system simulatable if the permutations can be obliviously generated and an element in the domain can be invertibly sampled — recall that this informally means that an element can be sampled in a way such that the random bits used to sample it (or similarly distributed bits) can be reconstructed from the element itself. We prove that if one starts from a simulatable trapdoor system and applies the construction of IND-CPA secure public-key encryption schemes by Blum and Goldwasser [BG85], then one gets a simulatable encryption scheme. We then construct a simulatable trapdoor system based on the RSA assumption.

Finally we prove that if one has a trapdoor system with the only extra condition that the domains can be invertibly sampled, then one can still build non-committing encryption. This is done by replacing the oblivious key-generation in the scheme based on simulatable trapdoor systems by the key-generation protocol from [CFGN96] described above. In doing this we identify an issue in the key-generation protocol from [CFGN96], which makes it insecure. We correct this issue using a technique which is of interest in itself as it provides an example that rewinding zero-knowledge proofs can be used in the UC framework. We also apply a small twist to the key-generation protocol to get rid of the common-domain assumption. The exis-

tence of trapdoor systems with invertible domain sampling is the weakest general assumption under which non-committing encryption is known to exist; the scheme in [CFGN96], besides the common domain property, also needs that the domains have invertible sampling.

No introduction to non-committing encryption is complete without mentioning the notion of a deniable encryption scheme [CDNO97] introduced by Canetti, Dwork, Naor and Ostrovsky. Basically a deniable encryption scheme allows e.g. the encryptor to compute an encryption $c = E_{pk}(m; r)$ and send it to the decryptor which is capable of retrieving m. However, if the sender is ever asked to reveal which message he sent, he can for some alternative message m' compute random bits r' such that $c = E_{pk}(m'; r')$ and such that it really looks as if c was an encryption of m'. This concept is very similar to that of non-committing encryption, but observe that while a deniable encryption scheme allows e.g. the sender to lie *in the real world* a non-committing encryption scheme is only required to allow the sender to do this *in the simulation.*

4.1.6 The Rest of the Chapter

The rest of chapter contains five sections. In Section 4.2 we will start by constructing non-committing encryption for the RO model. With respect to the flow of the presentation , the main reason for this section is that the construction in the RO model is very simple and so exemplifies the 'non-committing' property of an encryption scheme elegantly. Therefore Section 4.2 provides some intuition before Section 4.4, where we show how to construct non-committing encryption in the CT model. The non-committing encryption scheme in Section 4.2 is non-interactive, so in addition, along with Section 4.3 were it is proved that non-interactive non-committing encryption is impossible in the NPRO model, the section establishes the separation result discussed in Section 3.9. In Section 4.4 we introduce the notion of simulatable public-key encryption scheme and show how to construct non-committing encryption given an instantiation of this notion. In Section 4.5 we then show how to construct simulatable public-key encryption scheme based on the DDH assumption and the RSA assumption. Finally, in section Section 4.6 we show how to construct non-committing encryption based on any family of trapdoor permutations, where the domain sampler has invertible sampling.

4.2 Possibility of Non-Interactive NCE in the RO Model

In this section we prove that if trapdoor permutations exists, then non-interactive NCE exists in the PRO model, where non-interactive non-committing encryption is defined as follows:

Definition 4.1 *Consider a version $\mathcal{F}'_{\mathrm{SMT}}$ of $\mathcal{F}_{\mathrm{SMT}}$, where we restrict the functionality to two parties P_1 and P_2, add a parameter $s \in \mathbf{N}$ being the initialization round complexity, and add the command:*

initialize:

> *On input* `init` *to either P_1 or P_2 output* `init` *on the SOT and after s rounds output* `ready` *to P_1 and P_2. Until having output* `ready` *ignore all other inputs than the first* `init` *value.*

A two-party non-interactive non-committing encryption protocol is a $\mathcal{G}$-hybrid protocol $\pi = (P_1, P_2)$ realizing $\mathcal{F}'_{\mathrm{SMT}}$ of the following form: The communication pattern between P_1 and P_2 can be arbitrary in the s rounds following the first `init` *input to either of the parties, and the parties might access $\mathcal{G}$ in these s rounds. But, after P_1 and P_2 have output* `ready`*, then on each activation of the form (mid, P_j, m), P_i sends one message (mid, c) to P_j, which when receiving (mid, c) outputs (mid, P_i, m') for some $m' \in \{0,1\}^*$; I.e. the round complexity for transmission is $r = 1$. Furthermore, no other messages are sent after having output* `ready`*. I.e. a party P_j only sends a message to P_i if activated on some value (mid, P_i, m) and the parties are specified such that they ignore all messages from $\mathcal{G}$ after having output* `ready`*.*

Remark 4.1 *Notice that in specifying that the protocol is non-interactive we required that it only uses one round. Alternatively we could just have required that the communication is uni-directional, i.e. that there is only messages from P_i to P_j, and have disregarded the round complexity. The reason why we do not do this is that we want the definition to be independent of the whether we consider synchronous or asynchronous protocols, and whereas uni-directional asynchronous protocols, with respect to security, are equivalent to one-round uni-directional asynchronous protocols, for synchronous protocols already a two-round uni-directional protocol should be considered interactive.*

To see why this is the case, consider first asynchronous protocols. In the asynchronous model any secure uni-directional protocol can be transformed into a secure uni-directional one-round protocol by having P_i collect all the messages it wants to send to P_j into one message and then send it to P_j when all communication has been generated. Therefore in the asynchronous model it is no restriction to consider only one-round protocols in the definition. To see why the above transformation is secure, notice that even when P_i sends the messages in different rounds, the environment might decide to collect the message and deliver them in the same round.[2] *Therefore the communication pattern of the transformed protocol is a possible communication pattern of the original protocol. So, any attack against the transformed protocol is also an attack against the original protocol. This transformation does not work for synchronous protocols, as the communication pattern of the transformed protocol is not guaranteed to be a communication complexity of the original protocol. Indeed we will see that no such transformation exists, as NCE can be realized by a two-round uni-directional synchronous protocol but not by a non-interactive synchronous protocol (i.e. a one-round uni-directional protocol).*

To see why two-move uni-directional synchronous protocols should be considered interactive, we relate them to asynchronous protocols. Any secure r-round uni-directional synchronous protocol can be transformed into a secure $2r-1$ asynchronous protocol as follows: Assume that only P_i sends messages to P_j. In the transformed protocol, let P_j send an acknowledge message to P_i after each of the $r-1$ first messages and let P_i send message R only after having received $R-1$ acknowledgments. An attack on the transformed protocol can easily be transformed into an attack on the synchronous protocol by letting the attacker simulate the acknowledgment on its own. The above transformation takes any r-round uni-directional protocol into a bi-directional protocol when $r > 1$. This is of course not the same a saying that any secure transformation must do so. However, since two-round uni-directional syn-

[2] In the asynchronous model, the environment has the power to decide when to deliver a given message.

chronous protocols are stronger than one-round uni-directional synchronous protocols, any security preserving transformation of two-round uni-directional synchronous protocols into asynchronous protocols must generate some bi-directional protocols.[3] *Therefore the extra round corresponds to interaction when the synchronous model is realized in the asynchronous model. All in all, the model-independent definition of non-interactive protocols is to consider one-round uni-directional protocols.*

4.2.1 The Intuition

Our protocol is reminiscent of a construction of chosen ciphertext secure encryption in [BR93]. In the pre-processing phase the receiver P_R sends a description f of a trapdoor permutation to the sender P_S. Each message from P_S to P_R is transmitted as $(f(x), H(x) \oplus m)$, where x is a uniformly random element in the domain of f and H is the uniformly random oracle. To prove the scheme secure we construct a simulator $\mathcal{S}$. The simulator works by running internally a copy of the protocol. It tries to make the internal protocol consistent with the values of m input to the ideal process (knowing only the length of m). The simulation proceeds as follows: Run the pre-processing as in the real-life model. Then simulate the message transfers as follows: The parties of the protocol will request to evaluate the RO H on some values x. To simulate the RO, we return a uniformly random element r — if queried on the same x twice, we return the same r. To simulate the sending of m we generate random x and send $(f(x), H(x) \oplus 0^l)$, where l is the length of m and 0^l is the all-zero string of length l — if the simulated oracle H was not defined on x we set it to be a uniformly random element r as above. Assume that after simulating the sending of an arbitrary number of message the environment corrupts P_R. The simulator then receives from the ideal evaluation, for each $(f(x), H(x) \oplus 0^l)$ sent in the simulation, the real value m which should have been sent, and the simulator must then come up with an internal state of P_R consistent with m. Assume that the simulator defined $H(x) = r$, i.e. that $(f(x), r)$ was the message sent. The simulator then simulates by simply claiming that $H(x) = m \oplus r$. This is a perfect simulation as we get that $H(x)$ is uniformly random and $(f(x), r) = (f(x), (r \oplus m) \oplus m) = (f(x), H(x) \oplus m)$. However, there are two ways this simulation can fail. First of all, if the same x was used twice the simulator might be in the situation that it needs to define $H(x)$ to both $r \oplus m_1$ and $r \oplus m_2$ for $m_1 \neq m_2$. However, this happens with negligible probability as the x's are chosen uniformly at random by the simulator. Second, it might be that the environment queried H on x and therefore knows that $H(x)$ was defined to r, which *commits* the simulator to this choice and makes the simulation fail. However, if the environment queried H on x the environment intuitively had to invert the trapdoor function f on a uniformly random element: $\mathcal{Z}$ returned x given only $f(x)$. This would contradict the hardness of inverting f on random elements.

Intuitively we obtained the non-committing property as follows: The simulator is not required to define $H(x)$ until queried on x by $\mathcal{Z}$. Therefore, an encryption $c = m \oplus H(x)$ can

[3] Otherwise, the resulting secure asynchronous uni-directional protocol can always be transformed into a secure one-round asynchronous uni-directional protocol, which in turn can be transformed into a secure one-round synchronous protocol contradicting that two-rounds is stronger than one-round in the synchronous model.

Protocol π^F_{NCE}

The protocol runs with parties $P_1, \ldots, P_n$ in the PRO model and is parametrized by a family of trapdoor permutations F with generator *gen* and domain sampler *dom* for which the domains D_{pk} can be recognized in PPT given the public key pk. The protocol proceeds as follows:

preprocessing phase:
When activated the first time each party P_i generates $(pk_i, sk_i) \leftarrow gen(k)$ and sends pk_i to all other parties.

send:
On input $(\texttt{send}, mid, j, m)$ party P_i generates a uniformly random element $x \leftarrow dom(pk_j)$, computes $(mid, f_{pk_j}(x), H(mid\|i\|j\|x) \oplus m)$, and sends this value to P_j.[a]

receive:
If P_j receives (mid, y, R) from P_i, where $y \in D_{pk_j}$,[b] then P_j computes $x = f^{-1}_{sk_j}(y)$ and $m = R \oplus H(mid\|i\|j\|x)$ and outputs $(\texttt{receive}, mid, i, m)$.

[a] Recall that we use $\|$ to denote some injective and easily parsable encoding $\{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$.

[b] We have require from the family of trapdoor permutations, that one can efficiently check $y \in D_{pk_j}$. Intuitively this is needed for security because a corrupt party P_i might use an honest P_j to determine whether $y \in D_{pk_j}$ by observing P_j's behavior on the message (mid, y, R). So, to be sure this leaks no information we require that P_i could have computed this information itself.

Figure 4.1: The non-committing encryption protocol for a family of trapdoor permutations F.

always be opened to another plaintext m', by redefining $H(x)' := c \oplus m'$, until the simulator is queried on x, at which point it is forced to *commit* on a value for $H(x)$. To hide x from $\mathcal{Z}$ it is enough to send $f(x)$ for a random x, as the one-wayness of f, by definition, prevents the environment from querying on x after having seen only $f(x)$.

4.2.2 The Formal Treatment

The protocol is given in Fig. 4.1. Notice that restricted to P_1 and P_2, the protocol is a non-interactive protocol.

Theorem 4.1 *The protocol π^F_{NCE} realizes $\mathcal{F}_{\mathrm{SMT}}$ in the PRO model.*

Proof. Let $\mathcal{Z}$ be any PPT environment. We construct a simulator $\mathcal{S}$, which running in the ideal process will simulate an execution of π^F_{NCE} to $\mathcal{Z}$. Since the protocol runs in the PRO model, $\mathcal{S}$ will also have to simulate a RO H. It does this by defining $H(h)$ to be some uniformly random value $r \in \{0,1\}^k$, when $H(h)$ is needed. The simulator $\mathcal{S}$ will simulate the preprocessing phase by generating random keys as in the protocol. In fact, to make the proof of security easier we will assume that $\mathcal{S}$, besides running in the ideal process, participates in a trapdoor game (see Definition 2.7 on page 14). The public keys pk_i for the parties will then be obtained from the trapdoor game using n key generation requests. The trapdoors will therefore not be known to $\mathcal{S}$.

To be able to simulate without the trapdoors we represent H in a particular way using two dictionaries `raw` and `img`. At the beginning of the simulation both dictionaries are empty, and H is undefined on all values. We record a new definition $H(h) := r$ as follows:

- If h can be parsed as $mid\|i\|j\|x$, where i and j are indices of parties and $x \in D_{pk_j}$, then the entry $(mid\|i\|j\|y, r)$, where $y = f_{pk_j}(x)$, is added to `img`.
- If h cannot be parsed as described above, then (h, r) is added to `raw`.

We say that $H(h)$ is defined and $H(h) = r$ iff $h = mid\|i\|j\|x$ (for $x \in D_{pk_j}$) and $(mid\|i\|j\|f_{pk_j}(x), r) \in$ `img` or h cannot be parsed as specified and $(h, r) \in$ `raw`. Because f_{pk_j} is a permutation, this representation is consistent — $H(h) = r$ will become defined iff recorded. Equally important, this representation allows to define and evaluate H on $h = mid\|i\|j\|x$, where $x \in dom(pk_j)$ given only $mid\|i\|j\|y$, where $y = f_{pk_j}(x)$. We call these manipulations oblivious and they proceed as follows: Given pk_j, $h = mid\|i\|j\|y$ and r, where $y \in dom(pk_j)$, we do an oblivious record of $H(mid\|i\|j\|f^{-1}_{pk_j}(y)) := r$ by adding $(mid\|i\|j\|y, r)$ to `img`; Given pk_j and $h = mid\|i\|j\|y$, where $y \in dom(pk_j)$, we do an oblivious lookup of $H(mid\|i\|j\|f^{-1}_{pk_j}(y))$ as follows: If there exists $(mid\|i\|j\|y, r) \in$ `img`, then $H(mid\|i\|j\|f^{-1}_{pk_j}(y)) = r$. Notice that in the neither case do we know the point $mid\|i\|j\|f^{-1}_{pk_j}(y)$, but can still define and lookup $H(mid\|i\|j\|f^{-1}_{pk_j}(y))$. The simulation proceeds as specified in Fig. 4.2 on the facing page.

It is easy to see that if the simulation is not given up, then it is distributed exactly as a real-life execution. It is therefore enough to prove that the probability that the simulation is given up is negligible. Assume for the sake of contradiction that the simulation is given up with non-negligible probability. This means that with non-negligible probability

1. The simulator obtained $y = f_{pk_j}(x)$ from the trapdoor game and sent (mid, y, R) from honest P_i to honest P_j.
2. The dictionary was defined on the value $mid\|i\|j\|x$ before $\mathcal{S}$ needed to define it on that value.

Since P_i is guaranteed to be honest up to the point in the simulation where $\mathcal{S}$ needs to define the dictionary on the value $mid\|i\|j\|x$, we can exclude the probability that the simulator has defined H on $mid\|i\|j\|x$ twice, as it would involve choosing the same value y in the image of f_{pk_j} twice under the uniform distribution, which happens with negligible probability. Therefore the other definition of H on $mid\|i\|j\|x$ was made by the environment in RO query. The first definition of H on $mid\|i\|j\|x$ was therefore not oblivious, and thus $x = f^{-1}_{sk_j}(y)$ is known. Since both P_i and P_j are honest up to the point where the simulation is given up, the simulator has not given up on y or pk_j. This allows the simulator to win the trapdoor game with non-negligible probability, a contradiction to Theorem 2.1 on page 15. □

4.3 Impossibility of Non-Interactive NCE in the NPRO Model

In this section we prove that non-interactive non-committing encryption is impossible in the NPRO model. In particular this implies that the protocol π_{NCE} from the previous section is

Interface $\mathcal{S}_{\mathrm{NCE}}$

RO query:

If $\mathcal{Z}$ asks for an evaluation of the RO on some string h, then if $H(h)$ is defined, return $H(h)$, otherwise generate uniformly random $r \in \{0,1\}^k$, record $H(h) := r$, and return r.

send:

On input $(\texttt{send}, mid, i, j, |m|)$ from $\mathcal{F}_{\mathrm{SMT}}$ we know that P_i received input $(\texttt{send}, mid, j, m)$ for some $m \in \{0,1\}^{|m|}$ on $\mathcal{F}_{\mathrm{SMT}}$, which will then output $(\texttt{receive}, mid, i, m)$ to P_j.

- If P_j is corrupt, then $\mathcal{S}$ learns m from $\mathcal{F}_{\mathrm{SMT}}$ and will then simulate by following the protocol using m as the message.
- If P_j is honest, then $\mathcal{S}$ simulates the protocol to $\mathcal{Z}$ by sending the message (mid, y, R), where y is obtained as a uniformly random element in the image of f_{pk_j} from the trapdoor game and $R \in \{0,1\}^k$ is chosen uniformly at random. If at a later point P_i or P_j is corrupted, then:
  - If P_i was corrupted, then $\mathcal{S}$ learns m from $\mathcal{F}_{\mathrm{SMT}}$. The simulator then gives up on y in the trapdoor game and learns x, r such that $y = f_{pk_j}(x)$ and $x = dom(pk_j; r)$. The simulator then gives up on pk_i and learns sk_i, r such that $(pk_i, sk_i) = gen(k; r)$. Then the simulator gives this internal view of P_i to $\mathcal{Z}$ and records $H(mid\|i\|j\|x) := R \oplus m$.
  - If P_j was corrupted, then $\mathcal{S}$ learns m from $\mathcal{F}_{\mathrm{SMT}}$. The simulator then gives up on pk_j and learns sk_j, r such that $(pk_j, sk_j) = gen(k; r)$. Then the simulator gives this internal view of P_j to $\mathcal{Z}$ and records $H(mid\|i\|j\|x) := R \oplus m$.

  If $H(mid\|i\|j\|x)$ was already defined in either of the above cases (to a value different from $R \oplus m$), then the simulator gives up the simulation.

receive:

On the message (mid, y, R) from P_i to P_j, where $y \in dom(pk_j)$, the simulator $\mathcal{S}$ needs to make $\mathcal{F}_{\mathrm{SMT}}$ output $(\texttt{receive}, mid, i, m)$ to P_j, where $m = R \oplus H(mid\|i\|j\|f_{sk_j}^{-1}(y))$.

- If P_i is honest, then (mid, y, R) was sent by $\mathcal{S}$ itself and in that case the message $(\texttt{receive}, mid, i, m)$ will already been sent to P_j in the ideal process.
- If P_i is corrupt, then decrypt as follows: If H is not defined on $mid\|i\|j\|f_{sk_j}^{-1}(y)$, then obliviously define it to a uniformly random value. Then obliviously look up $H(mid\|i\|j\|f_{sk_j}^{-1}(y))$ and let $m = R \oplus H(mid\|i\|j\|f_{sk_j}^{-1}(y))$. Then input $(\texttt{send}, mid, i, j, m)$ on the SIT of $\mathcal{F}_{\mathrm{SMT}}$ to make it deliver the message $(\texttt{receive}, mid, i, m)$ to P_j.

Figure 4.2: The interface $\mathcal{S}_{\mathrm{NCE}}$ used in the proof of Theorem 4.1 on page 161.

not a realization of $\mathcal{F}_{\mathrm{SMT}}$ in the NPRO model (or the real-life mode for that sake) if H is instantiated without interaction. We start with a definition and a lemma.

Definition 4.2 *Let $S^{(\cdot)}$ be a probabilistic ITM with oracle access, let $D^{(\cdot)}$ be a probabilistic TM with oracle access, and let $l_m, l_{sk} : \mathbf{N} \to \mathbf{N}$. We say that $S^{(\cdot)}$ is a NPRO non-committing*

cryptosystem simulator with private key-length l_{sk}, message length l_m, and decryption algorithm $D^{(\cdot)}$ if the following holds: On input $k \in \mathbf{N}$ and access to a RO RO the ITM $S^{(\cdot)}$ outputs a string $c \in \{0,1\}^*$, which we think of as a ciphertext. Then on input $m \in \{0,1\}^{l_m(k)}$ the ITM $S^{(\cdot)}$ outputs a string $sk \in \{0,1\}^{\leq l_{sk}(k)}$, which we think of as a private key. For all $m \in \{0,1\}^{l_m(k)}$, let $\Pr_{S,m}[c, sk, r]$ denote the joint probability distribution of the values (c, sk, r) when c, sk are generated by S^{RO} on inputs (k, m) and r is a uniformly random string. Let $\Pr_S[c, r]$ be the distribution of the values (c, r), which are independent of m. We require that there exists k_0 such that for all $k > k_0$ and all $m \in \{0,1\}^{l_m(k)}$,

$$\Pr_{S,m}[D^{RO}(sk, c; r) = m] \geq \frac{3}{4} \ . \tag{4.1}$$

I.e., we require that the private key sk output by S makes c decrypt to m with probability at least $\frac{3}{4}$.

Lemma 4.1 *If $S^{(\cdot)}$ is a NPRO non-committing cryptosystem simulator with private key-length l_{sk}, message length l_m, and decryption algorithm $D^{(\cdot)}$, then for all $k > k_0$ it holds that $l_{sk}(k) + 2 \geq l_m(k)$.*

Proof. Assume for the sake of contradiction that there exists $k > k_0$ such that $l_{sk}(k) + 2 < l_m(k)$. For all $c \in \{0,1\}^*$ and all $m \in \{0,1\}^{l_m(k)}$ define

$$\begin{aligned} SK(c, m) &= \{sk \in \{0,1\}^{\leq l_{sk}(k)} \mid \Pr_S[D^{RO}(sk, c; r) = m] > \frac{1}{2}\} \\ X_m &= \sum_{c \in \{0,1\}^*} \Pr_S[c] |SK(c, m)| \\ X &= \sum_{m \in \{0,1\}^{l_m(k)}} X_m \ . \end{aligned}$$

Here $SK(c, m)$ is the set of private keys that make c decrypt to m with probability higher than $\frac{1}{2}$, for fixed c and m. So, $|SK(c, m)|$ is the number of such keys and X_m is the expected number of keys making c decrypt (with probability higher than $\frac{1}{2}$) to a fixed m when c is generated by S. Therefore the variable X is the expected number of messages that c can be decrypted to with probability higher than $\frac{1}{2}$. If $sk \in SK(c, m_1) \cap SK(c, m_2)$ and $m_1 \neq m_2$, then $\Pr_S[D^{RO}(sk, c; r) \in \{m_1, m_2\}] > 1$. So, for fixed c each $sk \in \{0,1\}^{\leq l_{sk}(k)}$ belongs to only one of the sets $SK(c, m)$. In words, a given private key sk can make c decrypt to at most one m with probability higher than $\frac{1}{2}$. Therefore

$$\begin{aligned} X &= \sum_{c \in \{0,1\}^*} \Pr_S[c] \sum_{m \in \{0,1\}^{l_m(k)}} |SK(c, m)| \\ &\leq \sum_{c \in \{0,1\}^*} \Pr_S[c](2^{l_{sk}(k)+1} - 1) = 2^{l_{sk}(k)+1} - 1 \ . \end{aligned} \tag{4.2}$$

In words, the expected number of messages that c can be decrypted to with probability higher than $\frac{1}{2}$ is bounded by $2^{l_{sk}(k)+1} - 1$. Since all c must be decryptable to all m, we can use this bound to bound the number of possible messages: If $X_m \geq \frac{1}{4}$ for all $m \in \{0,1\}^{l_m(k)}$, then by Eq. 4.2 we have that $2^{l_{sk}(k)+1} - 1 \geq X \geq 2^{l_m(k)} \frac{1}{4}$ contradicting $l_{sk}(k) + 2 < l_m(k)$.

So, there exists $m \in \{0,1\}^{l_m(k)}$ s.t. $X_m < \frac{1}{4}$. In particular, by the Markov inequality $\Pr_S[|SK(c,m)| \geq 1] < \frac{1}{4}$ and

$$\begin{aligned}
&\Pr_{S,m}[D^{RO}(sk,c;r) = m] \\
=&\Pr_{S,m}[|SK(c,m)| \geq 1] \cdot \Pr_{S,m}[D^{RO}(sk,c;r) = m||SK(c,m)| \geq 1] + \\
&\Pr_{S,m}[|SK(c,m)| = 0] \cdot \Pr_{S,m}[D^{RO}(sk,c;r) = m||SK(c,m)| = 0] \\
<&\frac{1}{4} \cdot 1 + 1 \cdot \frac{1}{2} = \frac{3}{4}
\end{aligned}$$

contradicting (4.1). □

Theorem 4.2 *There exists no non-interactive protocol realizing $\mathcal{F}_{\mathrm{SMT}}$ in the NPRO model with erasure.*[4]

Proof. Consider the execution of a non-interactive NCE protocol with two parties $P_S^{(\cdot)}$ and $P_R^{(\cdot)}$. Let $sk(k)$ denote the random variable describing the internal state of $P_R^{(\cdot)}$ after the pre-processing phase and before receiving the first encryption. Since this state is independent of the inputs to be sent — it is defined given the code of P_R — there exists a polynomial $l_{sk}(k)$ s.t. the expected value of $|sk(k)|$ is bounded by $\frac{l_{sk}(k)}{6}$ for large enough k. So, by the Markov inequality $\Pr[|sk(k)| \geq l(k)] < \frac{1}{6}$ for large enough k.

Consider the following environment $\mathcal{Z}^{(\cdot)}$: It first activates $P_S^{(\cdot)}$ on the message `init` and waits s rounds. Then it activates $P_S^{(\cdot)}$ on a message $m \in \{0,1\}^{l_m(k)}$, where $l_m(k) = l_{sk}(k) + 3$ and m is a prefix of the auxiliary string z. Let E_1 denote the event that $P_S^{(\cdot)}$ sends a value $c \in \{0,1\}^*$ to $P_R^{(\cdot)}$. If does not occur, the environment terminates with output 0. If E_1 occurs, the environment corrupts $P_R^{(\cdot)}$ *before c arrives*. Let E_2 be the event that $\mathcal{Z}^{(\cdot)}$ as response to this corruption receives a value $sk \in \{0,1\}^{\leq l_{sk}(k)}$. If E_2 does not occur, $\mathcal{Z}^{(\cdot)}$ outputs 0. If E_2 does occur, $\mathcal{Z}^{(\cdot)}$ generates uniformly random bits r and computes m' by running the code of $P_R^{(\cdot)}$ from internal state sk with input c and random bits r and using the random oracle RO — since P_R does not access any ideal functionalities during decryption and $\mathcal{Z}^{(\cdot)}$ has access to the same oracle RO as $P_R^{(\cdot)}$, the environment can actually carry out this computation. If $m' = m$ then $\mathcal{Z}$ outputs 1, otherwise $\mathcal{Z}$ outputs 0.

Let E be the event that E_1 and E_2 occurs. Note that in $\mathrm{HYB}^{\mathcal{G}}_{\pi^{RO},\mathcal{Z}^{RO}}(k,z)$, where we let $\mathcal{G}$ be any ideal functionality, the event E occurs with probability at least $\frac{5}{6}$ for large enough k. Furthermore, by the security (which implies correctness) of the protocol, the probability that $m' \neq D^{RO}(sk,c;r)$ for uniformly random r is bounded by a negligible function $\delta(k)$. Therefore $\Pr[\mathrm{HYB}^{\mathcal{G}}_{\pi^{RO},\mathcal{Z}^{RO}}(k,z) = 1] \geq \frac{5}{6} - \delta(k) > \frac{4}{5}$ for large enough k. Since the protocol is secure there exists $\mathcal{S}^{(\cdot)}$ and k_0 such that for all $k > k_0$ we have that $\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F}_{\mathrm{SMT}},\mathcal{S}^{RO},\mathcal{Z}^{RO}}(k,z) = 1] \geq \frac{3}{4}$. We use this to construct a NPRO non-committing cryptosystem simulator $S^{(\cdot)}$ as follows: On input k and access to oracle RO run

[4] We have not defined the erasure model formally, but the environment is simply given the current internal state of a party instead of its initial random bits, and the parties can erase values at desire during the computation.

$\text{IDEAL}^{\mathcal{G}}_{\mathcal{F}_{\text{SMT}},\mathcal{S}^{RO},\mathcal{Z}^{RO}}(k,?)$ on a message of length $l_m(k)$. Note that as long as no party is corrupted the execution does not require that we know the value of the message, which is the reason for the ?-notation. If during the execution the event E does not occur, $S^{(\cdot)}$ uses arbitrary values for c and sk. If E occurs, $S^{(\cdot)}$ proceeds as follows: Run $\text{IDEAL}^{\mathcal{G}}_{\mathcal{F}_{\text{SMT}},\mathcal{S}^{RO},\mathcal{Z}^{RO}}(k,?)$ until $\mathcal{S}$ outputs c and then output c. Then on input $m \in \{0,1\}^{l_m(k)}$ run $\text{IDEAL}^{\mathcal{G}}_{\mathcal{F}_{\text{SMT}},\mathcal{S}^{RO},\mathcal{Z}^{RO}}(k,m)$ until $P_R^{(\cdot)}$ is corrupted, give $\mathcal{S}^{(\cdot)}$ the value m, and run $\text{IDEAL}^{\mathcal{G}}_{\mathcal{F}_{\text{SMT}},\mathcal{S}^{RO},\mathcal{Z}^{RO}}(k,m)$ until $\mathcal{S}^{(\cdot)}$ outputs sk. Then output sk. Let $D^{(\cdot)}$ be the TM which on input c, sk and access to oracle RO runs P_R^{RO} from internal state sk and input c using uniformly random bits. By $\Pr[\text{IDEAL}^{\mathcal{G}}_{\mathcal{F}_{\text{SMT}},\mathcal{S}^{RO},\mathcal{Z}^{RO}}(k,z)=1] \geq \frac{3}{4}$ we have that S is an NPRO non-committing cryptosystem simulator with private key-length l_{sk}, message length l_m, and decryption algorithm D, contradicting Lemma 4.1. □

In the above proof we used essentially that P_S could send an arbitrary-length message to P_R and that we could corrupt P_R before this message arrived. If we changed the functionality $\mathcal{F}_{\text{SMT}}$ such that only fixed-length messages could be sent and such that P_S can only send a message *after* the previous message was *received* by P_R, then the proof fails as P_R might delete or change sk between the received messages. In that case we can only prove that the internal state sk of P_R must be at least the size of the message that can be sent in one invocation. A scheme where P_R is only sent fixed-length messages and is given the possibility of deleting between messages is called a key-evolving public-key encryption scheme. This notion was coined in the context of forward-secure public-key encryption [And97]. In a forward-secure public-key encryption scheme exposure of the secret key will not allow the adversary to decrypt previous ciphertexts. Theorem 4.2 on the page before was used by Canetti, Halevi and Katz [CHK03] to argue the necessity of key-evolving in non-interactive forward-secure public-key encryption schemes.

4.4 Non-Committing Encryption from Simulatable Public-Key Systems

In this section we describe how to construct non-committing encryption from so-called simulatable public-key cryptosystems. These are standard cryptosystems with the additional properties that a public-key can be invertibly sampled and that a ciphertext with the same distribution as an encryption of a random plaintext can be invertibly sampled.

Definition 4.3 *We call $(\mathcal{K},\widetilde{\mathcal{K}},\mathcal{E},\mathcal{D},\mathcal{M},\mathcal{C})$ a simulatable public-key cryptosystem if $(\mathcal{K},\mathcal{E},\mathcal{D},\mathcal{M})$ is a semantic secure public-key cryptosystem (according to Definition 2.8 on page 16 and Definition 2.9 on page 16), and the following holds:*

oblivious public-key generation:
$\widetilde{\mathcal{K}}$, called the oblivious public-key generator, is a PPT invertible sampler for the distribution given by running $(P,S) \leftarrow \mathcal{K}$ and outputting P.

oblivious ciphertext generation:
$\mathcal{C}$, called the oblivious ciphertext generator, is a PPTIS algorithm and the distribution

obtained by running

$$[(P,S) \leftarrow \mathcal{K}; C \leftarrow \mathcal{C}_P : (P,S,C)]$$

is statistically indistinguishable from the distribution obtained by running

$$[(P,S) \leftarrow \mathcal{K}; M \leftarrow \mathcal{M}_P; C \leftarrow \mathcal{E}_P(M) : (P,S,C)] \ .$$

4.4.1 Motivating Invertible Sampling

As mentioned in Section 2.8, invertible sampling is closely connected to adaptive security in models where security is defined by requiring that an adversary's view of a real-life execution of a protocol can be efficiently simulated (i.e in PPT) given just the data the adversary is entitled to, and where erasures are not allowed. We are now ready to explain why. The example also motivates why we require invertible sampling in Definition 4.3 on the facing page for the encryption scheme that we will soon use to build an adaptively secure protocol, namely to avoid problems as the following: Consider the functionality which on input `gen` from P_1 outputs a uniformly random $y \in \{0,1\}^l$ to P_1. Let f be a permutation on $\{0,1\}^l$ and consider the protocol where a party P_1 on input `gen` outputs $y = f(x)$ for uniformly random $x \in \{0,1\}^l$. All other parties do nothing. The output of the protocol is a uniformly random string y and therefore seems to be secure. However, in the real-life execution, after $\mathcal{Z}$ inputs `gen` to P_1 it sees output $y \in \{0,1\}^l$. If $\mathcal{Z}$ then corrupts P_1 it sees x such that $y = f(x)$. By definition of security there should exist a simulator which is given y when P_1 is corrupted and then computes x to show to the environment to simulate the internal state of P_1. It should of course be the case that $y = f(x)$, otherwise the environment could distinguish. Since the simulator is PPT it follows that the protocol is secure iff f has invertible sampling. Notice that if f is a one-way function, then indeed f does *not* have invertible sampling. So, if there exists one-way functions, then there exists 1-party protocols with no input (and no communication) and with correct output distribution, which are not adaptively secure in the cryptographic model. Notice that the above protocol is *static* secure and that *any* 1-party protocols with no input (and no communication) and with correct output distribution is static secure. This shows that the considered problem is closely related to the internal state shown to the adversary after a corruption.

4.4.2 Weak Test for Equality

We will realize the functionality in Fig. 3.24 on page 115 for secure message transmission given any simulatable public-key cryptosystem. We will use the UC framework to give a modular construction of and proof of security for our scheme: We first realize the functionality $\mathcal{F}_{\text{WTE}}$ in Fig. 4.3 on the following page and then show how $\mathcal{F}_{\text{WTE}}$ can be used to realize $\mathcal{F}_{\text{SMT}}$.

The functionality $\mathcal{F}_{\text{WTE}}$ allows two parties, P_i and P_j say, both of them holding a bit, c and d say, to test whether $c = d$ without revealing c (or d) to the environment — the outcome of the test is however public. We call such a protocol a test for equality. In the ideal functionality $\mathcal{F}_{\text{WTE}}$, if the outcome of the test is $c \neq d$, then in addition to learning this, the environment will also learn c (and thus d). I.e. the functionality only hides c and d when $c = d$. We call this a weak test for equality.

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

The functionality runs with parties $P_1, \ldots, P_n$.

first bit:

On input $(id, 1, j, c)$ for $c \in \{0, 1\}$ from any P_i, output (id, i) to P_j and output (id, i, j) to the adversary. Store $(id, 1, i, j, c)$. If P_i is corrupted in this round, then let the environment decide whether to delete $(id, 1, i, j, c)$ and drop the message (id, i) to P_j.

comparison:

On input $(id, 2, i, d)$ from any P_j, where $d \in \{0, 1\}$, if $(id, 1, i, j, c)$ is stored, then delete $(id, 1, i, j, c)$, store $(id, 2, i, j, c, d)$, output $s = c \oplus d$ to P_i and output $(id, s, s \wedge c)$ to the adversary. If P_j is corrupted in this round, then $\mathcal{Z}$ can send another message as follows: If the environment inputs (id, i, j, s) and P_j is corrupted and $(id, 2, i, j, c, d)$ is stored, then output s to P_i.

incorrect inputs:

If an honest P_i ever uses the same id id twice, then break down. If an honest P_j ever inputs $(id, 2, i, d)$, where $(id, 1, i, j, c)$ is not stored for some $c \in \{0, 1\}$, then break down.

Figure 4.3: The functionality for Weak Test for Equality.

Before showing how to realize $\mathcal{F}_{\text{WTE}}$, we motivate by showing how to reduce $\mathcal{F}_{\text{SMT}}$ to $\mathcal{F}_{\text{WTE}}$. Consider the protocol in Fig. 4.4 on the next page. Since all c_i and d_i are chosen uniformly at random and $4(|m| + k)$ indices are used it follows directly from the Markov inequality that $|m|$ indices will have $c_i = d_j$, except with negligible probability $\exp(-\frac{k}{2})$. And, for those indices the environment only see the information $s_i = 0$ about c_i and d_i, telling it that $c_i = d_i$. Therefore, c_i is a uniformly random bit stochastically independent of the view of the environment, which the parties can safely use for one-time pad encryption. In terms of simulators, when $c_i = d_i$ then the simulator is not committed to c_i before one of the parties is corrupted. Specifying a simulator for π_{WTE} is straight-forward and is left to the reader.

Theorem 4.3 *The protocol π_{SMT} realizes $\mathcal{F}_{\text{SMT}}$ in the $\mathcal{F}_{\text{WTE}}$-hybrid model.*

A realization of $\mathcal{F}_{\text{WTE}}$ is given in Fig. 4.5 on page 170. We give an intuitive sketch of the security. The bit d is encoded by which of the tuples (P_0, M_0, C_0) and (P_1, M_1, C_1) match in the sense that $C_d = \mathcal{E}_{P_d}(M_d)$. Because the secret key S_c is known to P_i it can compute the bit d: If $\mathcal{D}_{S_c}(C_c) \neq M_c$, then $c \neq d$ and if $\mathcal{D}_{S_c}(C_c) = M_c$, then, except with negligible probability, $c = d$ (we require that the size of the message space is super-polynomial). Therefore, except with negligible probability, $s = c \oplus d$. This guarantees the correctness. For the secrecy, observe that the values (P_0, M_0, C_0) and (P_1, M_1, C_1) observed by a passive adversary are computationally independent of c, almost by the definition of IND-CPA security (see Definition 2.9 on page 16). The protocol is therefore a secure test for equality against an eavesdropper. This then leaves us with the question whether it is a secure test for equality against an adversary corrupting the parties, adaptively. The answer is negative. It is only a secure weak test for equality.

Protocol π_{SMT}

The protocol runs with parties $P_1, \ldots, P_n$ in the $\mathcal{F}_{\text{WTE}}$-hybrid model.

1. P_i on input (id, j, m) proceeds as follows:
   (a) Let $l = 4(|m| + k)$.
   (b) Generate uniformly random $c \in \{0,1\}^l$.
   (c) Send $(id, |m|)$ to P_j and for $o = 1, \ldots, l$, input $(id \| o, 1, j, c_o)$ to $\mathcal{F}_{\text{WTE}}$.
2. P_j, on a message (id, p) from P_i and on receiving $(id \| o, i)$ from $\mathcal{F}_{\text{WTE}}$ for $o = 1, \ldots, l$, where $l = 4(|m| + k)$, proceeds as follows:
   (a) Generate uniformly random $d \in \{0,1\}^l$.
   (b) For $o = 1, \ldots, l$, input $(id \| o, 2, i, d_o)$ to $\mathcal{F}_{\text{WTE}}$.
3. P_i on receiving $(id \| o, j, s_o)$ from $\mathcal{F}_{\text{WTE}}$ for $o = 1, \ldots, l$, proceed as follows:
   (a) Pick $(o_1, \ldots, o_{|m|})$ such that $\{o_1, \ldots, o_{|m|}\} \subset [l]$, $|\{o_1, \ldots, o_{|m|}\}| = |m|$ and $s_{o_h} = 0$ for $h = 1, \ldots, |m|$. If no such value exists, then allow $s_{o_h} = 1$ for some o_h.
   (b) For $h = 1, \ldots, |m|$, let $f_h \leftarrow m_h \oplus c_{o_h}$.
   (c) Send $((id, 2), \{(o_h, f_h)\}_{h \in [|m|]})$ to P_j.
4. P_j on receiving $((id, 2), \{(o_h, f_h)\}_{h \in [|m|]})$ from P_i computes $m_h \leftarrow f_h \oplus d_{o_h}$ for $h \in [|m|]$ and outputs (id, i, m)

Figure 4.4: The protocol π_{SMT} reducing secure message transmission to weak test for equality.

To see why it *is* a secure weak test for equality, consider the following simulator: It runs both parties according to the protocol with the following exceptions: When $s = 1$ the simulator simulates by following the protocol; When $s = 0$ the key P_{1-c} is generated as $(P_{1-c}, S_{1-c}) \leftarrow \mathcal{K}(r_{1-c})$ and the ciphertext C_{1-d} is generated as $C_{1-d} \leftarrow \mathcal{E}_{P_{1-d}}(M_{1-d}, e_{1-d})$ for random M_{1-d}. Note that when $s = 1$ the simulator knows all inputs to the parties, so the simulation is trivial, and when $s = 0$ the keys P_0 and P_1 is generated in the same way and C_0 and C_1. Therefore the simulator does not have to know c and d. We assume in the following treatment that both parties are corrupted after all the communication has been simulated. The other cases are treated similarly. When $s = 0$ the communication of the simulation is of the form (P_0, M_0, C_0) and (P_1, M_1, C_1), where both triples are such that $C_i = \mathcal{E}_{P_i}(M_i)$. However, as long as no secret keys are known to the adversary this cannot be observed, by the IND-CPA security. Now assume that the adversary breaks into both parties. The simulator then learns the bit $c = d$ to be communicated (the cases where only one party is corrupted is a special case because no matter which party is corrupted the simulator will learn c or $d = c$). Now that c and d become known the random bits of the parties are simulated as follows: To be convinced that it is observing the state of parties from a real-life execution the adversary should be given random values $r'_c, r'_{1-c}, e'_d, e'_{1-d}$ which looks uniformly random

Protocol π_{WTE}

The protocol runs in the real-life model with parties $P_1, \ldots, P_n$.

1. P_i on input $(id, 1, j, c)$, where $j \in [n]$, proceeds as follows:
   (a) Generate a public-key/secret-key pair $(P_c, S_c) \leftarrow \mathcal{K}(r_c)$.
   (b) Generate a public-key obliviously $P_{1-c} \leftarrow \widetilde{\mathcal{K}}(r_{1-c})$.
   (c) Store (id, j, c, S_c) and send $(id, 1, P_0, P_1)$ to P_j.
   (d) P_j on receiving $(id, 1, P_0, P_1)$ from P_i stores (id, i, P_0, P_1) and outputs (id, i).
2. P_j on input $(id, 2, i, d)$, where $i \in [n]$ and (id, i, P_0, P_1) is stored, proceeds as follows:
   (a) For $b \in \{0, 1\}$, generate two random plaintexts $M_b \leftarrow \mathcal{M}_{P_b}$.
   (b) Generate an encryption $C_d \leftarrow \mathcal{E}_{P_d}(M_d; e_d)$.
   (c) Generate a random ciphertext $C_{1-d} \leftarrow \mathcal{C}_{P_{1-d}}(e_{1-d})$.
   (d) Send $(id, 2, M_0, M_1, C_0, C_1)$ to P_i.
3. P_i on a message $(id, 2, M_0, M_1, C_0, C_1)$ from P_j, where some (id, j, c, S_c) is stored, proceeds as follows: If $\mathcal{D}_{S_c}(C_c) = M_c$, then $s \leftarrow 0$. Otherwise $s \leftarrow 1$. Output s.

Figure 4.5: The protocol for weak test for equality.

and for which is holds that

$$(P_c, S_c) = \mathcal{K}(r'_c) \tag{4.3}$$
$$P_{1-c} = \widetilde{\mathcal{K}}(r'_{1-c}) \tag{4.4}$$
$$C_d = \mathcal{E}_{P_d}(M_d, e_d) \tag{4.5}$$
$$C_{1-d} = \mathcal{C}_{P_{1-d}}(e_{1-d}) \ . \tag{4.6}$$

Eq.s. 4.3 and 4.5 are easily handled by setting $r'_c = r_c$ and $e'_d = e_d$. Eq.s. 4.4 and 4.6 are handled using the random bits fakers given by the definition of invertible sampling. By Definition 2.16 on page 28 there exists algorithms $\widetilde{\mathcal{K}}^{-1}$ and $\mathcal{C}^{-1}$ such that $r'_{1-c} \leftarrow \widetilde{\mathcal{K}}^{-1}(P_{1-c})$ and $e'_{1-d} \leftarrow \mathcal{C}^{-1}(P_{1-d}, C_{1-d})$ has exactly the desired properties except for a statistical error. Observe that as opposed to the protocol it holds for $(P_{1-c}, M_{1-c}, C_{1-c})$ that $C_{1-c} = \mathcal{E}_{P_{1-c}}(M_{1-c})$. However, the adversary was not given S_{1-c}, so this difference will not be observed.

To see why the protocol is *not* necessarily a secure test for equality, consider an execution where $c \neq d$. If the adversary breaks into P_i after the execution he learns S_c and using S_c he can check that $\mathcal{D}_{S_c}(C_c) \neq M_c$. When he breaks into P_j he learns e_d and can check that $C_d = \mathcal{E}_{P_d}(M_d, e_d)$. I.e. exactly one of the tuples (P_0, M_0, C_0) and (P_1, M_0, C_0) match $C_i = \mathcal{E}_{P_i}(M_i, e_i)$ and as these tuples are defined before the corruption, the simulator will be committed to the value of c (and d) before the corruption takes place. If not it can generate values (P, S, M, e, C) such that P, S, M, and e looks random and $C = \mathcal{E}_P(M, e)$ and $\mathcal{D}_S(C) \neq M$. None of the encryption schemes that we construct later have this property.

Interface $\mathcal{S}_{\mathrm{WTE}}$

1. On a value (id, i, j) from $\mathcal{F}_{\mathrm{WTE}}$ proceeds as follows:

   (a) For $c' \in \{0,1\}$, generate a public-key/secret-key pair $(P_{c'}, S_{c'}) \leftarrow \mathcal{K}(r_{c'})$.

   (b) Store (id, j, S_0, S_1) and simulate a send of $(id, 1, P_0, P_1)$ to P_j.

   (c) If $\mathcal{Z}$ delivers $(id, 1, P_0, P_1)$,[a] then in the ideal process deliver the message (id, i) sent from $\mathcal{F}_{\mathrm{WTE}}$ to P_j to make P_j output (id, i).[b]

   If P_i is corrupted after this step we learn $c \in \{0,1\}$ and patch the random bits of P_i as follows:

   - $r'_{1-c} \leftarrow \widetilde{\mathcal{K}}^{-1}(P_{1-c})$.
   - $r'_c \leftarrow r_c$.

2. On a value (id, i, j, s, f) from $\mathcal{F}_{\mathrm{WTE}}$ proceeds as follows:

   - If $s = 1$, then

     (a) Let $c = f$ and let $d = 1 - c$.

     (b) Patch the random bits of P_i as described in Step 1.

     (c) Run P_j as in the protocol on input $(id, 2, i, d)$. If $\mathcal{Z}$ delivers the message $(id, 2, M_0, M_1, C_0, C_1)$ to P_i, then in the ideal process deliver the message $(id, j, 1, c)$ sent from $\mathcal{F}_{\mathrm{WTE}}$ to P_i.

   - If $s = 0$, then

     (a) For $b \in \{0,1\}$, generate two random plaintexts $M_b \leftarrow \mathcal{M}_{P_b}$.

     (b) For $b \in \{0,1\}$, generate an encryption $C_b \leftarrow \mathcal{E}_{P_b}(M_b; e_b)$.

     (c) Simulate a send of $(id, 2, M_0, M_1, C_0, C_1)$ to P_i. If $\mathcal{Z}$ delivers the message $(id, 2, M_0, M_1, C_0, C_1)$ to P_i, then in the ideal process deliver the message $(id, j, 0, 0)$ sent from $\mathcal{F}_{\mathrm{WTE}}$ to P_i.

     If P_i or P_j is corrupted after this step we learn c or d. Since $s = 0$ we have $c = d$. Patch the random bits of P_i and P_j as follows:

     (a) Patch the random bits of P_i as described in Step 1.

     (b) $e'_{1-d} \leftarrow \mathcal{C}^{-1}(P_{1-c}, C_{1-c})$.

     (c) $e'_d \leftarrow e_d$.

     If $\mathcal{Z}$ corrupts P_j and delivers another message of the form $(id, 2, M_0, M_1, C_0, C_1)$ to P_i, then we learn s from $\mathcal{F}_{\mathrm{SMT}}$, patch P_i as described in Step 1, and then run P_i on $(id, 2, M_0, M_1, C_0, C_1)$ to determine an output value s. Then input (id, i, j, s) to $\mathcal{F}_{\mathrm{WTE}}$ to make it deliver the message (id, j, s) to P_i.

[a] It could corrupt P_i and send another message.

[b] I.e., specify to $\mathcal{F}_{\mathrm{WTE}}$ not drop the message (id, i) to P_j.

Figure 4.6: The interface $\mathcal{S}_{\mathrm{WTE}}$ used in the proof of Lemma 4.2 on the following page.

Adversary A_{WTE}

The adversary runs as part of the RIND-CPA in Fig. 2.3 on page 17 with $S = (\mathcal{G}, \mathcal{E}, \mathcal{D})$. It generates oblivious keys and random ciphertexts by access to the RIND-CPA game. The first input to the adversary is $k \in \mathbf{N}$ and $z \in \{0,1\}^*$. Start running $\mathrm{REAL}_{\pi_{\mathrm{WTE}},\mathcal{Z}}(k,z)$, but with the following changes to π_{WTE}:

1. P_i on input $(id, 1, j, c)$, where $j \in [n]$, proceeds as follows:
   (a) Generate a public-key/secret-key pair $(P_c, S_c) \leftarrow \mathcal{K}(r_c)$.
   (b) Output (`generate`) to the RIND-CPA game and receive a public-key P_{1-c}. Then compute $r_{1-c} \leftarrow \widetilde{\mathcal{K}}^{-1}(P_{1-c})$.[a]
   (c) Store (id, j, c, S_c) and send $(id, 1, P_0, P_1)$ to P_j.
   (d) P_j on receiving $(id, 1, P_0, P_1)$ from P_i stores (id, i, P_0, P_1) and outputs (id, i).
2. P_j on input $(id, 2, i, d)$, where $i \in [n]$ and (id, i, P_0, P_1) is stored, proceeds as follows:
   (a) For $b \in \{0,1\}$, generate two random plaintexts $M_b \leftarrow \mathcal{M}_{P_b}$.
   (b) Generate an encryption $C_d \leftarrow \mathcal{E}_{P_d}(M_d; e_d)$.
   (c) i. If $c \neq d$, then generate a random ciphertext $C_{1-d} \leftarrow \mathcal{C}_{P_{1-d}}(e_{1-d})$.
   ii. If $c = d$, then generate a random plaintext $M' \leftarrow \mathcal{M}_{P_{1-d}}$ and output (`test-messages`, P_{1-d}, M', M_{1-d}) to the RIND-CPA game and receive a ciphertext C_{1-d}. Then let $e_{1-d} \leftarrow \mathcal{C}^{-1}(P_{1-d}, C_{1-d})$.[b]
   (d) Send $(id, 2, M_0, M_1, C_0, C_1)$ to P_i.
3. P_i on a message $(id, 2, M_0, M_1, C_0, C_1)$ from P_j, where some (id, j, c, S_c) is stored, proceeds as follows: If $\mathcal{D}_{S_c}(C_c) = M_c$, then $s \leftarrow 0$. Otherwise $s \leftarrow 1$. Output s.

When $\mathcal{Z}$ stops with some output bit b, output (`guess`, b) to the RIND-CPA game.

[a] Here we used that $\widetilde{\mathcal{K}}^{-1}$ does not use the bits used to generate P_{1-c}.
[b] Here we used that $\mathcal{C}^{-1}$ does not use the bits used to generate C_{1-d}.

Figure 4.7: The RIND-CPA adversary used in the proof of Lemma 4.2.

Lemma 4.2 *Based on a simulatable public-key cryptosystem* $(\mathcal{K}, \widetilde{\mathcal{K}}, \mathcal{E}, \mathcal{D}, \mathcal{M}, \mathcal{C})$ *with super-polynomial plaintext space, the protocol* π_{WTE} *realizes* $\mathcal{F}_{\mathrm{WTE}}$.

Proof. We prove that $\mathrm{REAL}_{\pi_{\mathrm{WTE}},\mathcal{Z}} \overset{\mathrm{c}}{\approx} \mathrm{IDEAL}_{\mathcal{F}_{\mathrm{WTE}},\mathcal{S}_{\mathrm{WTE}},\mathcal{Z}}$ for all environments $\mathcal{Z}$, where $\mathcal{S}_{\mathrm{WTE}}$ is given in Fig. 4.6 on the preceding page. We prove $\mathrm{REAL}_{\pi_{\mathrm{WTE}},\mathcal{Z}} \overset{\mathrm{c}}{\approx} \mathrm{IDEAL}_{\mathcal{F}_{\mathrm{WTE}},\mathcal{S}_{\mathrm{WTE}},\mathcal{Z}}$ using a hybrids arguments. Consider the RIND-CPA adversary A_{WTE} in Fig. 4.7. Let $\mathrm{REAL}_{\pi'_{\mathrm{WTE}},\mathcal{Z}}$ denote the copy of $\mathrm{REAL}_{\pi_{\mathrm{WTE}},\mathcal{Z}}$ executed in A_{WTE}, and for $c \in \{0,1\}$, let $\mathrm{REAL}_{\pi'_{\mathrm{WTE}}(c),\mathcal{Z}}$ denote the distribution of $\mathrm{REAL}_{\pi'_{\mathrm{WTE}},\mathcal{Z}}$ when $b = c$ in the RIND-CPA game. It is straight-forward to verify that $\mathrm{REAL}_{\pi'_{\mathrm{WTE}}(0),\mathcal{Z}}$, is statistically indistinguishable from $\mathrm{REAL}_{\pi_{\mathrm{WTE}},\mathcal{Z}}$. To see this observe that in $\mathrm{REAL}_{\pi'_{\mathrm{WTE}}(0),\mathcal{Z}}$ we have that C_{1-d} is an encryption of a random element as in the protocol $\mathrm{REAL}_{\pi_{\mathrm{WTE}},\mathcal{Z}}$ and e_{1-d} and r_{1-c} are statistically close to the distribution in $\mathrm{REAL}_{\pi_{\mathrm{WTE}},\mathcal{Z}}$ by Definition 4.3 on page 166. Furthermore,

$\mathrm{REAL}_{\pi'_{\mathrm{WTE}}(1),\mathcal{Z}}$, is identical distributed to $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{WTE}},\mathcal{S}_{\mathrm{WTE}},\mathcal{Z}}$. To see this observe that in $\mathrm{REAL}_{\pi'_{\mathrm{WTE}}(1),\mathcal{Z}}$ we have that C_{1-d} is an encryption of M_{1-d} as in $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{WTE}},\mathcal{S}_{\mathrm{WTE}},\mathcal{Z}}$ and e_{1-d} and r_{1-c} are computed in the same manner in both. The lemma then follows by the IND-CPA security of S and Theorem 2.2 on page 16. □

4.5 Some Simulatable Public-Key Systems

In this section we describe two simulatable public-key cryptosystems, one based on the DDH assumption and one based on the RSA assumption.

Definition 4.4 *The ElGamal encryption scheme in Q_p is defined as follows:*

key generation:

Generate a random k-bit prime p for which $q = (p-1)/2$ is a prime. In the following, let Q_p denote the quadratic residues modulo p and let $g = 4$. It is easy to verify that $|Q_p| = q$ and $\langle g \rangle_{\mathbf{Z}_p^*} = Q_p$. *Let $h = g^x \bmod p$ for uniformly random $x \in \mathbf{Z}_q$. The public key is (p, h) and the private key is (p, x).*

oblivious public-key generation:

To obliviously generate a public key, generate p as in the key generation above, generate $h' \in \mathbf{Z}_p^*$ uniformly at random and let $h = h^2 \bmod p$. The oblivious public-key is (p, h).

plaintext sampling:

The plaintext space is $\mathbf{Z}_q$, which can be invertibly sampled by Theorem 2.3 on page 28.

encryption:

Let (p, h) be a public-key and let $m \in \mathbf{Z}_q$ be a plaintext. We encrypt as follows: If $m \in Q_p$, then let $m' = m$; Otherwise, let $m' = p - m$. Generate uniformly random $r \in \mathbf{Z}_q$ and let $enc((p, h), m; r) = (\alpha, \beta) = (g^r \bmod p, m'h^r \bmod p)$.

decryption:

Given a private key (p, x) and a ciphertext (α, β), let $m' = \alpha^{-x}\beta \bmod p$. If $m' \in \mathbf{Z}_q$, then let $m = m'$; Otherwise, let $m = p - m'$. Then $dec((p, x), (\alpha, \beta)) = m$.

oblivious ciphertext generation:

To obliviously generate a ciphertext for public key (p, h), generate $\alpha', \beta' \in \mathbf{Z}_p^*$ uniformly at random and let $\alpha = \alpha'^2 \bmod p$ and $\beta = \beta'^2 \bmod p$. The oblivious ciphertext is (α, β).

Theorem 4.4 *The ElGamal encryption scheme in Q_p is simulatable and is IND-CPA secure (Definition 2.9 on page 16) if the DDH assumption (Assumption 2.1 on page 21) holds.*

Proof. That all algorithms in the encryption scheme is PPT is well-known and the correctness is easy to verify. That the oblivious public-key generator has the correct distribution follows from the fact that both $g^x \bmod p$ and h'^2 are random elements from Q_p. That the oblivious public-key generator has invertible sampling follows from Theorem 2.3 on page 28 and the fact that $x \mapsto x^2 \bmod p$ is 2-to-1 in $\mathbf{Z}_p^*$ and that a random preimage can be found in PPT. That the oblivious ciphertext generator has invertible sampling follows from the same observation. The

IND-CPA security of the ElGamal encryption scheme in Q_p relative to the DDH assumption in Q_p is well-known, here using that m' as defined in the encryption algorithm always is an element of Q_p. □

4.5.1 Simulatable Public-Key Systems from Simulatable Trapdoor Permutations

In this section we construct a simulatable encryption scheme based on the RSA assumption. We do this in two step. First we define the notion of simulatable family of trapdoor permutations and show that a standard construction of IND-CPA secure encryption schemes from families of trapdoor permutations maintain the simulatability. We then construct simulatable family of trapdoor permutations based on the RSA assumption.

Definition 4.5 *We call* $(gen, \widetilde{gen}, dom, f, f^{-1})$ *a* simulatable family of trapdoor permutations *if* (gen, dom, f, f^{-1}) *is a family of trapdoor permutations and*

oblivious index generation:
$\widetilde{gen}$*, called the* oblivious index-generator*, is a PPT invertible sampler for the distribution obtained by running* $(i, t_i) \leftarrow gen$ *and outputting* i*.*

invertible domain sampling:
dom *is a PPTIS algorithm.*

Definition 4.6 *We define an encryption scheme, which we will call the* Blum-Goldwasser scheme with fixed plaintext length*. Given a simulatable family of trapdoor permutations* $(gen, \widetilde{gen}, dom, f, f^{-1})$*, let* B *be a hard-core predicate of the family of trapdoor permutations. If no such* B *exists one can construct a new simulatable family of trapdoor permutations which has one, following the construction in [GL89] (which trivially preserves simulatability). We construct an encryption scheme* $(gen', \widetilde{gen}', enc, dec, pla, cip)$*. The (oblivious) key-generator is* $gen' = gen$ *(*$\widetilde{gen}' = \widetilde{gen}$*). The message space can be* $\{0,1\}^{p(k)}$ *for any polynomial* $p(k)$*, and the ciphertext space for* $P = i$ *is* $\{0,1\}^{p(k)} \times D_i$*, where* D_i *is the domain of* f_i*. For* $(i,t) \in gen$*, let* $X(i,x,n) = B(x)B(f_i(x))B(f_i^2(x)) \dots B(f_i^{n-1}(x))$ *be an* n*-bit pseudorandom string generated from* $x \in D_i$*. Then the encryption of* $m \in \{0,1\}^{p(k)}$ *under* i *is* $(m \oplus X(i,x,|m|), f_i^{|m|}(x))$ *for random* $x \leftarrow dom(i)$*. The decryption is trivial given* t*. To pick a ciphertext obliviously for a given public key* i*, generate* $m' \in \{0,1\}^{p(k)}$ *uniformly at random, generate* $x \leftarrow gen(i)$ *and let* $cip(i) = (m', x)$*. This will be distributed exactly as* $enc(i,m)$ *for uniformly random* $m \in \{0,1\}^{p(k)}$*. Invertible sampling is given by* $cip^{-1}(i, m', x) = (m', dom^{-1}(i,x))$*.*

Theorem 4.5 *The Blum-Goldwasser scheme with fixed plaintext space is a IND secure simulatable encryption scheme.*

Proof. This is straight-forward to verify. The oblivious index generation directly implies the oblivious public-key generation, the invertible domain sampling implies the oblivious ciphertext generation, and the IND-CPA security was proved by Blum and Goldwasser [BG85]. □

We proceed to construct a simulatable family of trapdoor permutations based on the RSA assumption. We cannot use the standard family of RSA-permutations, as we cannot prove that it has oblivious public-key generation. If the oblivious public-key generation learns the factorization of n, the random-bits-faking algorithm would have to factor n, which is hopefully hard. If the oblivious public-key generation does not learn the factorization of n it would have to test in PPT whether n is a factor of two large primes, which we do not know how to do. We therefore need a modification. As a starting point we take the RSA sub-group collection (Assumption 2.6 on page 26). We do not know whether the RSA-sub-group family is simulatable either, but we can use the RSA-sub-group family to construct a simulatable family secure relative to Assumption 2.6 on page 26.

As discussed in Section 2.7.4, if we draw a random number from $\mathbf{Z}_{k\log(k)}$, then it will have two primefactors larger than 2^k except with probability at most $1/c\log(k)$ for some constant c. This means that if we pick $l = k/(\log\log(k)+\log(c))$ random numbers from $\mathbf{Z}_{k\log(k)}$, then at least one of them will have two primefactors larger than 2^k except with probability at most 2^{-k}. This gives rise to the following family.

Definition 4.7 *We define a family which we call the simulatable RSA-sub-group family:*

index/trapdoor generation:

An index will be l uniformly random $k\log(k)$-bit numbers $n = (n_1,\ldots,n_l)$ and the trapdoor will be be $\phi(n) = (\phi(n_1),\ldots,\phi(n_l))$. Furthermore a random element $e \in \mathrm{PRIMES}(2k\log(k))$ is chosen. The encryption key is (n,e) and the decryption key is $(n, d = (d_1,\ldots,d_l) = (e^{-1} \bmod \phi(n_1),\ldots,e^{-1} \bmod \phi(n_l))$.

domain sampling:

The domain of a public key (n,e) will be $\mathbf{Z}^*_{n_1} \times \cdots \times \mathbf{Z}^*_{n_l}$.

function, inverse function:

The function is given by $f_{(n,e)}(x_1,\ldots,x_l) = (x_1^e \bmod n_1,\ldots,x_2^e \bmod n_l)$, and it is well-known that the inverse of $f_{(n,e)}(x_1,\ldots,x_l)$ can be computed in PPT on $\mathbf{Z}^*_{n_1} \times \cdots \times \mathbf{Z}^*_{n_l}$ given just d.

Theorem 4.6 *The simulatable RSA-sub-group family is a simulatable family of trapdoor permutations relative to Assumption 2.6 on page 26.*

Proof. The invertible sampling requirements follow directly from Theorem 2.3 on page 28. We sketch how to reduce the one-wayness to the one-wayness of the RSA-sub-group family. Assume that we are given (n', e) as defined in the index/trapdoor generation in Assumption 2.6 on page 26. Then generate an index as defined in the index/trapdoor generation in Definition 4.7. Let $I \subseteq [l]$ be the index of those n_i where the second largest primefactor is larger than 2^k. Pick $i \in I$ uniformly at random (we have already observed that $|I| \geq 1$ except with negligible probability). Let $n'_i = n'$ and for $j \in [l] \setminus \{i\}$, let $n'_i = n_i$. Then $(n'_1,\ldots,n'_l,e)$ is distributed exactly as specified in Definition 4.7, and inverting $f_{(n'_1,\ldots,n'_l,e)}$ involves inverting $f_{(n,e)}$. □

In [CFGN96] a similar family of trapdoor permutations is constructed, using the following observation: A random $2k$-bit number is a product of two k-bit primes with probability

approximately $\frac{1}{k^2}$. Therefore the permutation $x \mapsto x^e \bmod n$ (for a prime $e > n$) is a weak trapdoor permutation, and they then appeal to a general results to obtain a family of trapdoor permutations [Yao82b, GIL+90]. To obtain the same level of security as for the simulatable RSA-subgroup family, where we have a good 'sub index' with probability $1-2^{-k}$, such an amplification involves choosing $\Theta(k^3)$ 'sub indices', giving a total index length of $\Theta(k^4)$ bits, where the index length of the simulatable RSA-sub-group family is $O(k^2 \log(k))$. Another improvement over the RSA based scheme in [CFGN96] is that in the protocol we only send a constant number of (descriptions of) trapdoor permutations, where $\Theta(k)$ are sent in [CFGN96]. All in all this yields an improvement of a factor $k^3/\log(k)$ in communication complexity. The price for some if this is that we use an extra round of communication.

4.6 Non-Committing Encryption from Trapdoor Permutations with Invertible Domain Sampling

We now describe how one can replace the oblivious index generation by essentially the key-generation protocol from [CFGN96] described in Section 4.1. Since the approach is described in [CFGN96], we will only provide a sketch, and the reason for sketching at all is that we want to describe an attack against the protocol as described in [CFGN96] and provide a fix, and we want to apply a small twist to get rid of the requirement of the common-domain assumption.

4.6.0.1 The Attack and a Fix

Recall the protocol from [CFGN96] as sketched in Section 4.1. If the receiver, R say, of the permutations (also the receiver of the communication protocol) is honest, but a sender P_i of the permutations is corrupt, the adversary may choose to let P_i use as inputs to the OT a correct trapdoor for g_0^i and garbage for g_1^i. When the adversary sees the set A as part of the permutations sent from R to S he can then determine the value of c, the bit indicating for which of the permutations R learns the trapdoors. If $i \notin A$, then the sender must have $c = 1$ and have detected P_i's fraud. If $i \in A$ the sender must have $c = 0$. In any case the adversary learns c and with probability $\frac{1}{2}$ even without being detected. But the bit c is a piece of information that the adversary should not be able to get. The simulator's freedom to set c after corruption is exactly what makes the simulation go through.

We solve this by requiring that a sender P_i in an OT always proves in zero-knowledge to the receiver that he inputs correct information to the OT. I.e. prove that there exists r_0 and r_1 such that $(g_0^i, t_0^i) = gen(r_0)$ and $(g_1^i, t_1^i) = gen(r_1)$ and that there exists a bitstring r' such that (t_0, t_1, r') is the inputs to the OT (r' being the random bits used in the OT). This will imply that except with negligible probability, P_i will have to supply correct trapdoors to both permutations or be disqualified. Since what has to be proved is an NP statement and R knows both witnesses r and r' we can use the protocol from Section 2.9.6. We use the protocol in its generic form in Fig. 2.9 on page 42. Notice that this means that the protocol is only zero-knowledge against an honest verifier. This is however sufficient in our case, as it is the honest parties that are interested in *not* learning the witnesses (the trapdoors). Normally, the use of ZK proofs like that in Section 2.9.6 leads to problems against adaptive adversaries

because of the rewinding needed to simulate the proofs. However, in this protocol, it happens to be the case that the simulator never needs to prove a statement for which it doesn't know a witness, and so rewinding is not needed. The simulator will always provide correct inputs for honest parties and as discussed above, it does not matter that a cheating verifier can learn the witnesses — the witnesses are the trapdoors, which the cheating verifier can already learn by cheating in the OTs. Furthermore, the simulator does not have to extract witnesses (which would again require rewinding): We are only interested in membership soundness as the witnesses can be learned by cheating in the OT.

4.6.0.2 The Twist

Having executed all the OTs, R would in the protocol from [CFGN96] compose the permutations from honest parties to obtained two permutations, one with a known trapdoor and one with an unknown trapdoor. This requires and produces common-domain trapdoors. Instead of composing we simply concatenate.

Let $g_c^1, \ldots, g_c^l$ be the permutations that was correctly received and for which the corresponding trapdoor was received. From these permutations R defines a permutation f_c, where $f_c(x^1, \ldots, x^l) = (g_c^1(x^1), \ldots, g_c^l(x^l))$. Let B be a hard-core predicate [GL89] for the family of trapdoor permutations used. Then we use the hard-core predicate $B(x^1, \ldots, x^l) = \bigoplus_{i=1}^{l} B(x^i)$ for f_c for encryption using the construction in [BG85].

4.6.0.3 Using the Key Generation in Our Protocol

After an execution of the key generation protocol an honest receiver R has two public keys where he knows the private key to exactly one of them, but where the adversary cannot tell which one he knows. This is exactly the scenario that the first round in the π_{WTE} creates. Thus, the π_{WTE} protocol can be run with the described protocol instead of oblivious index generation.

4.6.0.4 The NCE Scheme in [CFGN96] also Needs Invertible Domain Sampling

Finally we argue that the protocol in [CFGN96] also needs invertible domain sampling. The OT scheme used in [CFGN96] is from [GMW87] and proceeds as follows: The sender has bits b_0 and b_1 and the receiver has a bit c. The sender generates an index and trapdoor (i, t) for a family of trapdoor permutations and sends i to the receiver. The receiver computes $x_c \leftarrow dom_i$, $y_c \leftarrow f_i(x_c)$ and $y_{1-c} \leftarrow dom_i$ and sends (y_0, y_1) to the sender. The sender returns (c_0, c_1), where $c_j = B(f_i^{-1}(y_j)) \oplus b_j$ and B is a hard-core predicate for f_i. Obviously this scheme has the desired properties. The simulator learns both bits by choosing both y_0 and y_1 as $y_j = f_i(x_j)$. Note however, that on corruption of the receiver the simulator must show to the adversary random bits r_c and r_{1-c} such that r_c looks as bits used to sample x_c (which is easy) and r_{1-c} looks as bits used to sample y_{1-c} (which is only possible if the domain of f has invertible sampling). Therefore, the scheme in [CFGN96] based on trapdoor permutations also needs invertible sampling of the domains of the trapdoors. This means that trapdoor permutations with invertible domain sampling currently is the most general assumption under which non-committing encryption is known to exist.

Theorem 4.7 *If there exists families of trapdoor permutations with invertible domain sampling, in the sense of Definition 4.5 on page 174, then* $\mathcal{F}_{\mathrm{SMT}}$ *can be realized by an* n*-party protocol secure against a adversary corrupting at most* $n-1$*-parties.*

CHAPTER 5

Zero-Knowledge Proofs

Information is not knowledge.
— Albert Einstein

5.1 Universally Composable Zero-Knowledge Proofs of Membership

In this section we present an efficient protocol which realizes the functionality $\mathcal{F}_{\text{ZK-PM}}$ for zero-knowledge proof of membership and realizes the functionality $\mathcal{F}_{\text{ZK-PK}}$ for proof of knowledge with rewinding. Our notion of zero-knowledge proof of knowledge with rewinding basically generalizes the notion of concurrent zero-knowledge [DNS98] by Dwork, Naor and Sahai to consider an adaptive adversary and considering being verifier for many provers at the same time. Both generalizations give additional challenges with respect to simulating the internal state of corrupted parties respectively extracting witnesses for interleaved proofs.[1] Notice that we do not claim that a realization of the ideal functionality $\mathcal{F}_{\text{ZK-PK}}$ proved secure using rewinding composes concurrently (while preserving security). As discussed in Section 3.7 this is indeed most likely *not* the case. However, the functionality $\mathcal{F}_{\text{ZK-PK}}$ explicitly allows to carry out many proofs at the same time. This means that any protocol realizing $\mathcal{F}_{\text{ZK-PK}}$ will allow this too. Notice that a realization of $\mathcal{F}_{\text{ZK-PK}}$ could use some coordination between proofs. This is a relaxation w.r.t the model in [DNS98], where all proofs are assumed to be run independently.

The purpose of presenting the protocol is primarily to capture precisely the security of it in a manner that will be used in all subsequent chapters, as to allow for more modular proofs. In particular, our realization is by a known protocol by Damgård [Dam00]. This protocol is based on Σ-protocols, and it turns out to be a universally composable zero-knowledge proof of membership when based on non-erasure Σ-protocols. The same protocol realizes $\mathcal{F}_{\text{ZK-PK}}$ with rewinding. In the protocol we require that the length l of the challenges is such that 2^l

[1] This last point can be stated as follows: Where as [DNS98] only consider preservation of *zero-knowledge* under concurrent composition, we also consider the preservation of *proof of knowledge* under concurrent composition. We later discuss why this is not property is not trivially guaranteed.

Protocol $\pi^{\Sigma}_{\text{DAM-ZK}}$

The protocol runs with two parties[a], the prover P and the verifier V, in the $\mathcal{F}_{\text{PRS}}^{\text{commit},\texttt{prs}}$-hybrid model,[b] with P acting as the requester of the key of V for a trapdoor commitment scheme commit — K will be the key and t the trapdoor — and the private reference string s for the Σ-protocol.

activation:

When a party is activated for the first time, it request the commitment key and private reference string for Σ of V and waits for output (K, s). Until this happens, ignore all inputs.

Activated on $(x, w) \in R$ (we drop the proof id's *pid* in the presentation) the protocol proceeds as follows:

1. The prover generates $a \leftarrow A(x, s, w; r_A)$ for uniformly random bits r_A.
2. The prover generates $c \leftarrow \text{commit}_K(a; r_c)$ and sends c to the verifier.
3. The verifier sends uniformly random $e \in \{0,1\}^l$ to the prover.
4. The prover computes $z \leftarrow Z(x, s, w, r_A, e)$ and sends (a, r_c, z) to the verifier.
5. The verifier accept iff $c = \text{commit}_K(a; r_c)$ and $B(x, s, a, e, z) = 1$.

If the protocol is run as a proof of knowledge we require that V does not output (pid, b) for any proof as long as there are ongoing proofs. For the n party version, we require that no honest party outputs (pid, b) as long as *any* honest party has an on-going proof!

[a] The extension to n parties is trivial as all pairs of parties act independently

[b] Here we use commit, $\texttt{prs}$ to denote the generator which generates a key for the commitment scheme and runs the generator for the PRS $\texttt{prs}$ of the Σ-protocol. Notice that by Definition 2.12 on page 20 we are guaranteed *statistical hiding* and *trapdoor opening* for the keys of corrupted parties even though they might not be random. This is a vital point.

Figure 5.1: The protocol $\pi^{\Sigma}_{\text{DAM-ZK}}$.

is super-polynomial. The protocol proceeds as described in Fig. 5.1.[2]

Theorem 5.1 *If* commit *is a trapdoor commitment scheme and* $\Sigma = (A, l, Z, B, \texttt{hvs}, \texttt{rbs}, \texttt{xtr}, \texttt{prs})$ *is a non-erasure* Σ*-protocol with special membership soundness with private reference string* $\texttt{prs}$ *and super-polynomial challenge space, for a relation* R*, then* $\pi^{\Sigma}_{\text{DAM-ZK}}$ *realizes* $\mathcal{F}^{R}_{\text{ZK-PM}}$ *in the* $\mathcal{F}_{\text{PRS}}^{\text{commit},\texttt{prs}}$*-hybrid model. If the* Σ*-protocol has special knowledge soundness with* $\texttt{prs}$ *and the protocol is run as a proof of knowledge, then* $\pi^{\Sigma}_{\text{DAM-ZK}}$ *realizes* $\mathcal{F}_{\text{ZK-PK}}$ *with rewinding in the* $\mathcal{F}_{\text{PRS}}^{\text{commit},\texttt{prs}}$*-hybrid model.*

Proof. To prove the first part of the claim, we construct an interface $\mathcal{S}_{\text{DAM-ZK}}$ for the exe-

[2] In Fig. 5.1 the private reference string s is mentioned explicitly. For the rest of the chapter we only mention s when we need it explicitly. In all other situation it will be implicitly present as $x := (x, s)$.

cution $\mathrm{IDEAL}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}^R_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ with an environment $\mathcal{Z}$ for which

$$\Pr[\mathrm{IDEAL}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}^R_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}} = \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}^R_{\text{ZK-PM}}\text{''}] \overset{c}{\approx} 0 \ .$$

In other words, let us assume that $x \in L(R)$ for all honest parties and $x \in S(R)$ for all corrupted parties.

The simulator is given in Fig. 5.2 on the following page. This simulator is almost identical to the simulator in [Dam00], but some modifications are introduced as to not use the trapdoor t until the verifier is corrupted. It was not necessary to consider this issue in [Dam00] as only static security was considered.

It is straightforward to prove that the simulation of the communication in the protocol is (statistically close to) perfect, using the properties of the non-erasure Σ-protocol and the trapdoor commitment scheme. What remains to show is that the probability that $\mathrm{IDEAL}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}^R_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ halts with output $\text{``}\mathtt{C} \rightsquigarrow \mathcal{F}^R_{\text{ZK-PM}}\text{''}$ is negligible. This comes down to proving that $\mathcal{S}_{\text{DAM-ZK}}$ only inputs $(pid, 1)$ to $\mathcal{F}_{\text{ZK-PM}}$ if V was activated on (pid, x) and $x \in L(R)$.

When $(pid, 1)$ is input on the SIT of $\mathcal{F}_{\text{ZK-PM}}$ it is always the case that a proof was accepted by $\mathcal{S}_{\text{DAM-ZK}}$ for the instance x in (pid, x). We prove that $x \in L(R)$ in that case. We do the proof of the second part of the theorem first, by proving that we can extend $\mathcal{S}_{\text{DAM-ZK}}$ to a rewinding simulator which can extract a witness and input (pid, w) to $\mathcal{F}_{\text{ZK-PK}}$. We denote it by $\mathcal{S}_{\text{DAM-ZK}}{}^{\curvearrowleft}$.

When $\mathcal{S}_{\text{DAM-ZK}}$ accepts a proof for (pid, x), then $\mathcal{S}_{\text{DAM-ZK}}$ has just received a value (pid, a, r_c, z) for an activation on (pid, x) started while the prover was corrupted and where $c = \mathrm{commit}_K(a; r_c)$ and $B(x, a, e, z) = 1$. Then $\mathcal{S}_{\text{DAM-ZK}}{}^{\curvearrowleft}$ proceeds as follows: It rewinds the state of the simulation to the point where (pid, e) was sent by $\mathcal{S}_{\text{DAM-ZK}}$ (on behalf of the honest V) and sends a new random challenge $e' \in \{0,1\}^l$. Formally $\mathcal{S}_{\text{DAM-ZK}}{}^{\curvearrowleft}$ outputs $(\mathtt{store}, pid)$ just before sampling and sending the challenge e for a proof with proof id pid. Rewinding then means outputting $(\mathtt{rerun\text{-}from}, pid)$. Then it runs until one of the following three events occur: 1) A value (pid, a', r'_c, z') is received, 2) V is corrupted, or 3) $\mathcal{Z}$ outputs a guess. It repeats this experiment with fresh e' until the event 1) occurs and $c = \mathrm{commit}_K(a'; r'_c)$ and $B(a', e', z') = 1$. When this happens, if for the newly generated (a', r'_c, z') we have that $e \neq e'$ and $a' = a$, then we are done, as we know that $x \in S(R)$ (by the required IO restriction from $\mathcal{Z}$) and we have computed a membership witness (x, a, e, z, e', z') from which we can then extract a witness using $\mathtt{xtr}$ — if not, then (x, a, e, z, e', z') is a value which allows us to win in the game $\mathrm{SKS}_{\Sigma,\mathtt{prs},B'}$ (see Section 2.9) for an adversary B' which we describe below. Therefore, give up the simulation and output the break of the extractor (x, a, e, z, e', z'). If $a' \neq a$, then $\mathcal{S}_{\text{DAM-ZK}}{}^{\curvearrowleft}$ gives up and outputs (c, a, r_c, a', r'_c), which is a double opening of the commitment c. If $e = e'$ and $a' = a$, then $\mathcal{S}_{\text{DAM-ZK}}{}^{\curvearrowleft}$ gives up too and outputs $\perp$.

Notice that during the reruns there might by other accepted proofs pid'. If this happens, then we first have to extract witness for pid' before we continue. In particular, we have to stop before we observed one of the three specified conditions for stopping for the proof pid. If we do not extract a witness for pid' to input to $\mathcal{F}_{\text{ZK-PK}}$, then $\mathcal{F}_{\text{ZK-PK}}$ outputs $(pid', 0)$, where

Interface $\mathcal{S}_{\text{DAM-ZK}}$

initialization:

Simulate $\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}$ as follows: For a corrupt verifier, receive (K,t) and (s,t_s) from $\mathcal{Z}$ and for a honest verifier, generate (K,t) and (s,t_s) honestly. We will not use t until V is corrupted and we never use t_s, except to show to $\mathcal{Z}$ when the party is corrupted. We will not mention s below, but use the convention that $x := (x,s)$.

activation:

The honest verifier is always simulated according to the protocol. If a proof with proof id *pid* is accepted from a corrupted prover, then $\mathcal{S}_{\text{DAM-ZK}}$ simply inputs $(pid,1)$ to $\mathcal{F}_{\text{ZK-PM}}^{R}$ to make it output $(pid,1)$ to V.[a]

As long as the prover is honest each activation, on $x \in L(R)$, is simulated as follows:

1. Generate a random challenge $\overline{e} \in \{0,1\}^l$ and generate $(\overline{a},\overline{z}) \leftarrow \mathtt{hvs}(x,e;\overline{r}_{\mathtt{hvs}})$ for uniformly random bits $\overline{r}_{\mathtt{hvs}}$. Note that the internal state r_A is not defined yet.
2. Compute $c = \text{commit}_K(\overline{a};\overline{r}_c)$ for uniformly random bits $\overline{r}_c$ and send c to the verifier. If $\mathcal{Z}$ corrupts P after this round the simulator proceeds as follows: It is given w such that $(x,w) \in R$ (from the conversation between P and $\mathcal{F}_{\text{ZK-PM}}^{R}$) and computes $r_A \leftarrow \mathtt{rbs}(x,w,\overline{e},\overline{r}_{\mathtt{hvs}})$ and presents the internal state $(r_A,\overline{r}_c)$ to $\mathcal{Z}$. Note that $\overline{a} = A(x,w;r_A)$ and $c = \text{commit}_K(\overline{a};r_c)$ as desired.
3. If the verifier is honest it is simulated by sending the challenge $e = \overline{e}$. If the verifier is corrupted the simulator receives a challenge $e \in \{0,1\}^l$ from $\mathcal{Z}$.
4. We look at two cases:
   (a) If the verifier is honest, then $e = \overline{e}$ and then the prover will send $(\overline{a},\overline{r}_c,\overline{z})$ as generated in the first round. If P is corrupted by $\mathcal{Z}$ after this round, then the simulator proceeds as in Step 2 to simulate the internal state.
   (b) If the verifier is corrupted, then except with negligible probability $e \neq \overline{e}$ and we cannot proceed as above. Therefore the simulator generates $(a,z) \leftarrow \mathtt{hvs}(x,e;r_{\mathtt{hvs}})$. Since the verifier is corrupted we now start using the trapdoor t of K, which is used to compute r_c such that $c = \text{commit}_K(a;r_c)$. Then send (a,r_c,z) to the verifier. If P is corrupted by $\mathcal{Z}$ after this round the simulator gets as input w such that $(x,w) \in R$. It then computes $r_A \leftarrow \mathtt{rbs}(x,w,e,r_{\mathtt{hvs}})$ and presents the internal state (r_A,r_c) to $\mathcal{Z}$.

[a] This might of course also result in $\text{IDEAL}^{\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}}^{R},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ halting with output ``C $\rightsquigarrow \mathcal{F}_{\text{ZK-PM}}^{R}$''. We return to that point later.

Figure 5.2: The interface $\mathcal{S}_{\text{DAM-ZK}}$ for the proof of Theorem 5.1 on page 180.

the protocol would have output $(pid',1)$ when the proof was accepted. Therefore $\mathcal{Z}$ will catch us simulating, so the run after that will not be distributed as the first run, rendering the rerunning useless. But, we cannot start extracting for pid'.[3] Clearly this would start a

[3] The necessity to extract witnesses of proof accepted during the rewindings is what makes it none trivial to maintain the *proof of knowledge* property under concurrent composition. In [DNS98] the case of maintaining the *zero-knowledge* property under concurrent composition is treated and a comment is made to the effect

nasty regression. We could of course give up, output $(\texttt{rerun-from}, pid)$ and try again until one of the three specified conditions is observed for pid. It is however easy to construct a scheduling of messages that would render that approach impossible.

The solution to this apparent problem is that if the protocol is run as a proof of knowledge, then V does not output (pid, b) for any proof as long as there are ongoing proofs. This ensures that from a challenge e is sent until one of the three specified conditions occur, no other proof is accepted. So, if $\mathcal{S}_{\text{DAM-ZK}}{}^{\curvearrowleft}$ does not give up, then it always computes a witness to input to $\mathcal{F}_{\text{ZK-PK}}$ to make it output $(pid, 1)$ to V. It is therefore enough to prove that $\mathcal{S}_{\text{DAM-ZK}}{}^{\curvearrowleft}$ is expected PPT and gives up only with negligible probability.

We prove the last claim first, assuming that the simulation with rewinding $\text{IDEAL}^{\curvearrowleft,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}}^{R},\mathcal{S}_{\text{DAM-ZK}}{}^{\curvearrowleft},\mathcal{Z}}$ is expected PPT. If the simulation with rewinding is expected PPT, then in particular the expected number of challenges e' generated during the reruns is bounded by a polynomial. Since 2^l is required to be super-polynomial this means that the probability that $e' = e$ for any of the challenges is negligible, if e is independent of the e''s. Since the sampling of e' does not depend on e and the reruns are independent of e, because e was originally sampled independently of the simulation *after* the configuration was stored, this is indeed the case. Therefore, if there is a non-negligible probability that $\mathcal{S}_{\text{DAM-ZK}}{}^{\curvearrowleft}$ cannot compute a witness, then with a non-negligible probability $\mathcal{S}_{\text{DAM-ZK}}{}^{\curvearrowleft}$ outputs a double opening under commit or a break of the extractor. Since the reruns performed by $\mathcal{S}_{\text{DAM-ZK}}{}^{\curvearrowleft}$ are stopped if V is corrupted, $\mathcal{S}_{\text{DAM-ZK}}{}^{\curvearrowleft}$ never uses the trapdoor of K or s up till the double opening is output. This means that from the simulation with rewinding we can construct an expected PPT algorithm which takes K and s as input, runs with those keys as the keys K and s used by the honest V, and outputs a double opening under K or a break of the extractor with non-negligible probability, a contradiction to either the computational binding of the trapdoor commitment scheme or the computational special knowledge soundness of Σ with private reference string $\texttt{prs}$.

Notice that the reduction to the computational binding of the commitment scheme (or the computational special knowledge soundness) is *expected* PPT, as it involves rerunning for an expected polynomial number of times. The definitions of security requires the adversary to be (strict) PPT. We can however deal with this as follows: The success of the constructed adversary, let us call the adversary B, is bounded away from 0 by some polynomial $q(k)$ on infinitely many k. Since B is expected PPT there exists a polynomial $p(k)$ bounding the expected running time of $B(k)$. Let $B'(k)$ be the adversary which runs $B(k)$ for $2\lceil \frac{1}{q(k)} p(k) \rceil$ steps and outputs as follows: If $B(k)$ outputs within said number of steps, then use the output of $B(k)$. Otherwise output a dummy value. Let $r(k)$ be the probability that $B(k)$ do not output in $2\lceil \frac{1}{q(k)} p(k) \rceil$ steps. Since $r(k) \cdot 2\lceil \frac{1}{q(k)} p(k) \rceil \leq p(k)$ it follows that $r(k) \leq \frac{1}{2} q(k)$,

that proof of knowledge is trivially maintained. Though the authors of [DNS98] do not formalize what is meant by this, the comment shows that what is considered is *not* the notion that we consider here, where also witnesses for all proofs accepted in the rewindings must be produced by the simulator. That the stronger notion considered here is indeed the desired notion in some contexts is demonstrated by Chapter 8, where $\pi_{\text{DAM-ZK}}$ is used for proving plaintext knowledge. This is done in a manner such that the witnesses (plaintexts) are needed to simulate the outer protocol which $\pi_{\text{DAM-ZK}}$ is run as a part of. Therefore, to rewind and rerun $\pi_{\text{DAM-ZK}}$ in the particular context one has to produce the witnesses of proofs accepted during the rerunning, to be able to rerun the outer protocol.

so the probability that $B'(k)$ outputs a double opening or break of the extractor is at least $q(k) - \frac{1}{2}q(k) = \frac{1}{2}q(k)$ for infinitely many k, and $\frac{1}{2}q(k)$ is a polynomial.

To prove that the simulation with rewinding runs in expected PPT we define a number of events and corresponding probabilities associated to the main simulation done by $\mathcal{S}_{\text{DAM-ZK}}$. We divide the execution into periods. Period l is defined to begin with the simulator $\mathcal{S}_{\text{DAM-ZK}}$ sending the l'th challenge to $\mathcal{Z}$, and the period ends when the environment replies with a correct value (a, r_c, z) or the entire execution halts because $\mathcal{Z}$ terminates, whatever comes first.

Let E_l denote the event that the l'th challenge is answered correctly by the environment — and that the reruns begin. Let (s, l) denote the event that the execution is in state s when the l'th challenge is sent by $\mathcal{S}_{\text{DAM-ZK}}$. Let $T_l(s)$ be the expected running time of period l given that it is in state s at the beginning of period l, let $T_{l,-}(s)$ denote the expected running time of period l given that the l'th challenge is not answered correctly by the environment, and let $T_{l,+}(s)$ denote the expected running time of period l given that the l'th challenge is answered correctly by the environment. Given that E_l occurs the expected time spend rerunning for the l'th challenge is no more than

$$
\begin{aligned}
&\sum_s \Pr[(s,l)|E_l] \left(\sum_{i=1}^{\infty} (1 - \Pr[E_l|(s,l)])^{i-1} \Pr[E_l|(s,l)]((i-1)T_{l,-}(s) + T_{l,+}(s)) \right) \\
=&\sum_s \frac{\Pr[(s,l)|E_l] \Pr[E_l|(s,l)]}{1 - \Pr[E_l|(s,l)]} \sum_{i=1}^{\infty} (iT_{l,-}(s) + (T_{l,+}(s) - T_{l,-}(s)))(1 - \Pr[E_l|(s,l)])^i \\
=&\sum_s \Pr[(s,l)|E_l] \left(\frac{T_{l,-}(s)}{\Pr[E_l|(s,l)]} + (T_{l,+}(s) - T_{l,-}(s)) \right) \\
=&\sum_s \frac{\Pr[(s,l)|E_l]}{\Pr[E_l|(s,l)]} ((1 - \Pr[E_l|(s,l)])T_{l,-}(s) + \Pr[E_l|(s,l)]T_{l,+}(s)) \\
=&\sum_s \frac{\Pr[(s,l)|E_l]}{\Pr[E_l|(s,l)]} T_l(s) = \frac{1}{\Pr[E_l]} \sum_s \Pr[(s,l)] T_l(s) = \frac{1}{\Pr[E_l]} T_l \ ,
\end{aligned}
$$

where the second equation follows by a standard series.

This means that the expected running time for each challenge l is $\Pr[E_l](\frac{1}{\Pr[E_l]}T_l) = T_l$. In particular it will be smaller than the expected running time T of $\text{IDEAL}^{\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}^R_{\text{ZK-PM}}, \mathcal{S}_{\text{DAM-ZK}}, \mathcal{Z}}$. Since $\mathcal{Z}$ is required to be (strict) PPT the total number of proofs is bounded by a polynomial $p(k)$, and so the total expected running time is bounded by $p(k)T$, which is polynomial.

To prove the first part of the theorem it is then enough to observe that we proved above that if $\mathcal{S}_{\text{DAM-ZK}}$ could rewind then it could compute a membership witness for each accepted proof. By definition this implies that $x \in L(R)$ as we know that $x \in S(R)$. The reason why we do not need the protocol to be run as a proof of knowledge for the first result is that in the reruns — which are run off-line for the sake of proof — we do not need witnesses for proofs accepted in the rewinding, we just input $(pid, 1)$ to $\mathcal{F}_{\text{ZK-PM}}$ as $\mathcal{S}_{\text{DAM-ZK}}$ would have done, and the fact that we *could* have extracted a witness guarantees that this is sound. □

Arguing the soundness of the interface $\mathcal{S}_{\text{DAM-ZK}}$ in the above proof we had to argue that if it input $(pid, 1)$ to $\mathcal{F}_{\text{ZK-PM}}$, then pid is the proof id for a true statement. We did this

by arguing that when a proof from a corrupted party is accepted, it holds that *if* we could rewind the entire execution of $\pi^{\Sigma}_{\text{DAM-ZK}}$ and rerun it, *then* we could extract a witness for the statement. This is somewhat stronger than just proving that only true statements are accepted. This property comes in handy later on, so we give it a name. We say that $\pi^{\Sigma}_{\text{DAM-ZK}}$ is a zero-knowledge proof of membership with off-line extraction. We call this property off-line extraction to oppose it to on-line extraction, where the interface can extract a witness for all accepted proofs during the simulation. Notice that an interface with on-line extraction can be used to prove that a protocol realizes $\mathcal{F}_{\text{ZK-PK}}$.

In the following we let $\mathcal{S}_{\text{DAM-ZK}}$ denote both the simulator for $\text{IDEAL}^{\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ and the rewinding simulator for $\text{IDEAL}^{\curvearrowleft,\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}}\curvearrowleft,\mathcal{Z}}$. Which is meant will always be clear from the context. Assume that a Σ-protocol Σ only has special knowledge soundness relative to some $\mathcal{I}$. When running $\text{IDEAL}^{\curvearrowleft,\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ for Σ it might then happen that a membership witness is computed, but `xtr` cannot cash it in for a witness. We will however recall for later use that if $\mathcal{S}_{\text{DAM-ZK}}$ fails, then at least it can present us with a break of the extractor `xtr` of Σ. We capture this as a corollary.

Corollary 5.1 *Under the conditions on Σ in Theorem 5.1 on page 180, except possibly that Σ has soundness relative to $\mathcal{I}$, if*

$$\Pr[\text{IDEAL}^{\mathcal{F}^{\text{commit},\mathit{prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}} = \text{``}H \rightsquigarrow \mathcal{F}^{R}_{\text{ZK-PM}}\text{''}] \overset{\text{c}}{\approx} 0 \ ,$$

then

$$\text{IDEAL}^{\curvearrowleft,\mathcal{F}^{\text{commit},\mathit{prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}} \overset{\text{c}}{\approx} \text{IDEAL}^{\mathcal{F}^{\text{commit},\mathit{prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}} \ ,$$

or with a non-negligible probability $\text{IDEAL}^{\curvearrowleft,\mathcal{F}^{\text{commit},\mathit{prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ *outputs a break of the extractor of Σ, where the instance of the break is one of the instances input by $\mathcal{Z}$.*[4]

Proof. Assume first that $\text{IDEAL}^{\curvearrowleft,\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ outputs break of the extractor `xtr` for one of the instances input by $\mathcal{Z}$ with a non-negligible probability. Then we are done. If not, consider the execution $\text{IDEAL}^{\curvearrowleft,\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$. It is defined by running a main copy of $\text{IDEAL}^{\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ which defines the output of $\text{IDEAL}^{\curvearrowleft,\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ if it reaches termination. Each time a proof is accepted, some reruns are done. Since $\mathcal{Z}$ is not allowed to terminate by $\mathcal{S}_{\text{DAM-ZK}}$ in these reruns ($\mathcal{Z}$ terminating is exactly one of the criteria for rerunning) and $\mathcal{S}_{\text{DAM-ZK}}$ does not input to $\mathcal{F}^{R}_{\text{ZK-PK}}$ in the reruns, the only way that the reruns can change the output distribution is that $\mathcal{S}_{\text{DAM-ZK}}$ gives up or that $\mathcal{Z}$ violates the IO behavior of $\mathcal{F}^{R}_{\text{ZK-PK}}$ (which makes $\text{IDEAL}^{\curvearrowleft,\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ terminate with output ``$\mathtt{H} \rightsquigarrow \mathcal{F}^{R}_{\text{ZK-PK}}$''). We have assumed that $\mathcal{S}_{\text{DAM-ZK}}$ does *not* give up, except with negligible probability. It

[4] It is not a mistake that we do not require $\Pr[\text{IDEAL}^{\curvearrowleft,\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}^{R}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}} = \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}^{R}_{\text{ZK-PM}}\text{''}] \overset{\text{c}}{\approx} 0$ in the premise. It is part of the result.

is therefore enough to prove that $\Pr[\text{IDEAL}^{\curvearrowleft,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}^R_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}} = \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}^R_{\text{ZK-PM}}\text{''}] \overset{\text{c}}{\approx} 0$. We claim that this follows from $\Pr[\text{IDEAL}^{\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}^R_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}} = \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}^R_{\text{ZK-PM}}\text{''}] \overset{\text{c}}{\approx} 0$. This is so because the view $\mathcal{Z}$ has of each rerun in $\text{IDEAL}^{\curvearrowleft,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}^R_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ is *identical* to the view it has in $\text{IDEAL}^{\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}^R_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$, as fresh random bits are used for each rerun. Therefore the probability of *any* event[5] in $\text{IDEAL}^{\curvearrowleft,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}^R_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ is at most a polynomial factor higher than in $\text{IDEAL}^{\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}^R_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$, the factor being given by the number of reruns. Since a polynomial times a negligible function is a negligible function, the claim follows. □

By inspection of the proof of Theorem 5.1 on page 180 we see that `rbs` is only used when a party is corrupted after simulating a proof. Since all parties are corrupted before the protocol execution when run under $\mathcal{IO}_{\text{static}}$ it follows that `rbs` is never used by the simulator, so the Σ-protocol does not need to be non-erasure.

Corollary 5.2 *If* commit *is a trapdoor commitment scheme and* $\Sigma = (A, l, Z, B, \mathtt{hvs}, \mathtt{xtr}, \mathtt{prs})$ *is a* Σ*-protocol with special membership soundness with private reference string* `prs` *and super-polynomial challenge space, for a relation* R*, then* $\pi^{\Sigma}_{\text{DAM-ZK}}$ *realizes* $\mathcal{F}^R_{\text{ZK-PM}}$ *under* $\mathcal{IO}_{\text{static}}$*. If the* Σ*-protocol has special knowledge soundness with* `prs` *and the protocol is run as a proof of knowledge, then* $\pi^{\Sigma}_{\text{DAM-ZK}}$ *realizes* $\mathcal{F}_{\text{ZK-PK}}$ *with rewinding under* $\mathcal{IO}_{\text{static}}$*.*

Corollary 5.3 *Under the conditions on* Σ *in Theorem 5.2, except possibly that* Σ *has soundness relative to* $\mathcal{I}$*, if*

$$\Pr[\text{IDEAL}^{\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\langle\mathcal{F}^R_{\text{ZK-PM}},\mathcal{IO}_{\text{static}}\rangle,\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{F}^R_{\text{ZK-PM}}\text{''} \cup \text{``}H \rightsquigarrow \mathcal{IO}_{\text{static}}\text{''}] \overset{\text{c}}{\approx} 0 \ ,$$

then

$$\text{IDEAL}^{\curvearrowleft,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\langle\mathcal{F}^R_{\text{ZK-PK}},\mathcal{IO}_{\text{static}}\rangle,\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}} \overset{\text{c}}{\approx} \text{IDEAL}^{\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\langle\mathcal{F}^R_{\text{ZK-PM}},\mathcal{IO}_{\text{static}}\rangle,\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}} \ ,$$

or with a non-negligible probability $\text{IDEAL}^{\curvearrowleft,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\langle\mathcal{F}^R_{\text{ZK-PK}},\mathcal{IO}_{\text{static}}\rangle,\mathcal{S}_{\text{DAM-ZK}},\mathcal{Z}}$ *outputs a break of the extractor of* Σ*, where the instance of the break is one of the instances input by* $\mathcal{Z}$*.*

[5] Even one that cannot be detected in PPT.

CHAPTER 6

Threshold Function Sharing

If liberty and equality, as is thought by some, are chiefly to be found in democracy, they will be best attained when all persons alike share in government to the utmost.
— Aristotle

6.1 Introduction

In this section we are going to introduce some well-known techniques from threshold cryptography and prove that they are secure in the UC framework. Assume that we have some function family (gen, f), where gen generates a public key pk and a secret key sk and f for each secret key defines a PPT computable function f_{sk}. A simple function sharing scheme for (gen, f) for n parties is basically made out of a key sharing algorithm, an algorithm for generating evaluation shares and an algorithm for combining evaluation shares. For a key pair $(pk, sk) \in gen$ the key-sharing algorithm will break sk up into n key shares $sk_1, \ldots, sk_n$. There is a threshold t associated to the scheme, with the property that if one is given t of the key shares sk_i, then one has absolutely no information about sk, but if one is given $t+1$ key shares, then one can compute or, as it is called, reconstruct sk. This means that the key is perfectly hidden from an adversary which can corrupt at most t parties and that if at least $t+1$ parties stay honest, then the key will not be lost. But we also want to use the key. We want that if at least $t+1$ parties are honest and they agree on evaluating f_{sk} on an input $x \in \{0,1\}^*$, then they will all learn $f_{sk}(x)$. This should be done without revealing sk, so they cannot just reconstruct sk among them. If one of them is under the control of the adversary, or one of them is later broken into, that would reveal the key to the adversary. This is what the algorithm for generating evaluation shares is for. It takes as input sk_i and x and computes a so-called evaluation share $y_i = f_{sk_i}(x)$. These evaluation shares have the property that given $t+1$ of them from distinct parties, and for the same x, one can compute $y = f_{sk}(x)$ efficiently. Therefore the parties can compute $y = f_{sk}(x)$ by all computing $y_i = f_{sk_i}(x)$ and exchanging shares. Again $t+1$ honest parties are enough. These evaluation shares have the salient property that given the result $y = f_{sk}(x)$ it is possible to compute all n evaluation shares $y_1 = f_{sk_1}(x), \ldots, y_n = f_{sk_n}(x)$, while knowing the key shares sk_i for only

t parties! Since the result and t key shares is what we allow the adversary to know — we assumed that t shares contain no information about sk and that all parties learn $y = f_{sk}(x)$ — this shows that given the information the adversary is entitled to it can compute all other values that it sees on its own. Intuitively this means that the above approach to evaluating f_{sk} is zero-knowledge relative to its task.

Depending on what we let f_{sk} be we obtain one of several standard objects from threshold cryptography. If sk is a signature key and pk a verification key, we have a threshold signature scheme, where it takes $t + 1$ parties to sign and where t parties cannot sign on their own — unless the signature scheme itself is insecure of course. Threshold signature schemes play a prominent role in Chapter 7 in realizing Byzantine agreement efficiently. If we let sk be a decryption key and let pk be an encryption key, then we have a threshold encryption scheme, which play a prominent role in Chapter 8 and Chapter 10. If we let sk be the key for a pseudorandom function, in which case pk has no role, we get a threshold pseudorandom function, which can e.g. be used to realize efficient common coin-flip. We remark that threshold signatures and threshold encryption schemes can be realized without threshold computing any given *function*. Indeed any threshold realization of ElGamal signatures would not realize a function as ElGamal signatures are inherently probabilistic. Since we will not need to share any probabilistic functions in the following chapters we stick to the framework of simple function sharing.

In this chapter we present two techniques in detail, a threshold RSA technique by Shoup [Sho00] and a threshold pseudorandom function by Naor and Reingold [NR97]. Furthermore, as a contribution we show how to use the pseudorandom function by Naor and Reingold to efficiently provide the parties in a network with a shared source of randomness. threshold pseudorandom function in its original form is not efficient.

6.1.1 Threshold Signature Schemes

The field of threshold cryptography was founded by Desmedt [Des87] and in [DF89] Desmedt and Frankel presented the first threshold signature scheme, for ElGamal signatures. The scheme was however not secure against an active adversary. Later Gennaro, Jarecki, Krawczyk and Rabin [GJKR96b,GJKR96a] presented DSS and RSA threshold signature schemes secure against an active adversary. Later again Shoup [Sho00] presented a different threshold RSA signature scheme. We have several reasons for choosing this scheme to use in the following, one being that it is a simple function sharing scheme, which allows for a simple presentation, and others being mentioned below. Notice that in principle any threshold scheme could be realized using a general MPC protocol like [GMW87]. Such a solution is however very inefficient compared to the above mentioned specialized protocols.

6.1.1.1 Adaptive Security

All the above mentioned schemes are only secure against a static adversary. In [CGJ+99] Canetti, Gennaro, Jarecki, Krawczyk and Rabin presented threshold signature schemes which are adaptively secure. These signature schemes are presented in the so-called single inconsistent party framework and it can be verified that they are, therefore, also adaptively secure

realizations of $\mathcal{F}_{\text{T-sig}}$ in the UC framework. We introduce the single inconsistent party framework in Chapter 10, and if it is not already so, by the time we reach Lemma 10.1 on page 307 it will be trivial that the threshold signature schemes in [CGJ+99] are adaptively secure realizations of $\mathcal{F}_{\text{T-sig}}$ in the UC framework. However, as mentioned above, we will primarily be interested in threshold signature schemes for realizing Byzantine agreement efficiently. Since the techniques in [CGJ+99] make extensive use of a broadcast channel, we can therefore not use these schemes. In fact, the unfortunate situation seems to be that *all* adaptively secure threshold schemes make extensive use of a broadcast channels, except of course the trivial scheme presented in Section 3.8.6. However, this scheme has signatures of size kt, which invalidates all applications with the purpose of efficiency if t is large. Notice however that even this scheme might be efficient for a very small t. It is an interesting open problem to construct adaptively secure efficient threshold signature schemes without using broadcast.

Even though we have no realization of $\mathcal{F}_{\text{T-sig}}$ which is at the same time efficient and adaptively secure, in the remaining chapters we will still prefer the threshold signature functionality $\mathcal{F}_{\text{T-sig}}$ over $\mathcal{F}_{\text{sig}}$. Every time we use signatures as part of an adaptively secure protocol this will then leave us with a choice when we realize $\mathcal{F}_{\text{T-sig}}$. We can realize it using e.g. the statically secure threshold signature scheme from [Sho00] and obtain a composed protocol which is only statically secure, or we can realize $\mathcal{F}_{\text{T-sig}}$ using the trivial scheme in Section 3.8.6 and maintain adaptive security, at a considerable loss of efficiency. The reason that we prefer $\mathcal{F}_{\text{T-sig}}$ is that in none of the contexts where we use signatures can we obtain anything with respect to efficiency by exploiting that we are using individual signatures. The use of individual signatures through the use of the trivial threshold signature realization always gives us the optimal solutions, as far as our understanding of e.g. the Byzantine agreement problem stretches. We will not repeat this discussion each time we use $\mathcal{F}_{\text{T-sig}}$.

The above is by far a thorough survey on the work on on threshold signature schemes, for such see e.g. [Des97, CGJ+99, Sho00].

6.1.2 Threshold Pseudorandom Functions

The notion of pseudorandom function (Definition 2.4 on page 13) was introduced by Goldreich, Goldwasser and Micali [GGM86] and the notion has found innumerable applications. Remember that a pseudorandom function family is a function F taking as input a key K and an element x, we write $F_K(x)$, where for a random key the output of F_K cannot be distinguished from uniformly random values if one does not know the key.

One immediate application of pseudorandom functions is using them for implementing the random oracle model (Section 3.9): Consider a protocol setting with n parties. A random oracle with domain D can be thought of as an ideal functionality. After a party inputs $(\texttt{evaluate}, x)$, where say $x \in \{0,1\}^*$, the functionality returns a uniformly random value $r_x \in D$ to the party, and uses r_x for all queries on x. This functionality defines a uniformly random function from $\{0,1\}^*$ to D. Numerous protocol constructions are known that can be proved secure assuming that a random oracle is available. However, any implementation of such a protocol must also provide an implementation of the oracle. As discussed in Section 3.9, in practice, a hash function is often used to replace a random oracle, but then the implementation is only secure if an adversary can do no better with the hash function

than he could with the oracle. This is something that in general cannot be proved, but must be a belief based on heuristics — in fact, as discussed in Section 3.9, for some protocols, this belief is always wrong.

In contrast, pseudorandom functions can be used to implement random oracles without loss of security. This can be done by generating K at the beginning of the protocol and letting $r_x = F_K(x)$ when r_x is needed. It is however clearly necessary that no party should know the key K, since the output of a pseudorandom function only looks random to parties who do not have the key. Therefore the key — and therefore also the ability to evaluate the function — must be distributed among the parties. We are going to realize a pseudorandom function which allows for particular efficient use in many contexts. The key is a prime Q, where $P = 2Q + 1$ is also a prime, a random value x from the subgroup of quadratic residues Q_P, along with $2l$ random values $\{\alpha_{j,b}\}_{j=1,\ldots,l,b=0,1}$ from $\mathbf{Z}_Q$. The function family maps from the set of strings of length at most l to Q_P, and given $\sigma = (\sigma_1, \ldots, \sigma_m) \in \{0,1\}^{\leq l}$, the output is $x^{\prod_{i=1}^{m} \alpha_{i,\sigma_i}} \bmod P$. In Section 6.6, we prove that this function family is pseudorandom under the DDH assumption.

Using a simple function sharing scheme for evaluating the exponentiations $x \mapsto x^\alpha \bmod P$, with α being the secret key, this pseudorandom function can be realized with communication complexity $O(ln^2k)$ bits per evaluation, where k is the security parameter (for each exponent each party sends to each other party k bits). However, if used correctly, in many uses the communication complexity will be $O(n^2k)$ bits per evaluation. The clue is that the domain of the function is $\{0,1\}^{\leq l}$. Therefore one can obtain an efficient solution by always evaluating on strings $\sigma = \sigma'|b$, where $b \in \{0,1\}$ and where we already evaluated on σ'. Thereby each evaluation will only cost one distributed exponentiation. In particular, this can be used to implement a coin-flip protocol, which allows the parties to learn a common random value in each round.

6.1.2.1 Related Work

The coin-flip protocol described above is from [Nie02b] and is the currently most efficient coin-flip protocol for the standard model. A protocol with the same communication complexity was proposed by Cachin, Kursawe and Shoup in [CKS00], but the security of this protocol relies on the random-oracle assumption, an assumption that we would like to avoid, especially when, as we demonstrate here, the assumption is not necessary.

Also Canetti and Rabin [CR03] considers universally composable protocols for coin-flip which does not rely on random oracles. Our protocol is more efficient than their protocols, but as opposed to their protocols it is only statically secure. Our protocol meets the lower bound proved in [CR03].

Sharing pseudorandom functions has also been considered by Micali and Sidney [MS95]. Their protocol is however only efficient for a small number of parties. The notion of verifiable pseudorandom functions was introduced by Micali, Rabin and Vadhan [MRV99] and Dodis [Dod03] presented the first efficient realization of a distributed verifiable pseudorandom function.

6.2 Distributed Function Evaluation

We present a simple framework for distributed function evaluation. The section has two purposes. First it introduces multiparty computation in a simple setting and in a very crisp manner it further develops the quintessential idea behind our model of security, that of simulating a protocol given the information one is entitled to. Besides this the section introduces a functionality, which we will be using a lot, along with a realization of it using a simple function sharing scheme.

We introduce the concept of distributed function evaluation. We assume that we are given a function family (gen, f), where gen is a PPT generator which on input k generates a pair of values $(pk, sk) \leftarrow gen(k)$ called the public key and the secret key and f is function which can be computed in PPT. We could also consider letting let f take some randomness as input. We will not need this is the presentation, so we let f be deterministic for simplicity. The goal is to give a protocol for distributed evaluation of f_{sk}. We want to distribute sk between n servers, each of them receiving a secret share sv_i, in such a way that sk is kept secret, while the servers can cooperate to evaluate f_{sk} on any point x. The value pk is given to all parties to allow them to compute f_{pk}, which will typically be the inverse of f_{sk}. The function f_{pk} might however be arbitrary. Such a protocol has two so-called thresholds associated to it, the construction threshold c and the corruption threshold t. The construction threshold is basically the number of honest parties which is needed in order for the protocol to guarantee termination. The corruption threshold is the number of corrupted parties which can be tolerated without leaking any information about the secret key.

We require of the protocol for distributed evaluation of f that it is zero-knowledge relative to its task, namely that when it is invoked to compute $y = f_{sk}(y)$, then only y is revealed. In particular, sk is kept secret. It turns out that this is not always obtainable efficiently. We might allow the protocol to leak some insignificant information besides y. So to say allow the corrupted parties to learn more information than the honest parties, as long as it is not harmful for the application of the protocol. We call this information the allowable side information. We therefore let gen and f be of the following forms: $(pk, sk, sk_0) \leftarrow gen(k)$ and $(y, y_0) \leftarrow f_{sk}(x)$, where sk_0 and y_0 is the extra information learned by the corrupted parties. The functionality is given in Fig. 6.1 on the following page.

The extra information sk_0 and y_0 is allowed to leak to allow for more efficient implementations, or implementations at all, but should of course be of a form such that it does not compromise the usability of the functionality. If e.g. $sk_0 = sk$, we would hardly have any use of $\mathcal{F}_{\text{thresh}}^{gen,f,c}$. In Section 6.3.3 we will give an example of a function which we only know how to distribute efficiently if we allow some side information.

The termination conditions of the evaluation command deserves a few comments. We will be using such ideal functionalities in synchronous protocols. Therefore it seems unfortunate that the evaluate does not guarantee that the honest parties terminate in the same round. The current formulation is chosen because it allows to use the functionality in more protocols and because the parties can enforced simultaneous termination if desired. If the honest parties can guarantee to input in the same round, then they can simply wait for r rounds before they continue. At this point they are guaranteed that all other honest parties are ready to continue too. Furthermore, we do not guarantee termination if $c \geq n - t$. The reason for

Functionality $\mathcal{F}_{\text{thresh}}^{gen,s,c}$

The functionality runs with parties $P_1, \ldots, P_n$ and is parametrized by a key generator gen and a construction threshold $0 < c \leq n$. Furthermore it is parametrized by round complexity r for evaluations. All inputs to the functionality is revealed on the SOT. The functionality proceeds as follows:

key generation:

If in some round all honest parties input `init`, then generate $(pk, sk, sk_0) \leftarrow gen(k)$ and output (pk, sk_0) on the SOT. In a round determined by the adversary, output pk to all parties.

evaluation:

If in some round any honest party inputs (eid, x), where $eid \in \{0,1\}^*$ is an evaluation id and $x \in \{0,1\}^*$, then compute $(y, y_0) \leftarrow f(x)$ and output (eid, i, y, y_0) on the SOT, and in the first round where (eid, i) is input on the SIT, output (eid, y) to P_i.

If $n - t \geq c$ and if in some round c honest parties have input (eid, x), then output (eid, y) to all honest parties no later than after r rounds, i.e. behave as if the input (eid, i) is input on the SIT for all honest P_i after r rounds.

incorrect inputs:

If in the first round were an honest party inputs a non-trivial value some honest party do not input `init`, then break down. Also break down if an honest party inputs `init` twice or uses an evaluation id eid twice.

Figure 6.1: The threshold functionality for evaluating a function f with construction threshold c.

this of course is that $n - t$ is the maximal number of parties which are guaranteed to be honest and participating, and c is the number of participating parties needed to guarantee construction. If therefore $c \geq n - t$, clearly the adversary can always make the protocol hang.

Notice that the input to evaluation is $x \in \{0,1\}^*$. This is done for simplicity. If f has some domain where the input is required to be in, the parties can simply extend f to some dummy value, $\perp$ say, outside this domain and realize evaluations on such points by just outputting $\perp$. If the domain of f cannot be recognized in PPT, then one would have to extend the functionality with an IO restriction. Since we know of now application of threshold function sharing requiring this, it has been avoided for simplicity. We will look at realizations of $\mathcal{F}_{\text{thresh}}$ of a particular form described now.

Definition 6.1 *We call* (SingleToThresh, sf, R, Σ, Combine) *a simple function sharing scheme if* SingleToThresh *and* Combine *are PPT algorithm, sf is a PPT function, R is a PPT binary relation, Σ is a Σ-protocol for R, and the following hold:*

key sharing:

The component SingleToThresh *is an algorithm for sharing the secret key between the servers. It is given as input a key $(pk, sk, sk_0) \in gen$ and a construction threshold c and outputs a value $(pv, sv_1, \ldots, sv_n) =$ SingleToThresh(pk, sk, sk_0), where pv is a public value and sv_i is the secret value of server P_i.*

evaluation sharing:

The component sf is a function which takes as input pv and sv_i, an element $x \in \{0,1\}^*$ and outputs an evaluation share *$y_i = sf(pv, sv_i, x)$, where we again assume that sf is a function for simplicity.*

The use of this function will be that each server computes $y_i = sf(pv, sv_i, x)$ and sends y_i to all other servers. The evaluation shares will then have the property that given c of them one can compute $y = f_{sk}(x)$ using the function Combine*, which we return to below.*

share verification:

We are considering a setting where some servers might be corrupted. Therefore some of the shares sent by the servers might be incorrect. It is essential for a server to be able to root out such evaluation shares, so that they do not enter the computation of y. This is the purpose of R and Σ. The component R is for verifying that an evaluation share was computed correctly and Σ is a Σ-protocol for R with special membership soundness.

We require that when $y_i = sf(pv, sv_i, x)$, then $((pv, x, i, y_i), sv_i) \in R$. Where $((pv, x, i, y_i), sv_i) \in R$ basically expressed that y_i is a correctly computed evaluation share for server i and for the secret key corresponding to the public value pv. The intended use is that after P_i has sent y_i to P_j, then P_i gives a proof to P_j using Σ and the witness sv_i. The receiver P_j uses the instance (pv, x, i, y_i). As usual we let $L(R)$ be the set of (pv, x, i, y_i) for which there exists sv_i such that $((pv, x, i, y_i), sv_i) \in R$.

The Σ protocol is allowed to have special soundness relative to the key being randomly generated as in key sharing *and the Σ-protocol is allowed to be a private reference string Σ-protocol.*

share combining:

The component Combine *allows to take c correct evaluation shares for the same input point x and construct the output $y = f_{sk}(x)$. We require that if one has $(pv, x, i, y_i) \in L(R)$ for $i \in C$, where $|C| = c$, then $f_{sk}(x) = \text{Combine}(\{y_i\}_{i \in C})$.*

We now show how these components are combined into a protocol for realizing $\mathcal{F}_{\text{thresh}}^{gen,c}$. The protocol will run in the hybrid model with a functionality which distributes the key between the parties. Nothing of course prevents one from realizing this functionality and then plugging in the realization using the composition theorem. The functionality is given in Fig. 6.2 on the next page. It should not require any comments. We present the realization in the hybrid model with the ideal functionality $\mathcal{F}_{\text{ZK-PM}}^{R}$, which we realized in Chapter 5. One can always plug in the realization from that chapter to obtain a realization for the $(\mathcal{F}_{\text{thresh}}^{gen,c}, \mathcal{F}_{\text{PRS}}^{\text{commit},\mathtt{prs}})$-hybrid model, where commit is a trapdoor commitment scheme and `prs` is the possible private reference string for the Σ-protocol for share verification. The realization is given in Fig. 6.3 on page 195.

The realization follows the intuition given above for the various components. The protocol is clearly correct, and this is of course very fine. But we are equally interested in secrecy or zero-knowledge. We want to prove that the protocol do not leak any information except for what is it designed for, which is $f(x)$. This is what is modeled by the ideal functionality in

Functionality $\mathcal{F}_{\text{key-dist}}^{gen,c,\text{SingleToThresh}}$

The functionality runs with parties $P_1, \ldots, P_n$ and is parametrized by a key generator gen, an algorithm SingleToThresh for sharing a secret key and a construction threshold $0 < c \leq n$.

generate:

If all honest parties inputs `init` in the same rounds, then run $(pk, sk, sk_0) \leftarrow gen(k)$ and $(pv, sv_1, \ldots, sv_n) \leftarrow \text{SingleToThresh}(pk, sk, sk_0)$ and output pv and sv_i, for all corrupted parties P_i, on the SOT. If P_i is corrupted later, then output sv_i on the SOT. In a round specified by the adversary, output (pv, sv_i) to each P_i.

incorrect inputs:

If in the first round were an honest party inputs a non-trivial value some honest party do not input `init`, then break down. Also break down if an honest party inputs `init` twice or any other value than `init`.

Figure 6.2: The functionality $\mathcal{F}_{\text{key-dist}}$ for distributing a threshold key for gen, with construction threshold c, shared using SingleToThresh.

Fig. 6.1 on page 192 only outputting $f(x)$ — again except for the allowable side information given on the SOT.

We describe two criteria which allow us to argue that the protocol in Fig. 6.3 on the facing page is a realization of $\mathcal{F}_{\text{thresh}}^{gen,f,c}$. We assume that there exist two PPT algorithms $\mathcal{S}_{\text{keydist}}$ and $\mathcal{S}_{\text{eval}}$, called the key distribution simulator respectively the evaluation simulator. The existence of these algorithms is used to argue that the communication in the protocol contains no information extra to what the protocol is allowed to leak, which is pk and sk_0 in the key generation and y and y_0 in an evaluation. Besides this the corrupted parties of course see their own sv_i values. We require that all other public values can be generated given just those said.

Definition 6.2 *We say that a simple function sharing scheme* $(\text{SingleToThresh}, sf, R, \Sigma, \text{Combine})$ *is statistically secure for corruption threshold* t *if there exist PPT algorithms* $\mathcal{S}_{\text{keydist}}$ *and* $\mathcal{S}_{\text{eval}}$ *for which the following hold:*

key distribution simulation:

The algorithm $\mathcal{S}_{\text{keydist}}$ *takes as input* (pk, sk_0, C)*, where* $C \subseteq [n]$ *is a set with* $|C| = t$ *which denotes the indices of the corrupted parties. The output is* $(pv, \{sv_i\}_{i \in C}) \leftarrow \mathcal{S}_{\text{keydist}}(pk, sk_0, C)$*.*

We require that if we generate values as follows

$$\begin{aligned}(pk, sk, sk_0) &\leftarrow gen(k)\\ (pv, \{sv_i\}_{i \in C}) &\leftarrow \mathcal{S}_{\text{keydist}}(pk, sk_0, C)\ ,\end{aligned}$$

then the distribution of the value $(pk, pv, \{sv_i\}_{i \in C})$ *is statistically close to the distribution of* $(pk, pv, \{sv_i\}_{i \in C})$ *when the values are generated as*

$$\begin{aligned}(pk, sk, sk_0) &\leftarrow gen(k)\\ (pv, sv_1, \ldots, sv_n) &\leftarrow \text{SingleToThresh}(pk, sk, sk_0)\ .\end{aligned}$$

Protocol $\pi^{gen,f,c}_{\text{thres}}$

The protocol runs in the $(\mathcal{F}^{R}_{\text{ZK-PM}}, \mathcal{F}^{gen,c,s}_{\text{key-dist}})$-hybrid model with parties $P_1, \ldots, P_n$ and is parametrized by a generator gen and a construction threshold $0 < c \leq n$. We assume that $\mathcal{F}^{R}_{\text{ZK-PM}}$ has a round complexity of $r-1$ per proof. Party P_i proceeds as follows:

key generation:

On input `init` the party P_i inputs `init` to $\mathcal{F}^{R}_{\text{ZK-PM}}$ and $\mathcal{F}^{gen,c,\text{SingleToThresh}}_{\text{key-dist}}$ and waits until $\mathcal{F}^{R}_{\text{ZK-PM}}$ has output `ready` and $\mathcal{F}^{gen,\text{SingleToThresh},c}_{\text{key-dist}}$ has output (pk, pv, sv_i). Then output pk.

evaluation:

1. On input (eid, x) the server P_i computes the evaluation share $y_i \leftarrow sf(pv, sv_i, x)$ and sends (eid, x, y_i) to all parties.
2. In the next round P_i then inputs $((eid, i), P_i, (pv, x, i, y_i), sv_i)$ to $\mathcal{F}^{R}_{\text{ZK-PM}}$.
3. When P_i receives a message (eid', x', y'_j) from P_j, then P_i inputs $((eid', j), P_j, (pv, x', j, y_j))$ to $\mathcal{F}^{R}_{\text{ZK-PM}}$ and waits for an output of the form (eid', b). If $b = 1$, then P_i adds (j, y'_j) to a set $\texttt{Shares}_{eid',x'}$, if there was not already a pair of the form (j, y''_j) in $\texttt{Shares}_{eid',x'}$. These sets are initially all empty.

   If during this $|\texttt{Shares}_{eid',x'}| = c$, then let $\{y_j\}_{j\in I}$ be the evaluation shares in $\texttt{Shares}_{eid',x'}$ and then output $(eid, \text{Combine}(\{y_j\}_{j\in I})$.

   Notice that if $\mathcal{F}^{R}_{\text{ZK-PM}}$ outputs $(eid', 1)$, then $(pv, x', j, y_j) \in L(R)$ by the share verification property. So, we will always without failure have that $\text{Combine}(\{y_j\}_{j\in I}) = f_{sk}(x')$, by the share combining property.

Figure 6.3: The threshold protocol for key generator gen, function f and construction threshold c.

And this should be true for all C with $|C| = t$.

The existence of $\mathcal{S}_{\text{keydist}}$ basically says that the values seen by t (corrupted) parties could have been generated from (pk, sk_0) alone, which is the information allowed to leak by $\mathcal{F}_{\text{thresh}}$.

evaluation simulation:

The algorithm $\mathcal{S}_{\text{eval}}$ takes as input $(pk, sk_0, pv, \{sv_i\}_{i\in C}, x, y, y_0)$, which except for x, y and y_0, are the values seen by t parties after key distribution, including the allowable side information. The output is $\{y_i\}_{i\in[n]\setminus C}$.

We require for all C with $|C| = t$ and all $x \in \{0,1\}^*$ that if we generate

$$\begin{aligned}(pk, sk, sk_0) &\leftarrow gen(k)\\ (pv, sv_1, \ldots, sv_n) &\leftarrow \text{SingleToThresh}(pk, sk, sk_0)\\ (y, y_0) &\leftarrow f_{sk}(x)\\ \{y_i\}_{i\in[n]\setminus C} &\leftarrow \mathcal{S}_{\text{eval}}(pk, sk_0, pv, \{sv_i\}_{i\in C}, x, y, y_0)\ ,\end{aligned}$$

then for $i \in [n] \setminus C$ it holds that

$$y_i = sf(pv, sv_i, x)\ .$$

The existence of $\mathcal{S}_{\text{eval}}$ basically say that if one knows the values that t parties are entitled to see, plus the adversary's allowed side information, and if one knows $y = f_{sk}(x)$, again including the allowable side information y_0, then one can compute the evaluation share of all parties! It is of course trivial to compute $y_i = sf(pv, sv_i, x)$ for the t values sv_i one knows, but the condition says that one can also compute $y_i = sf(pv, sv_i, x)$ for the sv_i one does not know. By inspection of Fig. 6.3 on the page before one can see that this means that one could generate all the information of the protocol given just the result and the values t parties are entitled to see.

Intuitively Definition 6.2 on page 194 should guarantee the security against t corrupted parties, as argued during the definition. We now verify this.

Theorem 6.1 *If the simple function sharing scheme* $(gen, f, \text{SingleToThresh}, sf, R, \Sigma, \text{Combine})$ *is statistically secure with corruption threshold t, then the protocol π_{thres} in Fig. 6.3 on the page before statistically secure realizes the functionality $\mathcal{F}_{\text{thresh}}$ from Fig. 6.1 on page 192 under $\mathcal{IO}_{\text{static}}(t)$.*

Proof. In the following we only consider environments $\mathcal{Z}$ which corrupts at most t parties and we drop the explicit mentioning of $\mathcal{IO}_{\text{static}}(t)$ in the notation. We construct a simulator $\mathcal{S}_{\text{thresh}}$ for which

$$\text{IDEAL}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}^R_{\text{ZK-PM}}}_{\mathcal{F}_{\text{thresh}},\mathcal{S}_{\text{thresh}},\mathcal{Z}} \overset{\text{s}}{\approx} \text{HYB}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}^R_{\text{ZK-PM}}}_{\pi_{\text{thres}},\mathcal{Z}}$$

for all PPT environments $\mathcal{Z}$ where $\Pr[\text{IDEAL}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}^R_{\text{ZK-PM}}}_{\mathcal{F}_{\text{thresh}},\mathcal{S}_{\text{thresh}},\mathcal{Z}} = \text{``H} \rightsquigarrow \mathcal{F}_{\text{thresh}}\text{''}] \overset{\text{s}}{\approx} 0$. The simulator is given in Fig. 6.4 on the next page.

Assume that we are given any set of values $X = (pv, \{sv_i\}_{i \in C})$ and consider the following experiment: Run $\text{IDEAL}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}^R_{\text{ZK-PM}}}_{\mathcal{F}_{\text{thresh}},\mathcal{S}_{\text{thresh}},\mathcal{Z}}$, but at Eq. 6.1 on the facing page in Fig. 6.4 on the next page where the simulator computes $(pv, \{sv_i\}_{i \in C}) \leftarrow \mathcal{S}_{\text{keydist}}(pk, sk_0, C)$, insert X. If the values in $X = (pv, \{sv_i\}_{i \in C})$ are generated as $(pv, sv_1, \ldots, sv_n) \leftarrow \text{SingleToThresh}(pk, sk, sk_0)$, where (pk, sk_0) is the values generated by $\mathcal{F}_{\text{thresh}}$ in $\text{IDEAL}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}^R_{\text{ZK-PM}}}_{\mathcal{F}_{\text{thresh}},\mathcal{S}_{\text{thresh}},\mathcal{Z}}$, then the distribution of all the values generated by $\mathcal{S}_{\text{thresh}}$ is distributed exactly as in $\text{HYB}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}^R_{\text{ZK-PM}}}_{\pi_{\text{thres}},\mathcal{Z}}$ by the evaluation simulation property. If on the other hand the values are generated as $(pv, \{sv_i\}_{i \in C}) \leftarrow \mathcal{S}_{\text{keydist}}(pk, sk_0, C)$, then the distribution of all the values generated by $\mathcal{S}_{\text{thresh}}$ is distributed exactly as in $\text{IDEAL}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}^R_{\text{ZK-PM}}}_{\mathcal{F}_{\text{thresh}},\mathcal{S}_{\text{thresh}},\mathcal{Z}}$.

By the key generation simulation property it therefore follows that the statistical difference between the values generated by $\mathcal{S}_{\text{thresh}}$ in $\text{IDEAL}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}^R_{\text{ZK-PM}}}_{\mathcal{F}_{\text{thresh}},\mathcal{S}_{\text{thresh}},\mathcal{Z}}$ and the values generated in the protocol is exponentially small. Therefore it remains to argue that the outputs of the parties simulated by $\mathcal{S}_{\text{thresh}}$ are actually consistent with those output by $\mathcal{F}_{\text{thresh}}$. In the protocol $\text{HYB}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}^R_{\text{ZK-PM}}}_{\pi_{\text{thres}},\mathcal{Z}}$ the outputs seen by $\mathcal{Z}$ are those of the parties. In $\text{IDEAL}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}^R_{\text{ZK-PM}}}_{\mathcal{F}_{\text{thresh}},\mathcal{S}_{\text{thresh}},\mathcal{Z}}$ on the other hand, the outputs that $\mathcal{Z}$ sees come from $\mathcal{F}_{\text{thresh}}$. So, they might be different from those of the simulated parties, which would allow $\mathcal{Z}$ to distinguish.

To verify that the outputs of the simulated parties are consistent with the output of $\mathcal{F}_{\text{thresh}}$ basically involves checking that if no honest party inputs (eid, x), then no honest party will output (eid, y) and that if $n - t \geq c$ and c honest parties have input (eid, x), then

Interface $\mathcal{S}_{\text{thresh}}$

key generation:

The simulator first waits for $\mathcal{Z}$ to do the at most t static corruptions. Let C denote the set of corrupted parties. We have that $|C| \leq t$; For notational convenience assume that $|C| = t$.

Then the environment starts activating, and if in some round it inputs `init` to all honest parties, then $\mathcal{S}$ receives (pk, sk_0) from $\mathcal{F}_{\text{thresh}}$ and computes

$$(pv, \{sv_i\}_{i \in C}) \leftarrow \mathcal{S}_{\text{keydist}}(pk, sk_0, C) \ . \tag{6.1}$$

In that round it simulates inputs `init` to the simulated ideal functionalities $\mathcal{F}_{\text{key-dist}}$ and $\mathcal{F}_{\text{ZK-PM}}^{R}$, and simulates output $(pv, \{sv_i\}_{i \in C})$ on the SOT of $\mathcal{F}_{\text{key-dist}}$.

Instruct $\mathcal{F}_{\text{thresh}}$ to terminate when $\mathcal{Z}$ has specified that the simulated ideal functionalities $\mathcal{F}_{\text{key-dist}}$ and $\mathcal{F}_{\text{ZK-PM}}^{R}$ should terminate. If the simulated $\mathcal{F}_{\text{key-dist}}$ terminates, then simulate output (pk, sv_i) to P_i for $i \in C$ and simulate an output to all honest P_i. Notice that $\mathcal{Z}$ does not see which value is output and therefore does not see that there was not a correct sv_i in the output of the honest parties.

evaluation:

Evaluations are handled equivalently. The first time the environment inputs (eid, x) to an honest party, $\mathcal{S}_{\text{thresh}}$ receives $(y, y_0) = f(x)$ from $\mathcal{F}_{\text{thresh}}$. Then it computes

$$\{y_i\}_{i \in [n] \setminus C} \leftarrow \mathcal{S}_{\text{eval}}(pk, sk_0, pv, \{sv_i\}_{i \in C}, x, y, y_0) \ ,$$

and simulates P_i sending (eid, x, y_i) to all parties. If another honest P_j later receives the input (eid, x) then use the y_j value computed above.

In the following round $\mathcal{S}_{\text{thresh}}$ then simulates the output $((eid, i), P_i, (pv, x, i, y_i))$ on the SOT of $\mathcal{F}_{\text{ZK-PM}}^{R}$. This is what $\mathcal{Z}$ expects to see as the witness sv_i is not leaked by $\mathcal{F}_{\text{ZK-PM}}$. Fortunately, because we do not know any witness. If $\mathcal{Z}$ ever inputs a value of the form $(eid, \texttt{term}, P_i, (pv, x, i, y_i), b')$ to the simulated $\mathcal{F}_{\text{ZK-PM}}^{R}$, then simulate output $(eid, P_i, 1)$ from $\mathcal{F}_{\text{ZK-PM}}^{R}$. By the evaluation simulation property we are guaranteed that $(pv, x, i, y_i) \in L(R)$, so this will not violate $\mathcal{IO}_{\text{ZK-PM}}$ — which would prematurely end the simulation with output ‘‘`H` $\rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$’’.

Figure 6.4: The interface used in the proof of Theorem 6.1 on the preceding page.

all honest parties will output $(eid, f_{sk}(x))$ within r rounds. Both checks are straight-forward. The fact that if the honest parties input (eid, x) and some honest party outputs (eid, y), then $y = f_{sk}(x)$ trivially follows from the fact that we are running in the $\mathcal{F}_{\text{ZK-PM}}^{R}$-hybrid model and was argued in Fig. 6.3 on page 195. □

Notice that in the simulation we used essentially that the simulator gets $y = f_{sk}(x)$ when the *first* honest party inputs (eid, x), to be able to run $\mathcal{S}_{\text{eval}}$ to produce the value to send on behalf of the party. At a first thought it might seem that a functionality where the value $y = f_{sk}(x)$ is not leaked on the SOT until $t - f$ honest parties have received input (eid, x) should be secure, where f is the number of actual corrupted parties in the execution and t is the maximal number of corrupted parties tolerated — we could just consider the $t - f$ honest parties having contributed a share corrupt and then the environment only has

information from t 'corrupted' parties, which should be secure. A proof of this would however require a substantially different proof relying on a computational assumption to allow the honest parties to send wrong evaluation shares in the simulation. The problem is that the environment determines which honest parties should send their share first, so in the above argument the simulator so to say adaptively 'corrupted' $t - f$ parties. Therefore we run into problems from constructing adaptively secure threshold cryptosystem. To deal with the fact that that the environment determines which honest party should send it share first it namely has to be able to simulate a correct share for *all* honest parties. But that clearly implies that the simulator can compute the result using Combine. Therefore, knowing the result when the *first* simulated share is sent is a necessary requirement, and, as we saw above, a sufficient requirement.

As a corollary to Theorem 5.1 on page 180, Theorem 6.1 on page 196 and Theorem 3.5 on page 100 we get that.

Corollary 6.1 *If the simple function sharing scheme* $(gen, f, \text{SingleToThresh}, sf, R, \Sigma, \text{Combine})$ *is statistically secure with corruption threshold* t *and the soundness of* Σ *is not relative to a PPT family of sets, then the protocol* $\pi_{\text{thres}}[\pi^{\Sigma}_{\text{DAM-ZK}}/\mathcal{F}_{\text{ZK-PM}}]$ *realizes the functionality* $\mathcal{F}_{\text{thresh}}$ *from Fig. 6.1 on page 192 under* $\mathcal{IO}_{\text{static}}(t)$ *in the* $(\mathcal{F}_{\text{key-dist}}, \mathcal{F}_{\text{PRS}}^{\text{commit},\mathtt{prs}})$*-hybrid model, where* $\mathtt{prs}$ *is the possible private reference string of* Σ*.*

Addressing the case where Σ only has computational soundness relative the key being randomly generated is considerably more involved. The reason is that in that case we do not have Theorem 5.1 on page 180, so we cannot apply Theorem 3.5 on page 100. The following theorem is however still true.

Theorem 6.2 *If the simple function sharing scheme* $(gen, f, \text{SingleToThresh}, sf, R, \Sigma, \text{Combine})$ *is statistically secure with corruption threshold* t*, then the protocol* $\pi_{\text{thres}}[\pi^{\Sigma}_{\text{DAM-ZK}}/\mathcal{F}_{\text{ZK-PM}}]$ *realizes the functionality* $\mathcal{F}_{\text{thresh}}$ *from Fig. 6.1 on page 192 under* $\mathcal{IO}_{\text{static}}(t)$ *in the* $(\mathcal{F}_{\text{key-dist}}, \mathcal{F}_{\text{PRS}}^{\text{commit},\mathtt{prs}})$*-hybrid model, where* $\mathtt{prs}$ *is the possible private reference string of* Σ*.*

Proof. In the following we only consider environments $\mathcal{Z}$ which corrupts at most t parties and we drop the explicit mentioning of $\mathcal{IO}_{\text{static}}(t)$ in the notation. We have to prove that there exists a simulator $\mathcal{S}$ such that

$$\text{IDEAL}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}_{\text{PRS}}^{\text{commit},\mathtt{prs}}}_{\mathcal{F}_{\text{thresh}},\mathcal{S},\mathcal{Z}} \overset{\text{c}}{\approx} \text{HYB}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}_{\text{PRS}}^{\text{commit},\mathtt{prs}}}_{\pi_{\text{thres}}[\pi^{\Sigma}_{\text{DAM-ZK}}/\mathcal{F}_{\text{ZK-PM}}],\mathcal{Z}}\ , \tag{6.2}$$

for all environments $\mathcal{Z}$ for which $\Pr[\text{IDEAL}^{\mathcal{F}_{\text{key-dist}},\mathcal{F}_{\text{PRS}}^{\text{commit},\mathtt{prs}}}_{\mathcal{F}_{\text{thresh}},\mathcal{S},\mathcal{Z}} = \text{``H} \rightsquigarrow \mathcal{F}_{\text{thresh}}\text{''}] \overset{\text{c}}{\approx} 0$. We claim that this holds for $\mathcal{S} = \mathcal{S}_{\text{thresh}}[\mathcal{S}_{\text{DAM-ZK}}]$, where $\mathcal{S}_{\text{thresh}}$ is given in Fig. 6.4 on the preceding page and $\mathcal{S}_{\text{DAM-ZK}}$ is given in Fig. 5.2 on page 182. The intuitive reason is that in the protocol the key is distributed using $\mathcal{F}_{\text{key-dist}}$ and is therefore exactly randomly generated and that all instances input to $\pi_{\text{DAM-ZK}}$ is for this randomly generated key. Therefore the computational soundness of the Σ-protocol relative to randomly generated keys should be enough. Formally, one observes that in the simulation the key is generated by $\mathcal{F}_{\text{thresh}}$ and

therefore randomly generated. Furthermore, only the public key is handed to $\mathcal{S}$ from $\mathcal{F}_{\text{thresh}}$ — exactly the value it would receive in the game SMS in Section 2.9 if trying to break the computational soundness of the Σ-protocol relative to randomly generated keys. If therefore Eq. 6.2 on the facing page do not hold, we can use Corollary 5.3 on page 186 to construct an expected PPT algorithm which takes as input a random public key and then outputs a break of the extractor with the given random key in the instance, contradicting the computational soundness of Σ relative to randomly generated keys.

The above proof sketch needs to have many details filled in. The techniques used to give a formal proof are almost identical to those used in the proof of Theorem 8.4 on page 275, which contains a detailed application of Corollary 5.3 on page 186 in a more complex setting. We therefore do not provide details here. □

6.3 Examples of Simple Function Sharing

In the above section we gave an abstract treatment of simple function sharing schemes. We defined what they are, gave criteria for calling them statically secure and showed that if they meet these criteria, then they can be used to realize $\mathcal{F}_{\text{thresh}}$. In the section we give some examples of statically secure simple function sharing schemes.

6.3.1 Shamir Secret Sharing

In Section 6.3.2 we will give an example of a function sharing scheme, but first we review a little algebra. The basic idea behind virtually all threshold protocols was introduced by Shamir in [Sha79]. It is based on the basic algebraic fact that if one knows $t+1$ evaluations $y_i = f(i)$, for $i \in C$ say, of a polynomial of degree at most t, then one can determine $f(j)$ in any other point j, as long as the difference between any two distinct points from C is invertible.

As described in [Sha79], this allows to share a secret value s between n servers in such a way that if any t of the servers get together, they have no information about s and if $t+1$ servers get together, then they can always reconstruct s. It is done as follows: Let R be ring such that $s \in R$ and let R^* denote the invertible elements of the multiplicative group of R. Let

$$f(X) = a_0 + \sum_{i=1}^{t} a_i X^i \tag{6.3}$$

be a polynomial over R of degree at most t. Let $I \subseteq R$ be n points for which $i, j \in I$ and $i \neq j$ implies that $(i-j) \in R^*$. Let $C \subset I$ be any subset for which $|C| = t+1$. Since $i, j \in I$ and $i \neq j$ implies that $(i-j) \in R^*$ this allows us to define for each $i \in C$ the following polynomial over R of degree at most t

$$\lambda_{i,C}(X) = \prod_{j \in C \setminus \{i\}} \frac{j - X}{j - i} \ . \tag{6.4}$$

Notice that $\lambda_{i,C}(i) = 1$ and $\lambda_{i,C}(h) = 0$ for $j \in C \setminus \{i\}$. Given any $t+1$ points y_i for $i \in C$ we can then define the following polynomial of degree at most t

$$F(X, \{y_i\}_{i \in C}) = \sum_{i \in C} y_i \lambda_{i,C}(X) \ . \tag{6.5}$$

By construction we have that $F(i) = y_i$ for $i \in C$. If in particular we let $y_i = f(i)$ for $i \in C$, then $F(i) = f(i)$ for $i \in C$. The fact that F and f have degree at most t, that $|C| = t+1$ and $(i-j) \in R^*$ for distinct $i, j \in I$ then imply that

$$F(X) = f(X) \ , \tag{6.6}$$

see e.g. [CF02].

We exploit this fact as follows: To share $s \in R$, generate a random polynomial as in Eq. 6.3 on the preceding page with $f(0) = s$. I.e. let $a_0 = s$ and pick $a_i \in R$ uniformly at random for $i = 1, \ldots, t$. Then for $i \in I$, give $y_i = f(i)$ to server P_i. If $t+1$ servers get together, let C be their indices, then they can pool their shares and compute

$$F(0; \{y_i\}_{i \in C}) = f(0) = s \ .$$

The above technique is called Lagrange interpolation and has exactly the flavor that we are after in Definition 6.2 on page 194, with construction threshold $c = t+1$.

6.3.2 Interpolation in the Exponent

We now show how to distribute the exponentiation function $x \mapsto x^\alpha \bmod p$ for a prime $p = 2q+1$, where q is also a prime. We have that $|\mathbf{Z}_p^*| = 2q$. The subgroup Q_p of order q is the quadratic residues and $g = 4$ is a generator. We describe a simple function sharing scheme for the following function

$$\begin{aligned} exp_{p,\alpha} : & Q_p \to Q_p \\ & x \mapsto x^\alpha \bmod p \ . \end{aligned}$$

The public key is p and the secret key is $\alpha \in \mathbf{Z}_q$. No side information is necessary. Notice that the domain Q_p can be recognized in PPT. We can simply define the function to the identity outside Q_p.

key sharing:

Given a key (p, α) and a construction threshold c we distribute a key as follows:

- $y \leftarrow g^\alpha \bmod p$.
- Let $f(X) \in \mathbf{Z}_q[X]$ be a uniformly random polynomial of degree at most $c-1$, for which $f(0) = \alpha$.
- For $i = 1, \ldots, n$ let $\alpha_i = f(i)$ and let $y_i = g^{\alpha_i} \bmod p$.

Then $pv = (p, y, y_1, \ldots, y_n)$ and $sv_i = \alpha_i$.

evaluation sharing:
: Server P_i computes an evaluation share on $x \in Q_p$ as $z_i = x^{\alpha_i} \bmod p$.

share verification:
: Notice that $z_i = x^{\alpha} \bmod p$ iff the discrete logarithm of z_i base x is identical to the discrete logarithm of y_i base g. Since g and y_i are public value we can therefore simply let R be the relation `equality of discrete logarithms` in Q_p from Section 2.9.2. A Σ-protocol for this relation is given in the same section.

share combining:
: Assume then that we have z_i for $i \in C$, where $(pv, x, z_i, i) \in R$ and $|C| = c$. Since $(pv, x, y_i, i) \in R$ there exists w such that $z_i = x^w \bmod p$ and $y_i = g^w \bmod p$. Since $y_i = g^{f(i)} \bmod p$ it follows that $z_i = x^{f(i)} \bmod p$. Therefore

$$\begin{aligned}\prod_{i \in C} z_i^{\lambda_{i,C}(0)} \bmod p &= \prod_{i \in C} x^{f(i)\lambda_{i,C}(0)} \bmod p \\ &= x^{\sum_{i \in C} f(i)\lambda_{i,C}(0)} \bmod p \\ &= x^{\sum_{i \in C} f(i)\lambda_{i,C}(0) \bmod q} \bmod p \\ &= x^{f(0)} \bmod p = x^{\alpha} \bmod p \ ,\end{aligned}$$

: as desired. Notice that the servers can indeed compute this value as

$$\lambda_{i,C}(0) = \prod_{j \in C \setminus \{i\}} \frac{j}{j-i} \bmod q$$

: is a constant from $\mathbf{Z}_q$, which is trivial to compute when q is known.

The share combining algorithm exploits that Lagrange interpolation is linear and therefore can be done in the exponent of x modulo the order of x. This technique is known as interpolation in the exponent. We have already argued that we have a simple function sharing scheme. Let us call it the threshold exponentiation in Q_p scheme. We prove it secure.

Theorem 6.3 *For all $0 < c \leq n$ the `distributed exponentiation in` Q_p scheme is statistically secure for corruption threshold $t = c - 1$.*

Proof. Assume that we are given a public key $(p, y = g^{\alpha} \bmod p)$ and a set C where $|C| = c-1$. We pick $\alpha_i \in \mathbf{Z}_q$ uniformly at random for $i \in C$ and compute $y_i = g^{\alpha_i} \bmod p$. We then *define* $f(0) = \alpha$ and let $f(i) = \alpha_i$ for $i \in C$. Letting $y_0 = y$ we then have $y_i = g^{f(i)} \bmod p$ for $i \in |C \cup \{0\}| = c$ and can compute $y_i = g^{f(j)} \bmod p$ for $i \in [n] \setminus C$ using interpolation in the exponent as in share combination, using $\lambda_{i,C\cup\{0\}}(j)$ instead of $\lambda_{i,C}(0)$. Then all y_i values are of the right form. It is therefore enough to prove that f is a uniformly random polynomial of degree at most t for which $f(0) = \alpha$. This follows from the fact that for all distinct $i, j \in C \cup \{0\}$ we have that $(i - j) \in \mathbf{Z}_q^* = \mathbf{Z}_q \setminus \{0\}$. Therefore each of the q^t sets of t values $(\alpha_i)_{i \in C} \in \mathbf{Z}_q^t$ give rise to a distinct polynomial f of degree at most t, by the claim right above Eq. 6.6 on the facing page. Since there are exactly q^t polynomials of degree at most t for which $f(0) = \alpha$, we have exactly sampled a uniformly random one.

Assume then that we are given an evaluation $z = x^{\alpha} \bmod p$ and $\alpha_i = f(i)$ for $i \in C$, where $|C| = c - 1$. From α_i we can compute $z_i = x^{\alpha_i} \bmod p$ for $i \in C$. Let $z_0 = z$. Since $\alpha = f(0)$ we therefore have $z_i = x^{f(i)} \bmod p$ for $i \in C \cup \{0\}$, and we can compute $z_j = x^{f(j)} \bmod p$ for $i \in [n]$ using interpolation in the exponent as in share combining. □

6.3.3 Threshold RSA

Our second example of a function sharing scheme is a distributed evaluation of the RSA function. Let $p = 2p' + 1$ and $q = 2q' + 1$ be safe primes and let $N = pq$ and $M = p'q'$. The distribution is done along the lines of the distributed exponentiation in Q_p. Here we also work in the sub-group of quadratic residues Q_N, which has order M. However, a subtly arises because M cannot be made public — given N and M it is trivial to compute p and q. Therefore we do not know the ring $\mathbf{Z}_M$ that we interpolate in. Substantial research has been aimed at solving this issue. Without exhausting the list we mention the results by Rabin [Rab98], Damgård and Koprowski [DK01] and Shoup [Sho00] which all represent different approaches. We are going to adapt the scheme by Shoup and in later chapters use some of the ideas by Rabin.

In [Sho00] Shoup showed how n servers can share between them a secret RSA modulus $d \in \mathbf{Z}_M$ and securely raise any given $x \in \mathbf{Z}_N^*$ to the power $4\Delta^2 d$, where $\Delta = n!$. The presence of $4\Delta^2$ has to do with the fact that we do not know the order of the group we interpolate in. The technique was presented for the setting where $d = e^{-1} \bmod M$ for a public value e. Here we present a modification allowing to use an arbitrary value d and work in $\mathbf{Z}_{N^{s+1}}^*$ for any $s \in \mathbf{N}$. The last modification allows use with Paillier's cryptosystem and was observed in several papers, by Fouque, Poupard and Stern [FPS00], for $s = 2$, and independently by Damgård and Jurik [DJ01]. We use this protocol in all but one of the following chapters, so even though the protocol is almost identical to the protocol in [Sho00], we will present it, for completeness and to verify that the protocol is universally composable.

The protocol by Shoup allows to distribute an RSA exponentiation function for an RSA modulus which is a product of safe primes. The technique requires that some side information is allowed. To be precise we make a definition.

Definition 6.3 *An RSA function for Shoup's threshold technique is a generator gen, which on input k outputs (p, q, e, d, v, w, sk_0), where p and q are $\lceil k/2 \rceil$-bit primes for which also $p' = (p-1)/2$ and $q' = (q-1)/2$ are primes, $d \in \mathbf{N}$ is a secret exponent, $e \in \{0,1\}^*$ is some arbitrary public information, v generates $Q_{N^{s+1}}$, where $N = pq$, $w = v^{\Delta^2 d} \bmod N^{s+1}$ and $sk_0 = v^d \bmod N^{s+1}$. The public key is (N, e, v, w), the secret key is (p, q, d) and sk_0 is some allowable side information. The function considered is $x \mapsto x^{4\Delta^2 d} \bmod N^{s+1}$ with side information $y_0 = x^{2d} \bmod N^{s+1}$.*

Notice that sk_0 and y_0 might actually be extra information. Whereas it is always possible to compute $x^{4\Delta^2 d} \bmod N^{s+1}$ from $x^{2d} \bmod N^{s+1}$, given N, the other direction is not easily computable, as the order of x is not known. This discrepancy in the information learned by the parties and the adversary seems to be essential for the realization to be statistically secure.

We start by describing the solution to the problem that we cannot interpolate modulo $N^s M$, when we do not know it. Let $I \subset [n] \cup \{0\}$ be such that $|I| = c$. The orders of all elements of $Q_{N^{s+1}}$ divide $p^s q^s p' q' = MN^s$. This means that if we do interpolation in the exponent of an element from $Q_{N^{s+1}}$, then we compute modulo $N^s M$. Consider the value in Eq. 6.4 on page 199 in that case. This is then a polynomial modulo $N^s M$, especially the divisions should be computed modulo $N^s M$. Since $N^s M$ is a secret value it seems that the parties cannot do interpolation. Consider instead the value

$$\Lambda_{i,C}(X) = n!\lambda_{i,C}(X) = n! \prod_{j \in C \setminus \{i\}} \frac{j - X}{j - i} \ . \tag{6.7}$$

It is well-known that $\binom{n}{j} = \frac{n!}{(n-j)!j!}$, so it follows from $\binom{n}{j}$ being an integer that $(n-j)!j!$ divides $n!$, from which it easily follows that $\Lambda_{i,C}(X)$ is an integer for $X \in \mathbf{N}$. The divisions are simply canceled out. Using Eq. 6.5 on page 200 and Eq. 6.6 on page 200 it then follows that

$$F^{\Delta}(X, \{y_i\}_{i \in C}) :\equiv \sum_{i \in C} y_i \Lambda_{i,C}(X) \equiv \Delta \sum_{i \in C} y_i \Lambda_{i,C}(X) \equiv \Delta f(X) \pmod{N^s M} \ , \tag{6.8}$$

for $y_i = f(i)$. We can then describe the distribution of the RSA function.

key sharing:
: Given a key (p, q, e, d, v, w, sk_0) and a construction threshold c, let $N = pq$, let $M = p'q'$. We distribute a key as follows:

  - Let $f(X) \in \mathbf{Z}_{MN^s}[X]$ be a uniformly random polynomial of degree at most $c - 1$, for which $f(0) = d$.
  - For $i = 1, \ldots, n$ let $s_i = f(i) \bmod MN^s$ and let $v_i = v^{\Delta s_i} \bmod N^{s+1}$.

  Then $pv = (N, v, e, v_1, \ldots, v_n)$ and $sv_i = s_i$.

evaluation sharing:
: Server P_i computes an evaluation share on $x \in \mathbf{Z}^*_{N^{s+1}}$ as $y_i = x^{2\Delta s_i} \bmod N^{s+1}$.

share verification:
: Let R' denote the relation `equality of RSA discrete logarithms`. The relation R for verifying an evaluation share is given by $((N, y_i, x^{2\Delta} \bmod N^{s+1}, v_i, v), s_i) \in R'$. Notice that $y_i = (x^{2\Delta})^{s_i} \bmod N^{s+1}$ and $v_i = v^{s_i} \bmod N^{s+1}$, so s_i is a correct witness.

share combining:
: Assume then that we have y_i for $i \in C$, where $(pv, x, y_i, i) \in R$ and $|C| = c$. Since $(pv, x, y_i, i) \in R$ there exists w such that $y_i^2 = (x^{2\Delta})^{2w} \bmod N^{s+1}$ and $v_i^2 = v^{2w} \bmod N^{s+1}$.

  Since $v_i = v^{s_i} \bmod N^{s+1}$ it follows that $v_i^2 = v^{2s_i} \bmod N^{s+1}$. Together with $v_i^2 = v^{2w} \bmod N^{s+1}$ and the fact that v^2 generates Q_N when v generates Q_N it then follows

that $w = s_i \bmod N^s M$, which implies that $y_i^2 = (x^{2\Delta})^{2s_i} \bmod N^{s+1}$. Combine as

$$\begin{aligned}\prod_{i\in C}(y_i^2)^{\Lambda_{i,C}(0)} &\equiv \prod_{i\in C}((x^{4\Delta})^{s_i})^{\Lambda_{i,C}(0)} \\ &\equiv \prod_{i\in C}(x^{4\Delta})^{s_i\Lambda_{i,C}(0)} \\ &\equiv (x^{4\Delta})^{\sum_{i\in C} s_i\Lambda_{i,C}(0)} \\ &\equiv (x^{4\Delta})^{\Delta f(0)} \equiv x^{4\Delta^2 f(0)} \equiv x^{4\Delta^2 d} \pmod{N^{s+1}} .\end{aligned}$$

Call this scheme the `distributed RSA` scheme.

Theorem 6.4 *For all $0 < c \leq n$ the `distributed RSA` scheme is statistically secure for corruption threshold $t = c - 1$.*

Proof. Assume that we are given a public key (N, e, v, w, sk_0) with side information $sk_0 = v^d \bmod N^{s+1}$, and a set C where $|C| = c - 1$. For $i \in C$, generate uniformly random $s_i \in \mathbf{Z}_{\lfloor N/4\rfloor N^s}$ and *define* a unique polynomial f by requiring that the degree is at most t and that $f(0) = d \bmod MN^s$ and $f(i) = s_i \bmod MN^s$ for $i \in C$, and *define* $s_j = f(j) \bmod MN^s$ for $i \in [n] \setminus C$. We know s_i for $i \in C$, so we can compute $v^{s_i} \bmod N^{s+1} = v^{f(i)} \bmod N^{s+1}$ for $i \in C$. Furthermore, we received $sk_0 = v^d \bmod N^{s+1} = v^{f(0)} \bmod N^{s+1}$ as side information. So, for $j \in [n]$ we can compute $v_j = v^{\Delta f(j)} \bmod N^{s+1}$ using interpolation in the exponent as in share combining, using v_i instead of y_i^2 and using $\Lambda_{i,C}(j)$ instead of $\Lambda_{i,C}(0)$. Then $pv = (N, v, e, v_1, \ldots, v_n)$ and $sv_i = s_i$ for $i \in C$. We prove that the distribution of the generated values is statistically close to that of a real key. Consider the distribution of the s_i in the real key. As in Theorem 6.3 on page 201 it follows that these values are uniformly random. Since $M = p'q' = (p-1)(q-1)/4 = N/4-(p+q)/2+1/4$ and p and q are $\lceil k/2\rceil$-bit primes it follows that the uniform distribution on $\mathbf{Z}_{MN^s}$ is statistically close to the uniform distribution on $\lfloor N/4\rfloor N^s$, which proves the claim.

Evaluations are handled equivalently. We receive $y = x^{2d} \bmod N^{s+1}$ as side information. We can then compute $x^{2s_i} \bmod N^{s+1}$ for $i \in C$ and then, using that $x^2 \in Q_{N^{s+1}}$, compute $y_i = (x^2)^{\Delta s_j} \bmod N^{s+1}$ using interpolation in the exponent as in share combining. □

6.4 Additive Sharing

For later use we describe how to construct a threshold RSA scheme without side information. We will need it in a context where the side information leaked by the Shoup scheme cannot be allowed — it would break the IND-CPA security of an encryption scheme.

The observation is that if we take the construction threshold to be $c = n$, then we do not need to do interpolation in the reconstruction, which rids of the Δ factors. Then we use the Σ-protocol for equality of even RSA discrete logarithms, which rids of a 2 factor.

Definition 6.4 *An RSA function for additive sharing without side information is a generator gen, which on input k outputs (p, q, e, d, v, w, sk_0), where p and q are $\lceil k/2\rceil$-bit primes for*

which also $p' = (p-1)/2$ *and* $q' = (q-1)/2$ *are primes,* $d \in 2\mathbf{N}$ *is an even secret exponent,* $e \in \{0,1\}^*$ *is some arbitrary public information,* v *generates* $Q_{N^{s+1}}$*, where* $N = pq$*,* $w = v^d \bmod N^{s+1}$ *and* sk_0 *is the empty string. The public key is* (N, e, v, w)*, the secret key is* (p, q, d)*. The function considered is* $x \mapsto x^d \bmod N^{s+1}$ *with no side information.*

key sharing:

Given a key (p, q, e, d, v, w) and a construction threshold c, let $N = pq$, let $M = p'q'$ and let $d = 2d'$. We distribute a key as follows:

- Let $s_1, \ldots, s_n \in \mathbf{Z}_{N^sM}$ be uniformly random elements for which $d' \equiv \sum_{i=1}^{n} s_i \pmod{N^sM}$.
- For $i = 1, \ldots, n$ let $v_i = v^{2s_i} \bmod N^{s+1}$.

Then $pv = (N, v, e, v_1, \ldots, v_n)$ and $sv_i = s_i$.

evaluation sharing:

Server P_i computes an evaluation share on $x \in \mathbf{Z}_N$ as $y_i = x^{2s_i} \bmod N^{s+1}$.

share verification:

Let R' denote the relation `equality of even RSA discrete logarithms`. The relation R for verifying an evaluation share is given by $((N, y_i, x, v_i, v), 2s_i) \in R'$.

share combining:

Assume then that we have y_i for $i \in C$, where $(pv, x, y_i, i) \in R$ and $|C| = c = n$. Since $(pv, x, y_i, i) \in R$ there exists w such that $y_i = x^{2w} \bmod N^{s+1}$ and $v_i = v^{2w} \bmod N^{s+1}$.

Since $v_i = v^{2s_i} \bmod N^{s+1}$ and v generates Q_N it follows that $w \equiv s_i \pmod{N^sM}$, which implies that $y_i \equiv x^{2s_i} \pmod{N^{s+1}}$. Therefore

$$\begin{aligned}\prod_{i\in[n]} y_i &\equiv x^{2\sum_{i\in[n]} s_i} \\ &\equiv (x^2)^{\sum_{i\in[n]} s_i \bmod N^sM} \\ &\equiv (x^2)^{d'} \\ &\equiv x^{2d'} \pmod{N^{s+1}}\ .\end{aligned}$$

It is straight-forward to check the below theorem along the lines of the proof of Theorem 6.4 on the preceding page. The main, and essential, difference being that given the result $y = x^{2d} \bmod N^{s+1}$ and $n-1$ evaluation shares $y_i = x^{2s_i} \bmod N^{s+1}$, the last evaluation share, y_j say, is simulated as $y_j = y \prod_{i\in[n]\setminus\{j\}} y_i^{-1} \bmod N^{s+1}$.

Theorem 6.5 *For all* $0 < c \leq n$ *the* `distributed even RSA` *scheme is statistically secure for corruption threshold* $t = n - 1$.

6.5 A Threshold Signature Scheme

In this section we present the RSA signature scheme from [Sho00]. Let *gen* be a generator which generates (p, q, e, d), where p and q are as in Definition 6.3 on page 202 and where $\gcd(e, n!) = 1$ and $\gcd(e, \phi(pq)) = 1$ and $d = e^{-1} \bmod \phi(pq)$. As suggested in [Sho00] we can e.g. pick e as a prime larger than n. The public key is $(N = pq, e)$ and the secret key is d. The function in consideration is $x \mapsto x^d \bmod N$. To distribute d we generate a uniformly random generator $v \in Q_N$ and use SingleToThresh from `distributed RSA` above on (p, q, e, d, v). Evaluation sharing and share verification is done as in `distributed RSA` and `share combination` is done as in `distributed RSA` until the value $y' = x^{4\Delta^2 d} \bmod N$ is computed. Then we use that $\gcd(e, 4\Delta^2) = 1$ to compute integers α and β such that $\alpha e + \beta 4\Delta^2 = 1$. Then we set $y = y'^{\beta} x^{\alpha} \bmod N$. We then have that $y^e \equiv (x^{4\Delta^2 de})^{\beta} x^{\alpha e} \equiv x^{\alpha e + \beta 4\Delta^2} \equiv x \pmod N$, which implies that indeed $y = x^d \bmod N$ as desired. Let is call this scheme the `distributed RSA signature scheme`.

We claim that this simple function sharing scheme is statistically secure. This follows almost directly from Theorem 6.4 on page 204. Consider namely instead the generator (p, q, e, d, v, w, sk_0), where (p, q, e, d) are as for *gen*, v is a random generator of Q_N and $w = v^{\Delta^2 d} \bmod N$ and $sk_0 = v^d \bmod N$. Since this is exactly the distribution we chose for v in SingleToThresh for *gen* we know that the described generator would be secure if extended that way. The result then follows from the observation that the three values v, w and sk_0 can be generated given just the public key, which then allows to reuse $\mathcal{S}_{\text{keydist}}$ from Theorem 6.4 on page 204. Assume namely that we are given a public key (N, e) for *gen*. We compute a public key (N, e, v, w, sk_0) for the extended generator as follows: Simply pick $sk_0 \in Q_N$ uniformly at random and let $v = sk_0^e \bmod N$ and $w = sk_0^{\Delta^2} \bmod N$. Since sk_0 generates Q_N except with exponentially small probability, let us assume that it does. In that case also $v = sk_0^e \bmod N$ is a uniformly random generator of Q_N as $e \in \mathbf{Z}^*_{\phi(N)}$. Furthermore $sk_0 \equiv sk_0^{ed} \equiv v^d \pmod N$ and $w = sk_0^{\Delta^2} \bmod N = v^{d\Delta^2} \bmod N$ as required. The generated key (N, e, v, w, sk_0) therefore has exactly the right distribution and can then be used as input to $\mathcal{S}_{\text{keydist}}$. We have argued that.

Theorem 6.6 *The simple function sharing scheme* `distributed RSA signature scheme` *is statistically secure for corruption threshold* $t = c - 1$.

Using Theorem 6.1 on page 196 we then get a realization of $\mathcal{F}_{\text{thresh}}^{c,\text{RSA}}$ for `distributed RSA signature scheme`, which we can in turn use to realize the functionality $\mathcal{F}_{\text{T-sig}}$. In Fig. 6.5 on the facing page a realization of $\mathcal{F}_{\text{T-sig}}$ is given in the $\mathcal{F}_{\text{thresh}}^{c,\text{RSA}}$-hybrid model for `distributed RSA signature scheme`. Because of a discrepancy between our specifications of $\mathcal{F}_{\text{T-sig}}$ and $\mathcal{F}_{\text{thresh}}$, we have to require a particular input-output behavior. The ideal functionality $\mathcal{F}_{\text{T-sig}}$ allows sending signature shares to individual parties, whereas an evaluation share in $\mathcal{F}_{\text{thresh}}$ is sent to all parties. We could fix this in the realization by sending some dummy message to parties indicating who the signature share is for. This would however be a very artificial realization, so we will instead simply require the IO restriction that when $(\texttt{sign}, j, m)$ is input to P_i for some $j \in [n]$, then $(\texttt{sign}, j, m)$ is input to P_i for all $j \in [n]$. We write the input as $(\texttt{sign}, m)$.

Protocol $\pi_{\text{tsig}}^{c,\text{RSA}}$

The protocol runs with parties $P_1, \ldots, P_n$ in the $\mathcal{F}_{\text{thresh}}^{c,\text{RSA}}$-hybrid model for `distributed RSA signature scheme`. Party P_i proceeds as follows:

initialize:

On input `init`, input `init` to $\mathcal{F}_{\text{thresh}}^{c,\text{RSA}}$ and wait for the output (N, e). Then output `ready`.

sign:

On input $(\texttt{sign}, m)$, if m was not input before, input $(m, H(m))$ to $\mathcal{F}_{\text{thresh}}^{c,\text{RSA}}$.

If $\mathcal{F}_{\text{thresh}}^{c,\text{RSA}}$ ever outputs (m, y), then store (m, y) and output $(\texttt{signed}, m)$.

transfer:

- On input $(\texttt{transfer}, j, m)$, if (m, y) is stored, then send (m, y) to P_j.
- The party P_j checks whether $H(M) = y^e \bmod N$. If so it stores (m, y) and outputs $(\texttt{transfered}, i, m)$.

Figure 6.5: Realizing $\mathcal{F}_{\text{T-sig}}$ using the simple function sharing scheme `distributed RSA signature scheme`.

Theorem 6.7 *If there exists an RSA generator $(N, e, d) \leftarrow gen$ and a hash-function $H : \{0,1\}^* \to \mathbf{Z}_N^*$ such that the signature scheme given by signature algorithm $\sigma = sig_{H,N,d}(m) = H(m)^d \bmod N$ and verification algorithm $ver_{H,N,e}(\sigma, m) = 1$ iff $\sigma^e \bmod N = H(m)$ is existentially unforgeable under chosen-message attack, then $\pi_{\text{tsig}}^{c,\text{RSA}}$ $(c-1)$-realizes $\mathcal{F}_{\text{T-sig}}^{c}$ under the IO restriction that signature shares are not sent to specific parties.*

Proof. We describe an interface $\mathcal{S}$ such that

$$\text{IDEAL}^{\mathcal{F}_{\text{thresh}}^{c,\text{RSA}}}_{\mathcal{F}_{\text{T-sig}}^{c},\mathcal{S},\mathcal{Z}} \overset{\text{c}}{\approx} \text{HYB}^{\mathcal{F}_{\text{thresh}}^{c,\text{RSA}}}_{\pi,\mathcal{Z}}$$

for all environments $\mathcal{Z}$ for which $\Pr[\text{IDEAL}^{\mathcal{F}_{\text{thresh}}^{c,\text{RSA}}}_{\mathcal{F}_{\text{T-sig}}^{c},\mathcal{S},\mathcal{Z}} = \text{``H} \rightsquigarrow \mathcal{F}_{\text{T-sig}}^{c}\text{''}] \overset{\text{c}}{\approx} 0$. The proof will be very similar to the proof of Theorem 3.7 on page 120, so we will be rather brief in our description.

The interface $\mathcal{S}$ simply generates a key $(N, e, d) \leftarrow gen(k)$ for the signature scheme and simulates the output (N, e) on the SOT of $\mathcal{F}_{\text{thresh}}^{c,\text{RSA}}$. Let us actually assume that $\mathcal{S}$ receives the key (N, e) from the forging game in Definition 2.11 on page 20. This is fine as it only has to simulate output (N, e). Then it simply simulates by following the protocol, except that each time where $\mathcal{F}_{\text{thresh}}^{c,\text{RSA}}$ would have computed $\sigma = sig_{H,N,d}(m)$, instead $\mathcal{S}$ outputs $(\texttt{sign}, m)$ to the forging game and receives $\sigma = sig_{H,N,d}(m)$ from the game. Following the line of reasoning in the proof of Theorem 3.7 on page 120 it can be seen that the simulation is perfect until the point where the environment sends a message (m, y) to an honest P_i where $H(m) = y^e \bmod N$ and where no honest party input $(\texttt{sign}, m)$ at any point. In that case P_i outputs $(\texttt{transfered}, j, m)$ in the simulation, but $\mathcal{S}$ cannot make $\mathcal{F}_{\text{T-sig}}^{c}$ output $(\texttt{transfered}, j, m)$. However, since no honest party input $(\texttt{sign}, m)$ it follows that $\mathcal{S}$ did not ask the forging game for a signature on m. Therefore it can output $(\texttt{forgery}, m, y)$ and win

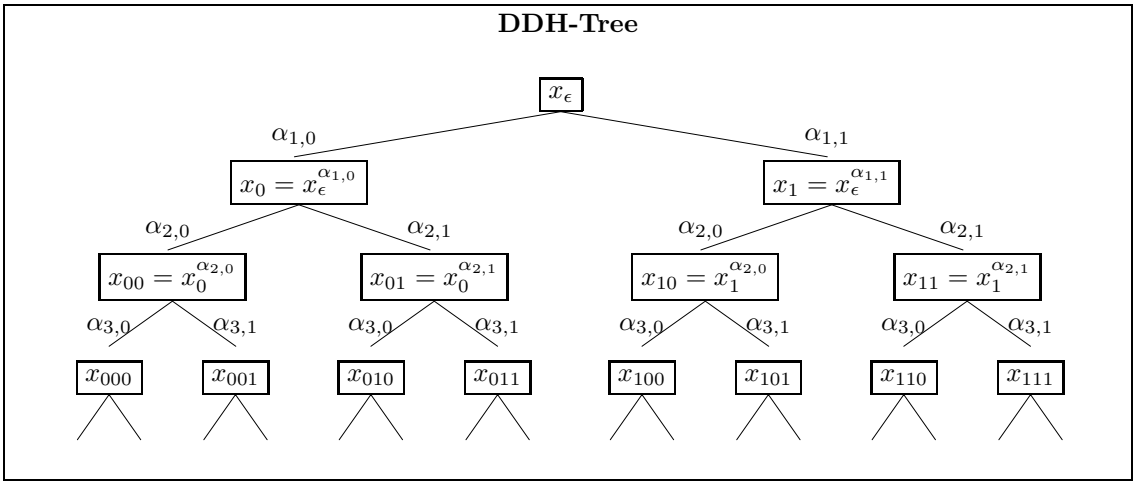

Figure 6.6: The DDH-Tree.

the forgery game. It follows that the simulation is perfect, except with negligible probability. □

6.6 A Distributed Pseudorandom Function

In this section we show how to distribute a pseudorandom function. We start by realizing what we call the growing pseudorandom tree (GRAT) functionality. One should think of the GPRAT functionality as a DDH-Tree as that in Fig. 6.6 sitting inside the functionality. Initially the parties know x_ϵ, and at any point the parties know some sub-tree of the tree rooted in x_ϵ, and can learn the value of a node in the tree if the node is adjacent to the known sub-tree and c parties agree that the node should become known.

Definition 6.5 *The DDH-Tree function family is indexed $i = (Q, \{\alpha_{j,b}\}_{j\in\{1,\ldots,l\},b\in\{0,1\}}, x_\epsilon)$, where Q is a random k-bit prime s.t. $P = 2Q + 1$ is also a prime, l is some polynomial in k, the elements $\alpha_{j,b}$ are random in $\mathbf{Z}_Q^*$, and x_ϵ is random in Q_P. For an index i we define a function*

$$f_i : \{0,1\}^{\leq l} \to Q_P \ , \qquad f_i(\sigma) = x_\epsilon^{\prod_{i=1}^{m} \alpha_{i,\sigma_i}} \bmod P \ ,$$

where $\sigma = (\sigma_1, \ldots, \sigma_m) \in \{0,1\}^{\leq l}$. We sometimes use the notation x_σ to mean $f_i(\sigma)$, when i is clear from the context. Note that in particular, we have $f_i(\epsilon) = x_\epsilon$.

We would like to extend the function f_i such that it outputs pseudorandom bit-strings, instead of elements in Q_P. To this end, given an element $y \in Q_P$, let $\lfloor y \rfloor = \min(y, P - y)$. Consider then an index i as above except that Q is a $(k+\delta)$-bit prime, where $\log(k)/\delta \in o(1)$ (e.g. $\delta = \log^2(k)$). We define the function

$$g_i : \{0,1\}^{\leq l} \to \{0,1\}^k \ , \qquad g_i(\sigma) = \lfloor f_i(\sigma) \rfloor \bmod 2^k \ .$$

The DDH-Tree function family is given by the functions g_i.

A salient property of the pseudorandom function is that is it pseudorandom for the domain $\{0,1\}^{\leq l}$. This is exactly what allows us to use it efficiently, by basically traversing the tree using a distributed evaluation of each step. Before turning our attention to this, let us prove that the construction is indeed a pseudorandom function.

Theorem 6.8 *Under the DDH assumption (Assumption 2.1 on page 21), the DDH-Tree function family is pseudorandom (according to Definition 2.4 on page 13).*

Proof. Let $i = (Q, \{\alpha_{j,b}\}_{j\in\{1,\ldots,l\},b\in\{0,1\}}, x_\epsilon)$ be a random index. Since -1 is not a square in $\mathbf{Z}_P^*$, the map $\lfloor\cdot\rfloor$ is bijective. Since Q is a $(k+\delta)$-bit prime, for a uniformly random value $x \in \mathbf{Z}_Q$, the value $x \bmod 2^k$ is statistically close to uniformly random in $\{0,1\}^k$. It is therefore enough to prove that the output of f_i for random i cannot be distinguished from uniformly random values from Q_P. For this purpose define for $j \in \{1,\ldots,l\}$ and $b \in \{0,1\}$ a function

$$f_{j,b} : Q_P \to Q_P, \qquad f_{j,b}(x) = x^{\alpha_{j,b}} \bmod P$$

and a function

$$g_{j,b} : Q_P \to Q_P$$

which is uniformly random from Q_P to Q_P. Then for $m \in \{0,\ldots,l\}$ let $h_{j,b}^m = f_{j,b}$ if $j \geq m$ and let $h_{j,b}^m = g_{j,b}$ otherwise. Finally let

$$h_i^m(\sigma) = h_{l,\sigma_l}^m \circ \cdots \circ h_{1,\sigma_1}^m(x_\epsilon) \ .$$

Then $f_i = h_i^0$ and h_i^l is statistically close to a uniformly random function from $\{0,1\}^{\leq l}$ to Q_P.[1]

It is therefore enough to prove that h_i^0 and h_i^l cannot be distinguished, which can be done by a hybrids argument. Assume namely that h_i^0 and h_i^l can be distinguished. This means that there exists $m \in \{1,\ldots,l\}$ such that the functions h_i^{m-1} and h_i^m can be distinguished by a PPT distinguisher D having black-box access to the functions. We show that this contradicts the DDH assumption. For this purpose, assume that we have access to a black-box o which returns random values of the form $(x, f_{j,0}(x), f_{j,1}(x))$ if $b = 0$ and returns random values of the form $(x, g_{j,0}(x), g_{j,1}(x))$ if $b = 1$. By a simple application of the DDH assumption it can be seen that no PPT algorithm can guess b with anything but negligible advantage. We reach our contradiction by using D to guess b. To be able to do this we show how to generate values $\{x_\sigma\}_{\sigma\in\{0,1\}^{\leq l}}$ distributed as those defined by h_i^{m-1+b} given oracle access to o: Pick all the values x_σ for $\sigma \in \{0,1\}^{\leq m-1}$ as uniformly random values with the only restriction that they are consistent with random functions, i.e. if $|\sigma_1| = |\sigma_2|$ and $x_{\sigma_1} = x_{\sigma_2}$, then for all suffixes σ make sure $x_{\sigma_1\|\sigma} = x_{\sigma_2\|\sigma}$. When a value x_σ where $|\sigma| = m-1$ is generated, then instead of generating x_σ directly, query o and receive a random evaluation (x, x_1, x_2), where x is uniformly random from Q_P. Then let $x_\sigma = x$, let $x_{\sigma\|0} = x_1$, and let $x_{\sigma\|1} = x_2$. Then generate the remaining values x_σ where $|\sigma| > m$ as done in h_i^{m-1} and h_i^m using random

[1] Only statistically close as collisions will distinguish h_i^l from a uniformly random function: If $|\sigma_1| = |\sigma_2|$ and $h_i^l(\sigma_1) = h_i^l(\sigma_2)$, then for all suffixes σ, $h_i^l(\sigma_1\|\sigma) = h_i^l(\sigma_2\|\sigma)$. However such collisions occur with exponentially small probability.

Functionality $\mathcal{F}^{c}_{\text{GDDH-Tree}}$

The functionality runs with parties $P_1, \ldots, P_n$ and is parametrized by a construction threshold $0 < c \leq n$ and a round complexity r for the evaluate command. All inputs are output on the SOT. It proceeds as follows:

initialize:

On input `init` from all honest parties, generate a random DDH-Tree index $i = (Q, \{\alpha_{j,b}\}_{j\in\{1,\ldots,l\},b\in\{0,1\}}, x_\epsilon)$ as defined in Definition 6.5 on page 208 and output x_ϵ to all parties.

Initialize a dictionary S mapping from $\sigma \in \{0,1\}^{\leq l}$ and indices $i \in \{0,1,\ldots,n\}$ into indices $\{1,\ldots,n\}$. We write $S(i,\sigma)$. Initially $S(i,\epsilon) = [n]$ and $S(i,\sigma \neq \epsilon) = \emptyset$. *We think of $S(m,i)$ as the index of those parties allowing P_i to learn $f_i(\sigma)$. Here $i = 0$ indexes the adversary, which inputs on the SIT.*

evaluate:

If P_i inputs $(\texttt{evaluate}, j, \sigma\|b)$, where $j \in [n]$, $\sigma \in \{0,1\}^{\leq l-1}$, $b \in \{0,1\}$ and $|S(i,\sigma)| \geq c$, then proceed as follows: First output $(\texttt{evaluate}, i, j, \sigma\|b, f_i(\sigma\|b))$ on the SOT. If after r rounds P_i is honest or $(\texttt{interrupt}, i, j, \sigma\|b)$ was not input on the SIT, then let $S(j,\sigma) = S(j,\sigma)\cup\{i\}$ (In the following we use $(\texttt{evaluate}, m)$ as a shorthand for $(\texttt{evaluate}, j, m)$ for all $j \in [n]$). If during this $|S(j,\sigma\|b)| \geq c$, for $j \in [n]$, then output $(\texttt{evaluated}, \sigma\|b, f_i(\sigma\|b))$ to P_j in a round specified be the adversary and no later than r rounds after the last honest party input $(\texttt{evaluate}, j, \sigma\|b)$.

incorrect inputs:

If in the first round were an honest party inputs a non-trivial value some honest party do not input `init`, then break down. Also break down if an honest party inputs `init` twice.

Figure 6.7: The Growing DDH-Tree functionality with construction threshold c.

exponents. It is straightforward to verify that the values thus defined are distributed as in h_i^{m-1} if $b = 0$ and as in h_i^m if $b = 1$.

To use D to distinguish, run it, and when it queries on $\sigma \in \{0,1\}^{\leq l}$ return x_σ. To make the process efficient, the values x_σ are generated when needed. When D makes a guess at b, we output the same guess. □

6.6.1 Distributed Evaluation of DDH-Tree

In this section we show how to distribute the evaluation of the DDH-tree. We present the protocol for the $(\mathcal{F}^{GDDH,c}_{\text{key-dist}}, \mathcal{F}_{\text{ZK-PM}})$-hybrid model, where $\mathcal{F}^{GDDH,c}_{\text{key-dist}}$ is the ideal functionality in Fig. 6.8 on the facing page for distributing a key for the DDH-Tree function family. We then appeal to the composition theorem and 'plug in' the zero-knowledge proof of membership from Chapter 5 to get a protocol for the $(\mathcal{F}^{GDDH,c}_{\text{key-dist}}, \mathcal{F}^{\text{commit}}_{\text{PRS}})$-hybrid model, which only uses the ideal functionalities for setting up keys.

Theorem 6.9 *For $0 < c \leq n$ and $t = c - 1$ the protocol $\pi^{c}_{\text{GDDH-Tree}}$ realizes $\mathcal{F}^{c}_{\text{GDDH-Tree}}$ under $\mathcal{IO}_{\text{static}}(t)$.*

Functionality $\mathcal{F}_{\text{key-dist}}^{GDDH,c}$

The functionality runs with parties $P_1, \ldots, P_n$ and is by a construction threshold $0 < c \leq n$.

generate:

If all honest parties inputs `init` in the same rounds, output as described below in a round determined by the adversary:

- $p = 2q + 1$, where p is a random k-bit safe prime.
- $g = 4$, a generator of Q_p, where Q_q is the sub-group of quadratic residues.
- For $j = 1, \ldots, l$ and $b = 0, 1$:
  - $\alpha_{j,b} \in \mathbf{Z}_q^*$, a uniformly random element.
  - $y_{j,b} = g^{\alpha_{j,b}} \bmod p$.
  - $f_{j,b}(X) \in \mathbf{Z}_q[X]$, a uniformly random polynomial with degree at most $c-1$ for which $f_{j,b}(0) = \alpha_{j,b}$.
  - For $i = 1, \ldots, n$:
    - $\alpha_{j,b,i} = f_{j,b}(i)$.
    - $y_{j,b,i} = g^{\alpha_{j,b,i}} \bmod p$.
- $x_\epsilon \in Q_p$, a uniformly random element.

The values $(q, g, x_\epsilon, \{y_{j,b}\}_{j=1,b=0}^{l,1}, \{y_{j,b,i}\}_{j=1,b=0,i=1}^{l,1,n})$ are output to all parties and the adversary, and the values $\{\alpha_{j,b,i}\}_{j=1,b=0}^{l,1}$ are output to P_i only. The values to the adversary is output on the SOT when the first honest party inputs `init` and the values to the parties are output in a round specified by the adversary.

incorrect inputs:

If in the first round were an honest party inputs a non-trivial value some honest party do not input `init`, then break down. Also break down if an honest party inputs `init` twice or any other value than `init`.

Figure 6.8: The functionality $\mathcal{F}_{\text{key-dist}}$ for distributing a threshold key for DDH-Tree, with construction threshold c.

Proof. The protocol for evaluating one node is the protocol from Fig. 6.3 on page 195 for the scheme `distributed exponentiation in` Q_p for which the claim follows from Theorem 6.3 on page 201 and Theorem 6.1 on page 196. Except for the fact that all $2l$ instances of the protocol use the same prime we could therefore have realized the protocol in the hybrid model with $2l$ ideal functionalities $\mathcal{F}_{\text{thresh}}$ for the scheme the `distributed exponentiation in` Q_p and then plugged in the realization from Fig. 6.3 on page 195. It is straight-forward to verify that using the same prime in all instances does not serve a problem. □

6.6.2 Growing a Random Tree

We now make the obvious observation that if the DDH-Tree function family is pseudorandom then $\mathcal{F}_{\text{GDDH-Tree}}^c$ might as well return random values. No PPT environment can see the difference. We formalize this by specifying a functionality with exactly this behavior. It is

Protocol $\pi_{\text{GDDH-Tree}}$

The protocol runs in the $(\mathcal{F}_{\text{key-dist}}^{GDDH,c}, \mathcal{F}_{\text{ZK-PM}}^{R})$-hybrid model, where R is the relation `equality of discrete logarithms`. The protocol proceeds as follows:

initialize:

On input `init` the party P_i inputs `init` to $\mathcal{F}_{\text{ZK-PM}}^{R}$ and $\mathcal{F}_{\text{key-dist}}^{GDDH,c}$ and waits until $\mathcal{F}_{\text{ZK-PM}}^{R}$ have output `ready` and $\mathcal{F}_{\text{key-dist}}^{GDDH,c}$ have output the public values $(q, g, x_\epsilon, \{y_{j,b}\}_{j=1,b=0}^{l,1}, \{y_{j,b,i}\}_{j=1,b=0,i=1}^{l,1,n})$ along with the secret exponent shares $\{\alpha_{j,b,i}\}_{j=1,b=0}^{l,1}$. Say that the value x_ϵ received as part of the public values is defined.

evaluation:

On input $(\texttt{evaluate}, j, \sigma\|b)$ to P_i, where $\sigma \in \{0,1\}^{\leq l-1}$ and x_σ is defined, party P_i computes the evaluation share

$$x_{\sigma\|b,i} = x_\sigma^{\alpha_{|\sigma|+1,b,i}} \bmod p$$

and sends the value to P_j and proves to P_j that $\log_{x_\sigma}(x_{\sigma\|b,i}) = \log_g(y_{|\sigma|+1,b,i})$ by inputting (pid, x, w) to $\mathcal{F}_{\text{ZK-PM}}^{R}$, where we use the proof id $pid = (\sigma\|b, i, j)$, instance $x = (p, x_\sigma, x_{\sigma\|b,i}, g, y_{|\sigma|+1,b,i})$ and witness $w = \alpha_{|\sigma|+1,b,i}$. The party P_j inputs pid and x.

If a party has received evaluation shares and acceptable proofs from all $i \in C$, where $|C| = c$, the party computes

$$x_{\sigma\|b} \leftarrow \prod_{i \in C} x_{\sigma\|b,i}^{\lambda_{i,C}(0)} \bmod p \ ,$$

outputs $\lfloor x_\sigma \rfloor \bmod 2^k$ and says that $x_{\sigma\|b}$ is defined.

Figure 6.9: The protocol $\pi_{\text{GDDH-Tree}}$.

given in Fig. 6.10 on the next page and grows a random tree.

Theorem 6.10 *For $0 < c \leq n$ and $t = c - 1$ the protocol $\pi_{\text{GDDH-Tree}}^{c}$ realizes $\mathcal{F}_{\text{GDDH-Tree}}^{c}$ under $\mathcal{IO}_{\text{static}}(t)$ if the DDH assumption in Q_p holds (Assumption 2.1 on page 21).*

Proof. By Theorem 6.9 on page 210 we have that there exists an interface $\mathcal{S}$ such that

$$\text{IDEAL}_{\mathcal{F}_{\text{GDDH-Tree}}, \mathcal{S}, \mathcal{Z}_{|\mathcal{IO}(\mathcal{F}_{\text{GDDH-Tree}})}}^{(\mathcal{F}_{\text{key-dist}}^{GDDH,c}, \mathcal{F}_{\text{ZK-PM}}^{R})} \overset{\text{c}}{\approx} \text{HYB}_{\pi_{\text{GDDH-Tree}}^{c}, \mathcal{Z}_{|\mathcal{IO}(\mathcal{F}_{\text{GDDH-Tree}})}}^{(\mathcal{F}_{\text{key-dist}}^{GDDH,c}, \mathcal{F}_{\text{ZK-PM}}^{R})}$$

for all environments $\mathcal{Z}$. Where we can consider the self-restricting $\mathcal{Z}_{|\mathcal{IO}(\mathcal{F}_{\text{GDDH-Tree}})}$ instead of making the condition that $\Pr[\text{IDEAL}_{\mathcal{F}_{\text{GDDH-Tree}}, \mathcal{S}, \mathcal{Z}}^{(\mathcal{F}_{\text{key-dist}}^{GDDH,c}, \mathcal{F}_{\text{ZK-PM}}^{R})} = \text{``H} \rightsquigarrow \mathcal{F}_{\text{GDDH-Tree}}\text{''}] \overset{\text{c}}{\approx} 0$, as $\mathcal{IO}(\mathcal{F}_{\text{GDDH-Tree}})$ can be checked in PPT. We claim that

$$\text{IDEAL}_{\mathcal{F}_{\text{GRAN-Tree}}, \mathcal{S}, \mathcal{Z}_{|\mathcal{IO}(\mathcal{F}_{\text{GRAN-Tree}})}}^{(\mathcal{F}_{\text{key-dist}}^{GDDH,c}, \mathcal{F}_{\text{ZK-PM}}^{R})} \overset{\text{c}}{\approx} \text{HYB}_{\pi_{\text{GDDH-Tree}}^{c}, \mathcal{Z}_{|\mathcal{IO}(\mathcal{F}_{\text{GRAN-Tree}})}}^{(\mathcal{F}_{\text{key-dist}}^{GDDH,c}, \mathcal{F}_{\text{ZK-PM}}^{R})}$$

Functionality $\mathcal{F}^{c}_{\text{GRAN-Tree}}$

The functionality runs with parties $P_1, \ldots, P_n$ and is parametrized by a construction threshold c, a round complexity r for the evaluate command and a depth l. It proceeds as follows:

initialize:

On input `init` from all honest parties, generate a random k-bit string x_ϵ and output it to all parties. Say that x_ϵ is defined.

Initialize a dictionary S mapping from $\sigma \in \{0,1\}^{\leq l}$ and indices $i \in \{0,1,\ldots,n\}$ into indices $\{1,\ldots,n\}$. We write $S(i,\sigma)$. Initially $S(i,\epsilon) = [n]$ and $S(i,\sigma \neq \epsilon) = \emptyset$. *We think of $S(m,i)$ as the index of those parties allowing P_i to learn the random node x_σ. Here $i = 0$ indexes the adversary, which inputs on the SIT.*

evaluate:

If P_i inputs $(\texttt{evaluate}, j, \sigma\|b)$, where $j \in [n]$, $\sigma \in \{0,1\}^{\leq l-1}$, $b \in \{0,1\}$ and $|S(i,\sigma,)| \geq c$, then proceed as follows: If after r rounds P_i is honest or $(\texttt{interrupt}, i, j, \sigma\|b)$ was not input on the SIT, then let $S(j,\sigma) = S(j,\sigma) \cup \{i\}$ and output $(\texttt{evaluate}, i, j, \sigma\|b)$ on the SOT. (In the following we use $(\texttt{evaluate}, m)$ as a shorthand for $(\texttt{evaluate}, j, m)$ for all $j \in [n]$). If during this $|S(j,\sigma\|b)| > t$, for $j \in [n]$, then output $(\texttt{evaluated}, \sigma\|b, x_{\sigma\|b})$ to P_j in a round specified be the adversary and no later than r rounds after the last honest party input $(\texttt{evaluate}, j, \sigma\|b)$, where if $x_{\sigma\|b}$ is not already defined, we define it to be a uniformly random k-bit string.

incorrect inputs:

If in the first round were an honest party inputs a non-trivial value some honest party do not input `init`, then break down. Also break down if an honest party inputs `init` twice.

Figure 6.10: The Growing RANdom Tree functionality with construction threshold c.

for all environments $\mathcal{Z}$, which proves the theorem. Since $\mathcal{IO}(\mathcal{F}_{\text{GRAN-Tree}}) = \mathcal{IO}(\mathcal{F}_{\text{GDDH-Tree}})$ it is enough to prove that

$$\text{IDEAL}^{(\mathcal{F}^{GDDH,c}_{\text{key-dist}}, \mathcal{F}^{R}_{\text{ZK-PM}})}_{\mathcal{F}_{\text{GDDH-Tree}}, \mathcal{S}, \mathcal{Z}_{|\mathcal{IO}(\mathcal{F}_{\text{GRAN-Tree}})}} \overset{\text{c}}{\approx} \text{IDEAL}^{(\mathcal{F}^{GDDH,c}_{\text{key-dist}}, \mathcal{F}^{R}_{\text{ZK-PM}})}_{\mathcal{F}_{\text{GRAN-Tree}}, \mathcal{S}, \mathcal{Z}_{|\mathcal{IO}(\mathcal{F}_{\text{GRAN-Tree}})}} . \tag{6.9}$$

By the premise of the theorem we can assume the DDH assumption in Q_p. By Theorem 6.8 on page 209 this applies that the function in Definition 6.5 on page 208 is a PRF according the Definition 2.4 on page 13. Consider the following PRF adversary A, designed to run in the game in Fig. 2.1 on page 14, with F being the function from Definition 6.5 on page 208. It receives (k,z). Then it starts running $\text{IDEAL}^{(\mathcal{F}^{GDDH,c}_{\text{key-dist}}, \mathcal{F}^{R}_{\text{ZK-PM}})}_{\mathcal{F}_{\text{GRAN-Tree}}, \mathcal{S}, \mathcal{Z}_{|\mathcal{IO}(\mathcal{F}_{\text{GRAN-Tree}})}}(k,z)$ except that we replace the instruction *"if $x_{\sigma\|b}$ is not already defined, we define it to be a uniformly random k-bit string"* with the following instruction *"if $x_{\sigma\|b}$ is not already defined, we output $(\texttt{evaluate}, \sigma\|b)$ in the game in Fig. 2.1 on page 14 and define $x_{\sigma\|b}$ to be the k-bit string we receive from the game"*. Notice that if $b = 0$ in Fig. 2.1 on page 14, then

$$\text{IND}^{\texttt{prf},0}_{F,A} = \text{IDEAL}^{(\mathcal{F}^{GDDH,c}_{\text{key-dist}}, \mathcal{F}^{R}_{\text{ZK-PM}})}_{\mathcal{F}_{\text{GRAN-Tree}}, \mathcal{S}, \mathcal{Z}_{|\mathcal{IO}(\mathcal{F}_{\text{GRAN-Tree}})}}$$

as $x_{\sigma\|b}$ is still defined to be a random k-bit string in that case. If one the other hand $b = 1$, then

$$\mathrm{IND}_{F,A}^{\mathtt{prf},1} = \mathrm{IDEAL}_{\mathcal{F}_{\text{GDDH-Tree}},\mathcal{S},\mathcal{Z}_{|\mathcal{IO}(\mathcal{F}_{\text{GRAN-Tree}})}}^{(\mathcal{F}_{\text{key-dist}}^{GDDH,c},\mathcal{F}_{\text{ZK-PM}}^{R})} .$$

Since A is PPT it follows from Definition 2.4 on page 13 that

$$\mathrm{IND}_{F,A}^{\mathtt{prf},0} \overset{\mathrm{c}}{\approx} \mathrm{IND}_{F,A}^{\mathtt{prf},1} ,$$

which proves Eq. 6.9 on the preceding page, as desired. □

6.6.3 An Efficient and Juicy Unnamed Coin-Flip Protocol

We now show how the parties can use $\mathcal{F}_{\text{GRAN-Tree}}$ to efficiently flip coins. The functionality returns a uniformly random k-bit string to all parties when they have all input `flip`. The value of this string is not chosen until the first honest party has input `flip`. To obtain an efficient realization we consider unnamed coin flip. In a named coin-flip, the parties input $(\mathtt{flip}, name)$, where $name \in \{0,1\}^*$. In response to this the functionality generates a uniformly random string c_{name} and outputs $(name, c_{name})$. We require that the parties do not use the same name twice. In an unnamed coin-flip, after all parties have input `flip` for the r't time they receive the r'th random value. Notice that a named coin-flip is basically just a random oracle, whereas an unnamed coin-flip is weaker. Basically, an unnamed coin-flip functionality provides the parties with a fresh random value in each round, a value which the adversary did not know in the previous round. This turns out to be an extremely powerful resource to have in a network, as we discuss in much more detail in Chapter 7. Here we show how to realize it efficiently. The reader of course already knows how. We simply traverse the tree and output the random values in the nodes. We do this in a manner such that each new value comes from a node we never visited before. If e.g. we do a left-depth-first traversal, then we only use one evaluation per new value. Therefore we have an unnamed coin-flip protocol with communication complexity $O(n^2k)$ and constant round complexity (using the protocol $\pi_{\text{DAM-ZK}}$ from Fig. 5.1 on page 180 for realizing $\mathcal{F}_{\text{ZK-PM}}$).

In [CR03] Canetti and Rabin introduced the notion of universal composition with joint state (JUC), pronounced *juicy*. The idea is the following. Certain functionalities either require an expensive initial setup phase or have to completely rely on an ideal functionality for example for distributing keys.[2] It would therefore be convenient if all protocols could share the realization, so that each protocol would not have to run a separate setup phase. An example of such a protocol is the realization of $\mathcal{F}_{\text{GRAN-Tree}}$ in Fig. 6.9 on page 212. Another prominent example is the realization of $\mathcal{F}_{\text{sig}}$ in Fig. 3.29 on page 121, which requires an initial broadcast. The case of signatures is treated in detail in [CR03]. We show that $\mathcal{F}_{\text{GRAN-Tree}}$ allows for an efficient realization of the so-called multi-session extension of the unnamed coin-flip functionality. The concept of multi-session extension of a given functionality is introduced in [CR03]. Basically it runs internally many copies of the given functionality and all messages are then tagged with the id of the copy they are coming from respectively

[2] In practice an ideal functionality for distributing keys would probably have to be realized by common trust in some third party which could generate and distribute the keys.

Protocol π_{access}

The protocol runs with parties $P_1, \ldots, P_n$ in the $\mathcal{F}_{\text{GRAN-Tree}}$-hybrid model. Party P_i proceeds as follows:

initialize:

On input `init`, input `init` to $\mathcal{F}_{\text{GRAN-Tree}}$ and wait for output x_ϵ from $\mathcal{F}_{\text{GRAN-Tree}}$. Say that x_ϵ is computed and is being computed.

evaluate:

On input $(\texttt{access}, \sigma)$, let $\sigma[j]$ denote the j-bit prefix of σ. Let l be the largest integer $l \leq |\sigma|$ for which $\sigma[l]$ is being computed. Then say that $\sigma[m]$ for $m = l+1, \ldots, |\sigma|$ are being computed. For $m \in \{l+1, \ldots, |\sigma|\}$, when $\sigma[m-1]$ is computed, input $(\texttt{evaluate}, \sigma[m])$ to $\mathcal{F}_{\text{GRAN-Tree}}$ and when $\mathcal{F}_{\text{GRAN-Tree}}$ outputs $(\sigma[m], x_{\sigma[m]})$, say that $x_{\sigma[m]}$ was computed. When x_σ is computed, output $(\texttt{access}, x_\sigma)$.

Figure 6.11: The protocol for accessing a given node.

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

The functionality runs with parties $P_1, \ldots, P_n$. It is parametrized by a session id length l_1 and a round complexity r_1 for initialization and a round complexity r_2 for coin-flips. All inputs are output on the SOT. The functionality proceeds as follows:

initialize:

If all honest parties input $(\texttt{init}, sid)$ in the same round, then set $R_{sid,i} = 0$ for $i \in [n]$. Then for r_1 rounds, ignore all messages of the form $(sid, \ldots)$.

flip: When a party P_i inputs $(\texttt{flip}, sid)$, then let $R_{sid,i} = R_{sid,i} + 1$.

For all $R > 0$ and $sid \in \{0,1\}^l$, the first time $R_{sid,i} = R$ for any honest party, generate a uniformly random string $c_{sid,R} \in \{0,1\}^k$ and output $(sid, R, c_{sid,R})$ on the SOT.

If in some round (sid, R', i) is input on the SIT and $R' \leq R_{sid,i}$ and $(sid, R', c_{sid,R'})$ was not output to P_i, then output $(sid, R', c_{sid,R'})$ to P_i. Do this no later than r_2 rounds after $R_i \geq R'$ for all honest parties P_i.

incorrect inputs:

If in the first round were an honest party inputs $(\texttt{init}, sid)$ some honest party do not input $(\texttt{init}, sid)$, then break down on session sid. I.e. ignore all inputs of the form $(sid, \cdots)$ and allow the adversary to deliver outputs out the form $(sid, \cdots)$. Also break down on session sid if $|sid| > l_1$ or an honest party inputs $(\texttt{init}, sid)$ twice.

Figure 6.12: The multi-session functionality for unnamed coin-flip.

heading to. Instead of first specifying the unnamed coin-flip protocol and then taking the multi-session extension, we have specified the multi-session extension directly in Fig. 6.12. Notice that we have divided the IO restriction into an IO restriction for each session. This means that if one protocol using the functionality violates the IO restriction, it will not bring down the whole system.

We sketch how one can realize $\mathcal{F}_{\text{coin}}$ efficiently in the $\mathcal{F}_{\text{GRAN-Tree}}$-hybrid model. We assume that the depth of the tree we use is $l = 2l_1 + l_2$ and we realize $\mathcal{F}_{\text{coin}}$ with session id

length l_1 and depth l_2 of the tree in each session.

We are going to separate the nodes of the main-tree into two structures. The main-tree and the sub-trees. For a string $sid \in \{0,1\}^{\leq l_1}$, let main-tree$(sid) = sid_1 0 sid_2 0 \cdots sid_{m-1} 0 sid_m$, where $m = |sid|$ and sid_i is the i'th bit of sid. In particular, main-tree$(\epsilon) = \epsilon$. Furthermore, let sub-tree$(sid) =$ main-tree$(sid)1$.

We call the nodes indexed by main-tree(sid) for some $sid \in \{0,1\}^{\leq l_1}$ the main-tree nodes and we call the sub-tree rooted at the node indexed by sub-tree(sid) the sid-sub-tree. We call the nodes indexed by main-tree$(sid)0$ the junk nodes. Note that each node of the DDH-tree is exclusively either a main-tree node, a junk node or a node of some sid-sub-tree for $sid \in \{0,1\}^{\leq l_1}$. If we eliminate the junk nodes, what is left is essentially a binary tree, where in addition each node sid also holds a binary tree, being the sid-sub-tree. Clearly, each of the sid-sub-trees have depth at least l_2.

We can now access nodes in tree using $(sid, \sigma) \in \{0,1\}^{\leq l_1} \times \{0,1\}^{\leq l_2}$ as follows: On input (sid, σ), let γ be the longest prefix of sub-tree$(sid)\sigma$ for which the node indexed by γ is defined. For each node on the path from γ to sub-tree$(sid)\sigma$, evaluate that node. Then output the value of the node indexed by sub-tree$(sid)\sigma$. We call this protocol π_{access}, see Fig. 6.11 on the page before. Since the sid-sub-trees are non-overlapping, the value of any node sub-tree$(sid)\sigma$ in the sid-sub-tree is unrevealed until some party has input (sid, σ'), where σ is a prefix of σ'. This actually realizes a multi-session extension of $\mathcal{F}_{\text{GRAN-Tree}}$. The parties can then realize $\mathcal{F}_{\text{coin}}$ by simply doing a left-depth-first traversal of the sid-sub-tree in the sid session.

Clearly, reaching the root of the sid-sub-tree might require as many as $2|sid|+1$ `evaluate`-calls to $\mathcal{F}_{\text{GRAN-Tree}}$. So, the initialization of the multi-session extension of unnamed coin-flip is not in general constant round. However, if also the session ids makes up an interval , or are dense in an interval, then initialization will be efficient too. If we plug in the realization of $\mathcal{F}_{\text{ZK-PM}}$ from Fig. 5.1 on page 180, then the round-complexity of each proof is 3. The trapdoor commitment scheme needed for $\mathcal{F}_{\text{ZK-PM}}^{\text{commit}}$-hybrid model in which $\pi_{\text{DAM-ZK}}$ runs can be constructed based on the DH assumption which is implied by the DDH assumption that we already need, see Section 2.6. Since the evaluation share can be sent with the first message in the proof, we have argued the following:

Theorem 6.11 *If the DDH assumption (Assumption 2.1 on page 21) holds, then for all $0 < c \leq n$ and $l_1, l_2 \in \mathbf{N}$, there exists a protocol which under $\mathcal{IO}_{\text{static}}(c-1)$ realizes $\mathcal{F}_{\text{coin}}$ in the $(\mathcal{F}_{\text{key-dist}}^{GDDH,c}, \mathcal{F}_{\text{PRS}}^{\text{commit}})$-hybrid model with session id length l_1, sub-tree depth l_2 and round-complexity 3 for evaluations and round-complexity $6l_1 + 3$ for initialization.*

CHAPTER 7

The Broadcast Model

Stand upright, speak thy thoughts, declare
The truth thou hast, that all may share;
Be bold, proclaim it everywhere:
They only live who dare.
— Voltaire

7.1 Introduction

In this chapter we consider the Byzantine agreement problem and a generalization of the problem called the multi Byzantine agreement problem. We will give an efficient solution to the multi Byzantine agreement problem and show how to use it for realizing the broadcast model efficiently. In this introduction we first describe the Byzantine agreement problem an review some previous work on the problem. Then we introduce and motivate the multi Byzantine agreement problem. We then describe our contributions and compare them to previous work.

7.1.1 Byzantine Agreement

The Byzantine agreement (BA) problem [LSP82] is one of the fundamental problems in reliable distributed systems. The problem generalizes a number of problems in keeping a distributed system consistent when some parts might malfunction or even act according to some malevolent plan. The Byzantine agreement problem is an extensively studied and well understood problem.

A Byzantine agreement protocol consists of n parties. Parties take one input of the form v_i, where $v_i \in \{0,1\}^*$, and produces at most one output of the form w_i, where $w_i \in \{0,1\}^*$. The correctness notions for Byzantine agreement is divided into agreement and validity. Agreement specifies that if two honest parties output w_i respectively w_j, then $w_i = w_j$. Validity specifies that if all honest parties have the same input $v_i = v$, then no honest party outputs w_i for $w_i \neq v$. For now we assume that the honest parties are simultaneously activated, i.e. they all receive their input v_i in the same round. We say that the BA is decided in round r if all honest parties have output some w_i by round r.

7.1.1.1 Known Bounds

It was proved in [LSP82] that for an asynchronous network no solution to the BA problem exists when $n \leq 3t$. If however one assumes that the parties proceed in synchronized rounds of communication and that they can sign messages, then the situation improves somewhat; In that model solutions to the BA problem exists as long as $n > 2t$. A BA protocol where the parties have access to signing messages is usually called an authenticated BA protocol. The synchronization assumption is essential: If the parties are not synchronized, still $n < 3t$ is required even for an authenticated BA protocol. As for the worst-case round complexity it was proved by Fischer, Lynch and Paterson [FLP85] that no asynchronous protocol solving the BA problem in the presence of even one corrupt party can have a bounded round-complexity. Since, as discussed below, we are here primarily interested in the worst-case complexities, we will therefore only be interested in the synchronous model. For the synchronous model, Fischer and Lynch [FL82] proved that any BA protocol must have worst-case round complexity at least $t + 1$, and Garay and Moses [GM93] presented the first polynomial unauthenticated protocol with optimal resilience $(n - 1)/3$ which met this bound.

The result in [FLP85] was shown to hold also for authenticated BA by Dolev and Strong [DS82] and independently by DeMillo, Lynch and Merritt [DLM82]. The $t+1$ bound is rather pessimistic. It says that the price for a high resilience is a high worst-case round complexity. There is however some granularity. The bound does not say that a protocol which can tolerate t faults must use $t + 1$ rounds in all runs. Dolev, Reischuk and Strong [DRS90] proved that any BA protocol with resilience t must have some runs where $f < t$ parties were corrupted and the last honest party terminates in round $f + 2$. This is somewhat less pessimistic as the bound says that even with a high resilience one might get by paying only a price proportional to the actual corruption level. It is proved in [DRS90] that if one imposes the extra condition on the BA protocol that all honest parties terminates in the same round, known as simultaneous BA, then the protocol will have runs with $t+1$ rounds where no parties were corrupted. It follows that any protocol beating the $t + 1$ bound must accept that in some runs honest parties will terminate in different rounds — we say that the protocol has a staggered termination.

As touched upon already, another way to beat the pessimistic $t + 1$ bound is in the expectation. The $t + 1$ bound was first beaten in the expectation by Rabin [Rab83] and independently by Ben-Or [Ben83]. In [Rab83] authenticated BA protocols for both the synchronous and asynchronous models are presented, both with a constant expected round complexity. For the synchronous model the resilience is $(n - 1)/4$. For the asynchronous model the resilience is $(n-1)/10$. In [Ben83] an asynchronous BA protocol is presented with resilience $(n - 1)/5$. Whereas the protocols from [Rab83] have a constant expected round complexity for the mentioned resiliences, the protocol from [Ben83] is only constant expected round when t is $O(\sqrt{n})$.

The protocols from [Rab83, Ben83] have a similar structure, which we call the polling-lottery-decision (PLD) structure following [Rab83]. The source of the efficiency of the protocols from [Rab83] is the introduction of the notion of a common coin. A common coin can be thought of as a network resource which at the desire of the parties gives all parties access to a random bit with the properties that all honest parties agree on the value and that no

corrupted party could predict the value prior to the common coin-toss. We will use essential ideas from the PLD paradigm in our protocols, so we review it here.

7.1.1.2 The Polling-Lottery-Decision Paradigm

In general a PLD BA protocol proceeds as follows: Initially every party holds a bit. The protocol then proceeds in PLD phases. In the polling step each party sends to every other party a value thought of as indicating what that party currently thinks the outcome of the protocol should be, initially its input value. In the lottery step the parties toss a common coin. In the decision step each party then updates its bit as follows: A party which saw a large enough majority for either 0 or 1 in the polling updates its bit to that majority value. This step is constructed such that if any honest party sees a large enough majority for 0, then no honest party sees a large enough majority for 1. The parties that do not see a large enough majority takes the value of the common coin as its new bit. This means that in each round there is at least one good coin value which makes all parties agree: All parties that do not adopt the common coin value saw a majority vote for the same value. If therefore in some phase the coin hits this value, all parties will have the same vote after that phase. We say that the state of the protocol is perfectly biased. The protocol then contains a mechanism that allows the parties to terminate from such a perfectly biased state.

Bracha [Bra84] was first to obtain optimal resilience for constant (requiring that t is $O(\sqrt{n})$) expected round asynchronous BA. Independently Toueg [Tou84] presented optimally resilient synchronous and asynchronous authenticated BA protocols. These protocols are PLD protocols. An important contribution of [Tou84] is the introduction of what we will call vote justifiers following [CKS00]. In [Tou84] the parties sign the vote sent in the polling step, and a party has to send along signatures from enough other parties from the previous step to justify its vote. This technique basically forces the corrupted parties to follow the decision step or abstain from sending a bit. Still, in [Tou84], as in [Rab83], the common coin is assumed to be a resource provided by the environment.

Feldman and Micali [FM97] were the first to present an instantiation of the PLD paradigm with optimal resilience $(n-1)/3$ for the unauthenticated synchronous model. Another main contribution of [FM97] was to implement the common coin within the network. This is done using so-called verifiable secret sharing (VSS) techniques [CGMA85]. Later Canetti and Rabin [CR93] presented an instantiation of the PLD paradigm for the asynchronous model with optimal resilience $(n-1)/3$, and later again Cachin, Kursawe and Shoup [CKS00] presented a constant expected round and optimally resilient instantiation of the PLD paradigm for the authenticated asynchronous model. Their protocol is the first protocol to obtain an expected communication complexity of $O(n^2k)$ bits, where k is the length of a signature. They use the idea of vote justifiers from [Tou84]. The gain in efficiency primarily comes from a novel use cryptography to implement the common coin and compact the vote justifiers. The coin is implemented using a distributed pseudorandom function by Naor and Reingold [NR97] based on the decisional Diffie-Hellman assumption. The vote justifiers in [Tou84] have bit-length $\Theta(nk)$, as they consist of $\Theta(n)$ signatures on votes from the previous PLD phase. In [CKS00] the vote justifiers are compacted using a threshold signature scheme from [Sho00] based on the RSA assumption and the random-oracle assumption. Remember that in a threshold

signature scheme there is some threshold t such that it takes precisely $t+1$ signatures on a message m from distinct parties to construct a threshold signature on m. Using the scheme from [Sho00] the length of the threshold signature is the same as of an individual signature. Therefore the vote justifications sent by the parties will only have the length of one individual signature. This gives a communication complexity of $O(n^2k)$ bits for each phase.

7.1.2 Multi Byzantine Agreement

In this chapter we consider a less studied variation of the Byzantine agreement problem,[1] which we call the multi Byzantine agreement (MBA) problem, and give near optimal protocols for the synchronous models. Studying the multi Byzantine agreement problem can be seen as the problem of realizing the broadcast model, while allowing to obtain a relatively higher efficiency per BA, by exploiting that many BAs might be carried out in the same session.

A multi Byzantine agreement protocol consists of n parties. Parties take inputs of the form $(baid, b_i)$, where $baid$ if a BA id and $b_i \in \{0,1\}$, and produces outputs of the form $(\texttt{decide}, baid, c_i)$ where $c_i \in \{0,1\}$.[2] A given $baid$ will occur as input to a given party at most once and we require that it occurs in an output at most once. The correctness notions for Byzantine agreement generalize straight-forwardly to give the following correctness notions for multi Byzantine agreement: Agreement: If two honest parties output $(\texttt{decide}, baid, v)$ respectively $(\texttt{decide}, baid, w)$, then $v = w$. Validity: If any honest party outputs $(\texttt{decide}, baid, v)$ in round r, then in some round $r' \leq r$ an honest party received $(baid, v)$ as input.[3] For now we assume that if any honest party receives $(baid, \cdot)$ in round r, then all honest parties receive $(baid, \cdot)$ in round r — we say that $baid$ is simultaneously activated in round r. We say that a BA $baid$ is decided in round r if all honest parties have output $(\texttt{decide}, baid, \cdot)$ by round r.

We define a notion of round-complexity. We say that a round is activating if some BA was activated in that round, and we say that a round is active if there exists some BA which was activated in that round or an earlier round and which was not decided. A round is also considered active if any party sent or received a message. The round-complexity of a protocol is the number of active rounds. We will express the round-complexity of a run as a function of t, f and A, where t is the maximal number of corruptions tolerated by the protocol, f is the actual number of corruptions during the run and A is the number of activating rounds. As for the BA problem, the resilience is the maximal number of corrupted parties which can be tolerated and the communication complexity is the total number of bits sent by honest parties.

Since a MBA protocol can be used for running one BA it follows that the worst-case round-complexity is at least $\min(f+2, t+1)$. This of course does not mean that a run with A activations must have round-complexity $A\min(f+2, t+1)$. Indeed, the only bound easily obtainable using [DRS90] is the bound $\min(f+2A, t+2A-1)$, see Section 7.9. We

[1] The only work we know of considering the problem is [BDGK95]. We relate our work to [BDGK95] below.

[2] For simplicity, we consider the binary BA problem — the multivalued BA problem reduces efficiently to the binary case. For the unauthenticated model see Turpin and Coan [TC84], and for the authenticated model, see Section 7.3.4.

[3] Notice that since we consider binary inputs this condition is equivalent to saying that if all honest parties receive the same input, then no honest party decides on a value different from the common input.

will obtain an upper bound which is only a small constant factor off this lower bound. The $c\min(f+2A, t+2A-1)$ upper bound makes a deterministic MBA protocols practical in many settings. If $A > f$, then even though some individual BAs might take $c(f+2)$ rounds, the amortized round complexity per BA will be no more that $3c$. This means that if the MBA protocol is used by a protocol which is not reactive, but outputs to its environment only after having run all of the sequential BAs, then the use of a deterministic MBA protocol for all purposes has the same effect as having used a BA protocol with a constant round complexity of $3c$. There are two advantages of this approach as compared to using a BA protocol with only constant *expected* round complexity. First, the round complexity of the protocol using the BA protocol as sub-protocol will have a worst-case bound unless this protocol is itself randomized, and probably more important, the deterministic MBA protocols are very efficient w.r.t. communication complexity, as compared to known randomized BA protocols.

For the authenticated model another strong motivation exists for considering the MBA problem: In [LLR02a] Lindell, Lysyanskaya and Rabin showed that to obtain resilience $t \geq (n-3)/2$, in the authenticated model, one must use an independent signature scheme for each BA, which reduces to using a unique session id for each BA. Therefore, some non-trivial coordination between BAs is needed under all circumstances. One might therefore consider the MBA problem as the problem of choice for the authenticated model. Coordination between BAs is needed anyway, so one might as well gain what is possible from this requirement, namely efficiency.

7.1.2.1 Our Contribution and Related Work

We present several contributions within the area of MBA, most of which have not yet been published.

Our Contribution First we present a solution to the MBA problem for the unauthenticated synchronous model and the authenticated synchronous model. Both protocol can be run in either a deterministic mode or a randomized mode. For the unauthenticated deterministic protocol, the resilience is $(n-1)/3$, the round-complexity is $O(A+f)$, where A is the number of activating rounds and f is the number of actual faults, and the communication complexity is $O(nt(B+f))$, where B is the number of BAs. For the unauthenticated randomized protocol, the resilience is $(n-1)/3$, the round-complexity is $O(A+f)$, the *expected* round complexity is $O(A)$, the communication complexity is $O(nt(B+f)+t^4(A+f))$, where the $t^4(A+f)$ term comes from using the coin from [FM97] in each round, and the *expected* communication complexity is $O(ntB+t^4A)$. For the authenticated deterministic protocol, the resilience is $(n-1)/2$, the round-complexity is $O(A+f)$ and the communication complexity is $O(knt(B+f))$, where k is the length of a signature. For the authenticated randomized protocol, the resilience is $(n-1)/2$, the round-complexity is $O(A+f)$, the *expected* round complexity is $O(A)$, the communication complexity is $O(knt(B+f))$, when the coin from [CKS00] is used, and the *expected* communication complexity is $O(kntB)$.

Parallel Broadcast Previously Ben-Or and El-Yaniv [BE03] studied the problem of running n BAs in parallel with constant expected round complexity. This is a problem which

occurs naturally in a protocol where BA is used to simulate broadcast. They showed how to use a PLD protocol to do n BAs in a constant expected number of rounds, while paying only a factor $\log(n)$ in overhead when compared to plainly running n copies of the PLD protocol in parallel.[4] In particular they consider the Feldman-Micali protocol which gives a communication complexity of $n\log(n)(nt+t^4) = O(n^5\log(n))$. As a corollary to our result (with $A=1$ and $B=n$) we get that the problem can be solved with communication complexity $O(n^2t+t^4) = O(n^4)$. The improvement comes from the fact that we use only one coin, instead of one coin per instance of the protocol, and that we do not carry out a factor $\log(k)$ 'extra' BAs. We note that [BE03] also present a protocol for running many BA in parallel in the asynchronous model, a problem that we do not consider here.

Both Optimal Worst-Case and Optimal Expected Round-Complexity In [GP90] Goldreich and Petrank showed how to compose a PLD protocol and a deterministic BA protocol with optimal worst-case round complexity to obtain a BA with constant expected round complexity and near optimal worst-case round complexity. First the PLD protocol is run for at most $\log(t)$ phases and if not terminated the protocol is finished by running the deterministic BA protocol for $\min(f+2, t+1)$ rounds. This gives a worst-case round complexity of $c\log(t)+f+2$. This is near optimal, but if $f=\omega(\log(t))$ it is more than a constant factor off. If our MBA protocol is used in place of the PLD protocol in [GP90] we obtain a bound of $c\min(f+2,\log(t))+f+2$, which is never more that a constant factor from optimal and which equals the bound from [GP90] when $f+2 \geq \log(t)$. This is of theoretical interest, as it proves that the $\log(t)$ term in the worst-case round complexity is not inherent for a BA protocol with constant expected round complexity. The improvement might also be of practical interest as the optimal worst-case round complexity $\min(f+2,t+1)$ is arguably most interesting exactly when f is very small, 1 say, where the protocol in [GP90] already has worst-case round complexity $\log(t)$.

Previous Work on Amortized MBA In [BDGK95] Bar-Noy, Deng, Garay and Kameda study a problem similar to the MBA problem — in [BDGK95] only the unauthenticated model is studied. The protocol in [BDGK95] has resilience $(n-1)/3$, round complexity $O(A+f)$ and communication complexity $O(ntB+nt^3)$, where our protocol has communication complexity $O(ntB+ntf)$, an improvement of at least a factor t when $B=O(t)$. Another improvement over [BDGK95] is that our protocol has an optimal message size. In particular, in any round where B BAs are running the message size will be $O(B)$, whereas the protocol from [BDGK95] can in the worst case send messages of size $O(t)$ even when only one BA is running. Our protocol builds on ideas from [BDGK95] and is in many respects similar to the protocol from [BDGK95], which we be discussed more elaborate when our protocol has been presented.

Besides improving on the efficiency of the protocol from [BDGK95] we identify an issue with their formulation of the problem, which is basically the one we have adapted in the introduction, and which we will change accordingly after describing the problem. The problem with the current formulation of the MBA problem is the following: Whereas the

[4] In which case the expected round complexity is $\log(n)$.

$\mathcal{F}_{\text{MBA}}$ can be efficiently realized — here we let $\mathcal{F}_{\text{MBA}}$ denote a functionality that captures the MBA problem — it cannot be efficiently *used*! The problem with the functionality is rather simple and stand out the first time one tries to use it. The problem is this: If one has to activate several dependent BAs[5] on the functionality, then there will be runs where they are activated with at least $t+1$ rounds between any two BAs, even in an execution where no party is corrupted, and even if each individual BA terminates in, say, expected constant round. We sketch why this is the case. Because we require the BAs to be activated simultaneously the claim follows directly from the result in [DRS90] on simultaneous BA. Since the BAs are dependent, at the least the l'th BA is activated after the $(l-1)$'th BA decided. So, we can let the $(l-1)$'th BA terminate when the l'th BA is activated, which by the requirement that the l'th activation is simultaneous would guarantee simultaneous termination of the $(l-1)$'th BA. In other words, any protocol which is able to use a MBA protocol correctly can be used as a sub-protocol for simultaneous termination. We notice that this is not a problem which is by any means associated to the fact that we consider deterministic protocols; Remember that, *any* BA protocol terminating in less than $t+1$ rounds must have a staggered termination.

We solve the problem that the MBA functionality cannot be *used* efficiently by formulating an alternative functionality which allows for staggered activation, i.e. it allows that the parties supply inputs in different rounds. This immediately invalidates the line of argument that showed that $\mathcal{F}_{\text{MBA}}$ cannot be used efficiently, and indeed the functionality, which we will call $\mathcal{F}_{\text{SMBA}}$ (for 'staggered multi BA'), can be used efficiently. Summarizing, we allow staggered termination to facilitate efficient realization and we allow staggered activation to facilitate efficient use.

7.1.3 Realizing the Broadcast Model

There is one problem left however, namely that the functionality by far is as interesting to have access to as the abstract BA functionality $\mathcal{F}_{\text{BA}}$ from Fig. 3.17 on page 91, because $\mathcal{F}_{\text{SMBA}}$ has staggered termination. Having to control the staggering gap in the protocol using $\mathcal{F}_{\text{SMBA}}$ is tedious. This motivates our second contribution. We show how to execute any protocol for the $\mathcal{F}_{\text{BA}}$-hybrid model, in the $\mathcal{F}_{\text{SMBA}}$-hybrid model, with one call to $\mathcal{F}_{\text{SMBA}}$ per call to $\mathcal{F}_{\text{BA}}$. Since $\mathcal{F}_{\text{BA}}$ cannot be realized efficiently, we cannot appeal to the composition theorem. Instead, we will provide a 'compilation result' which takes any protocol for the $\mathcal{F}_{\text{BA}}$-hybrid model and compiles it into a protocol for $\mathcal{F}_{\text{SMBA}}$-hybrid model with the same input-output behavior, except for one-round staggered activation and termination.

7.1.3.1 Related Work on Staggered MBA

The compilation result improves on a result by Lindell, Lysyanskaya and Rabin [LLR02b] who showed how to securely *sequentially* compose protocols with a known upper bound on the termination staggering gap, while maintaining the round complexity of the protocols, except for a small constant factor. This was then used to argue that any *non-reactive* protocol for the broadcast model could be implemented using staggered BA. Here we establish this

[5] By dependent we mean that the input of the l'th BA depends on the output of the $(l-1)$'th BA.

result for reactive protocols. Besides improving the generality of the result we improve the efficiency and the complexity of the techniques considerable. This is done using some simple observations about staggering gaps: That fixed staggering gaps efficiently reduce to one-round staggering gaps and that sequential and parallel composition of protocols with one-round staggering gap yields protocols with one-round staggering gaps. These observations allow a trivial solution to the problem of composing protocols with staggering gaps. As a result we can do without the so-called synchronizing BAs done in the protocol from [LLR02b], which adds most of the complexity in [LLR02b], in bits communicated and techniques.

7.1.4 Applications

We use the $\mathcal{F}_{\text{BA}}$-hybrid model several times in the remaining chapters, and we will rely on the fact that it can be implemented efficiently. Let us however already mention one application of our results to motivation the reader. In [HM01] Hirt and Maurer presented the currently most efficient secure general multiparty computation protocol for the information theoretic model. In [HM01] it was proved that if, over a finite field, the function to be computed can be specified as an arithmetic circuit with m multiplication gates and multiplicative depth d, then the function can be securely evaluated with communication complexity $O(mnt)$ and round-complexity $O(nt + d)$. The term nt in the round complexity comes from the fact that the computation is done in n layers and that after each layer a BA is done to detect errors. Since we have $A = n$ activating rounds, it follows that if the BAs are carried out using $\mathcal{F}_{\text{SMBA}}$ then the round complexity can be reduced to $O(n + d)$, while maintaining the same communication complexity.

7.2 The BA Compilation Theorems

In the subsequent sections we prove that any secure protocol π for the BA model can be turned into a corresponding protocol π' realizing the BAs over point-to-point channels. The protocol π' has communication complexity and round complexity in the order of the protocol π plus three additional terms: A term of $O(n^2)$ communicated bits per bit on which a BA is done;[6] A term $O(f)$ rounds, where f is the number of parties corrupted during the execution; And a term $O(n^2 f)$ bits communicated. The results are stated in more details in the following section.

7.2.1 The Theorems

Theorem 7.1 *If π $\lceil (n-1)/3 \rceil$-realizes $\mathcal{F}$ in the $(\mathcal{F}_{\text{BA}}, \mathcal{G})$-hybrid model, then there exists a protocol π' $\lceil (n-1)/3 \rceil$-realizing* $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}, 1)$ *in the* $\text{Stagger}^{\mathcal{IO}}(\mathcal{G}, 1)$*-hybrid model. The communication complexity and active-round complexity of any run of π' is only a constant factor higher than the communication complexity respectively the round complexity of π on the same sequence of commands, plus a term of $O((C + f)n^2 k)$ bits in communication complexity*

[6] If the parties inputs $\{(baid, m_i)\}_{i \in H}$ to the BA protocol we say that a BA was done on $\max\{|m_i|\}_{i \in H}$ bits.

and a term of $O(f)$ rounds in round complexity, where C is the total number of bits on which a BA is done and f is the total number of parties which were corrupted during the run.

Theorem 7.2 *If π $\lceil(n-1)/2\rceil$-realizes $\mathcal{F}$ in the $(\mathcal{F}_{\text{BA}}, \mathcal{G})$-hybrid model and there exists an protocol π_{tsig} with initializing functionality $\mathcal{I}$ for the real-life model $\lceil(n-1)/2\rceil$-realizing $\mathcal{F}_{\text{T-sig}}$,[7] then there exists a protocol π' $\lceil(n-1)/2\rceil$-realizing $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}, 1)$ in the $\text{Stagger}^{\mathcal{IO}}(\mathcal{G}, 1)$-hybrid model with initializing functionality $\mathcal{I}$. Assuming that π_{tsig} has communication complexity $O(k)$ per command and round complexity $O(1)$ per command, the communication complexity and active-round complexity of any run of π' is only a constant factor higher than the communication complexity respectively the round complexity of π on the same sequence of commands, plus a term of $O((B+f)n^2k + n^2C)$ in communication complexity and a term $O(f)$ in round complexity, where B is the total number of BAs.*

7.2.2 Proof Overview

The proof is divided into several parts.

7.2.2.1 Compiling $(\mathcal{F}_{\text{BA}}, \mathcal{G})$-Hybrid Protocols to $(\mathcal{F}_{\text{SMBA}}, \text{Stagger}^{\mathcal{IO}}(\mathcal{G}, 1))$-Hybrid Protocols

In this part of the proof we observe that, using Section 3.11, any protocol π realizing $\mathcal{F}$ in the $(\mathcal{F}_{\text{BA}}, \mathcal{G})$-hybrid model can be turned into a protocol $\text{Stagger}(\pi, 1)$ realizing $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}, 1)$ in the $(\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{BA}}, 1), \text{Stagger}^{\mathcal{IO}}(\mathcal{G}, 1))$-hybrid model, with a communication complexity and a round complexity in the order of that of π. To obtain the stated results it is therefore enough to sufficiently efficiently realize $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{BA}}, 1)$. What is sufficiently efficient depends on the corrupting threshold: For the threshold $t = \lceil(n-1)/3\rceil$ it is sufficient to show that $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{BA}}, 1)$ can be realized with communication complexity $O((C+f)n^2)$ bits and round complexity $O(A+f)$, where A is the total number of different rounds where a BA was activated; The reason being that the protocol π will already have round complexity at least A. For the threshold $t = \lceil(n-1)/2\rceil$ it is sufficient to show that $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{BA}}, 1)$ can be realized with communication complexity $O((B+f)n^2k+n^2C)$ bits and round complexity $O(A+f)$. We can then appeal to the composition theorem and get the stated results. This part of the proof is discussed in more detail in Section 7.3, where we also introduce a functionality $\mathcal{F}_{\text{SMBA}}$ which is $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{BA}}, 1)$ except for some insignificant syntactical details. This functionality is introduced to have a full specification of $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{BA}}, 1)$. The remaining parts of the proof then focus on realizing $\mathcal{F}_{\text{SMBA}}$ sufficiently efficient.

7.2.2.2 Realizing $\text{MultiRound}(\mathcal{F}_{\text{BA}})$ using $\mathcal{F}_{\text{MBA}}$ and $\mathcal{F}_{\text{sync}}$

In this part of the proof we let $\mathcal{F}_{\text{MBA}}$ be the ideal functionality $\mathcal{F}_{\text{SMBA}}$ with the restriction that parties must deliver inputs in the same round[8], we let $\mathcal{F}_{\text{sync}}$ be an ideal functionality

[7] Candidates for this realization of $\mathcal{F}_{\text{T-sig}}$ are the protocol $\pi_{\text{tsig-triv}}$ in Fig. 3.31 on page 125 (with $\mathcal{F}_{\text{sig}}$ replaced by π_{sig} from Fig. 3.29 on page 121) or the protocol from Section 6.5.

[8] I.e. $\mathcal{F}_{\text{MBA}}$ has simultaneous activation and one-round staggered termination.

which allows staggered parties to synchronize[9] and we show that by first running $\mathcal{F}_{\text{MBA}}$ and then $\mathcal{F}_{\text{sync}}$ we get a realization of the functionality MultiRound($\mathcal{F}_{\text{BA}}$).[10] This part of the proof is done in Section 7.4.

7.2.2.3 Realizing $\mathcal{F}_{\text{MBA}}$

In this part of the proof we show how to efficiently realize $\mathcal{F}_{\text{MBA}}$. This is done in two steps: In Section 7.7 we show how to efficiently realize a so-called special phase-king BA functionality. We show how to do this for the threshold $\lceil (n-1)/3 \rceil$ in the real-life model and for the threshold $\lceil (n-1)/2 \rceil$ in the $\mathcal{F}_{\text{T-sig}}$-hybrid model. In Section 7.8 we then show how to t-realize $\mathcal{F}_{\text{MBA}}$ using a t-realization of a special phase-king BA protocols.

7.2.2.4 Realizing $\mathcal{F}_{\text{SMBA}}$ using $\mathcal{F}_{\text{SGR}}$

In this part of the proof we basically combine the above two parts and apply the staggered compilation theorem. Having a realization of MultiRound($\mathcal{F}_{\text{BA}}$) using $\mathcal{F}_{\text{MBA}}$ and $\mathcal{F}_{\text{sync}}$ and a realization of $\mathcal{F}_{\text{MBA}}$ we get a realization of MultiRound($\mathcal{F}_{\text{BA}}$) in the $\mathcal{F}_{\text{sync}}$-hybrid model by appealing to the composition theorem. Then appealing to the staggered compilation theorem in Section 3.11 we get a realization of $\text{Stagger}^{\mathcal{IO}}(\text{MultiRound}(\mathcal{F}_{\text{BA}}), 1)$ in the $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$-hybrid model. We then observe that except for insignificant syntactical differences, $\text{Stagger}^{\mathcal{IO}}(\text{MultiRound}(\mathcal{F}_{\text{BA}}), 1)$ and $\mathcal{F}_{\text{SMBA}}$ are identical; Both solve the BA problem, both allow one-round staggered activation and both allow one-round staggered termination. This gives us a realization of $\mathcal{F}_{\text{SMBA}}$ in the $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$-hybrid model. We then observe that $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$ is a functionality which given inputs from staggered parties terminates with some fixed output to all parties with at most one-round staggering. So, instead of synchronizing, $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$ reduces a staggering gap to a staggering gap of 1. We give an explicit formulation of this functionality called the staggering gap reduction functionality, $\mathcal{F}_{\text{SGR}}$, and note that we have a realization of $\mathcal{F}_{\text{SMBA}}$ in the $\mathcal{F}_{\text{SGR}}$-hybrid model. This part of the proof is done in Section 7.6.

7.2.2.5 Realizing $\mathcal{F}_{\text{SGR}}$

In this part of the proof we show how to realize $\mathcal{F}_{\text{SGR}}$. We show how to do this for the threshold $\lceil (n-1)/3 \rceil$ in the real-life model and for the threshold $\lceil (n-1)/2 \rceil$ in the $\mathcal{F}_{\text{T-sig}}$-hybrid model. Combined with the above part of the proof, this gives us a realization of $\mathcal{F}_{\text{SMBA}}$. For the threshold $\lceil (n-1)/3 \rceil$ the realization is for the real-life model and for the threshold $\lceil (n-1)/2 \rceil$ the realization is for the $\mathcal{F}_{\text{T-sig}}$-hybrid model. Finally we can then replace $\mathcal{F}_{\text{T-sig}}$ by any protocol realizing it, e.g. the protocol $\pi_{\text{tsig-triv}}$ based on standard signatures or the threshold signature scheme in Section 6.5. This part of the proof is done in Section 7.5.

[9] I.e. after being activated by all parties, $\mathcal{F}_{\text{sync}}$ terminates with some fixed output value in the same round at all parties.

[10] The functionality MultiRound($\mathcal{F}_{\text{BA}}$) requires parties to input in the same round and delivers outputs in the same round to all parties.

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

The functionality runs with parties $P_1, \ldots, P_n$ and outputs all input values on the SOT. It proceeds as follows:

activation:

Let H denote the set of indices of honest parties, and let `Running` be a set, which is initially empty. If in two consecutive rounds each P_i for $i \in H$ inputs $(baid, b_i)$ for $b_i \in \{0,1\}$, then add $B = (baid, \{(i, b_i)\}_{i \in H})$ to `Running` and output B on the SOT.

decision:

If in any round a value $(baid, S, b)$ is input on the SIT and $B = (baid, I) \in$ `Running`, then remove B from `Running`, let the adversary specify an output round and a one-round staggering along with a bit b, and output $(\texttt{decide}, baid, b')$ to each party, where b' is determined as follows: If for all $(i, b_i) \in I$ it holds that $b_i \neq b$, then let $b' \leftarrow 1 - b$, otherwise let $b' = b$.

incorrect inputs:

Output `fail` and break down as soon as one of the following conditions hold:

- In some round an honest party inputs a value which is not of the form $\perp$ or $(baid, b_i)$ for $b_i \in \{0,1\}$.
- Some honest party input a value of the form $(baid, \cdot)$ and there does not exists two consecutive rounds during which each P_i for $i \in H$ input $(baid, b_i)$ for $b_i \in \{0,1\}$.
- Some honest party used the same BA is *baid* twice.
- Two BAs are activated in different orders at two honest parties.

Figure 7.1: The Staggered Multi BA functionality.

7.3 From $(\mathcal{F}_{\text{BA}}, \mathcal{G})$-Hybrid Protocols to $(\mathcal{F}_{\text{SMBA}}, \text{Stagger}^{\mathcal{IO}}(\mathcal{G}, 1))$-Hybrid Protocols

7.3.1 The Staggered MBA Functionality, $\mathcal{F}_{\text{SMBA}}$

As was discussed in the introduction, any protocol implementing a BA functionality with simultaneous termination in the real-life model must use $t + 1$ rounds per activation. If we want an efficient implementation of the broadcast model we therefore have to take another approach. In this section we introduce the staggered MBA functionality, $\mathcal{F}_{\text{SMBA}}$, which allows both staggered activation and staggered termination. The ideal functionality is given in Fig. 7.1. The functionality was formulated to allow only one-round staggering. The reason for this is that one-round staggering is sufficient to obtain an efficient realization.

Notice that we allow the functionality to use several rounds per activation, letting the adversary decide the running time. The reason for not capturing guaranteed termination in the general ideal functionality is that in the synchronous UC framework the ideal functionality can capture the exact round behavior of a protocol, and the precise formulation of guaranteed termination is therefore best seen as a specific of a given protocol, as opposed to a part of the general statement of the problem. Notice that even though the ideal func-

tionality does not guarantee termination, it guarantees that if any party decides, then all honest parties decide within a gap of at most one round. An IO behavior is defined by the conditions in incorrect inputs. The last condition turns out to be essential for us here. We can realize $\mathcal{F}_{\text{SMBA}}$ almost optimally under $\mathcal{IO}_{\text{SMBA}}$, defined by incorrect inputs in Fig. 7.1 on the preceding page, but have no equally efficient realization under the IO behavior without the requirement that the BAs are activated in the same order at all parties.

As staggered termination is sufficient to facilitate an efficient realization of the MBA problem we discuss why we allow staggered activation.

7.3.2 If You Stagger, I Stagger

As mentioned in the introduction, allowing for a staggering gap allows for more efficient implementations: worst-case $f + 2$ rounds, where f is the number of actual faults and *expected* constant round. If one is not careful, the problem however remains. The reason is that after the first run of the BA protocol the parties will not be synchronized anymore. This means that if we require that the parties have the IO behavior $\mathcal{IO}_{\text{MultiRound}}$, then they cannot activate the next BA when the previous one terminates. This means that if we formulate a functionality which is allowed to have staggered termination, but does not allow for staggered activation, then the functionality cannot be used round efficiently even though each activation might return in expected or amortized constant round.

In a bit more detail, assume that a BA functionality is used to carry out l dependent BAs in sequence, all with simultaneous activation, where we define dependent by the minimal requirement that a party cannot input the next BA before the current one has decided at that party. We argue that then there are contexts where there are $t+1$ rounds between each activation. Consider namely a run of the protocol, where the i'th BA is activated in round r_i, and where the i'th BA terminates in round $t_i \leq r_{i+1}$. Possibly some parties terminate before round t_i, but we can transform the protocol into one where, for $i = 1, \ldots, l-1$, all parties terminate the i'th BA simultaneously in round r_{i+1} — simply by letting the parties run until round $r_{i+1} \geq t_i$ without sending any values. Since the first $l-1$ BAs are simultaneous BAs it follows from the result in [DRS90], using an argument similar to the one given in Section 7.9, that with significant probability each of these BAs will run for at least $t+1$ rounds in some execution where there are no corruptions. Therefore we will not have any use of the fact that each individual BA might terminate quickly — the parties still have to wait $t+1$ rounds to activate the next one.

Intuitively the problem is that if a BA terminates quickly, here meaning in less than $t+1$ rounds, then there will be a staggering gap with significant probability, so the parties cannot just activate the next BA when the current one terminates. This then suggests the solution, namely to allow a staggering gap in the activation. When the protocol itself delivers a gap in the termination, it is only fair that it can expect one in the next activation. Since any fixed staggering gap can be efficiently reduced to staggering gaps of one round as shown in the Section 7.5, we use a one-round staggering gap in the definitions.

7.3.3 Compiling $(\mathcal{F}_{\mathrm{BA}}, \mathcal{G})$-Hybrid Protocols to $(\mathcal{F}_{\mathrm{SMBA}}, \mathrm{Stagger}^{\mathcal{IO}}(\mathcal{G}, 1))$-Hybrid Protocols

In this section we show how to compile $(\mathcal{F}_{\mathrm{BA}}, \mathcal{G})$-hybrid protocols to $(\mathcal{F}_{\mathrm{SMBA}}, \mathrm{Stagger}^{\mathcal{IO}}(\mathcal{G}, 1))$-hybrid protocols with essentially the same complexity. As a corollary to Theorem 3.11 on page 149 we get that:

Corollary 7.1 *If $\pi = (P_1, \ldots, P_n)$ realizes $\mathcal{F}$ in the $(\mathcal{F}_{\mathrm{BA}}, \mathcal{G})$-hybrid model with initializing functionality $\mathcal{I}$, then there exists a protocol $\pi' = (P'_1, \ldots, P'_n)$ realizing $\mathrm{Stagger}^{\mathcal{IO}}(\mathcal{F}, 1)$ in the $(\mathrm{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\mathrm{BA}}, 1), \mathrm{Stagger}^{\mathcal{IO}}(\mathcal{G}, 1))$-hybrid model with initializing functionality $\mathcal{I}$. The communication complexity and the active-round complexity of any run of π' on a given sequence of inputs and using random bits $(r_1, \ldots, r_n)$ for the parties $(P_1, \ldots, P_n)$ is only a constant factor higher than the communication complexity respectively the round complexity of π on the same sequence of inputs and using random bits $(r_1, \ldots, r_n)$ for the parties $(P_1, \ldots, P_n)$.*

Lemma 7.1 *There exists a protocol, with communication complexity $O(n^2C)$ and active-round complexity $O(A)$,[11] $\lceil (n-1)/3 \rceil$-realizing $\mathrm{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\mathrm{BA}}, 1)$ in the $\mathcal{F}_{\mathrm{SMBA}}$-hybrid model and if there exists a realization of $\mathcal{F}_{\text{T-sig}}$ with the properties specified in Theorem 7.2 on page 225, then there exists a protocol, with communication complexity $n^2kB + n^2C$,[12] $\lceil (n-1)/2 \rceil$-realizing $\mathrm{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\mathrm{BA}}, 1)$ in the $\mathcal{F}_{\mathrm{SMBA}}$-hybrid model with initializing functionality $\mathcal{I}$.*

Proof. We first show the claim for a version of $\mathcal{F}_{\mathrm{SMBA}}$ which takes inputs $b_i \in \{0,1\}^*$. For this version of $\mathcal{F}_{\mathrm{SMBA}}$, the protocol simply relays the inputs to the protocol to $\mathcal{F}_{\mathrm{SMBA}}$ and uses the outputs from $\mathcal{F}_{\mathrm{SMBA}}$ as the outputs of the protocol. Since the interfaces of the ideal functionalities $\mathrm{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\mathrm{BA}}, 1)$ and $\mathcal{F}_{\mathrm{SMBA}}$ are the same and the environment has the same power over the two ideal functionalities[13] it follows that the protocol is secure. It is therefore enough to realize (with the above complexities) $\mathcal{F}_{\mathrm{SMBA}}$ with arbitrary inputs using $\mathcal{F}_{\mathrm{SMBA}}$ (and for the threshold $t = \lceil (n-1)/2 \rceil$, the initializing functionality $\mathcal{I}$). For the threshold $t = \lceil (n-1)/3 \rceil$ this is straight-forward using the techniques in Turpin and Coan [TC84], and for the threshold $t = \lceil (n-1)/2 \rceil$ this is done in Theorem 7.4 on page 231 below. □

Putting the above to results together we get that:

Theorem 7.3 *If $\pi = (P_1, \ldots, P_n)$ realizes $\mathcal{F}$ in the $(\mathcal{F}_{\mathrm{BA}}, \mathcal{G})$-hybrid model with initializing functionality $\mathcal{I}$ and if there exists a realization of $\mathcal{F}_{\text{T-sig}}$ with the properties specified in Theorem 7.2 on page 225, then there exists a protocol $\pi' = (P'_1, \ldots, P'_n)$ realizing $\mathrm{Stagger}^{\mathcal{IO}}(\mathcal{F}, 1)$ in the $(\mathcal{F}_{\mathrm{SMBA}}, \mathrm{Stagger}^{\mathcal{IO}}(\mathcal{G}, 1))$-hybrid model with initializing functionality $\mathcal{I}$. The communication complexity and the active-round complexity of any run of π' on a given sequence*

[11] As above A denotes the total number of rounds where a BA is activated and C denotes the total number of bits on which a BA is done.

[12] As above B denotes the total number of BAs.

[13] The only difference being the syntax of the messages input by the environment to control the ideal functionalities.

Protocol π_{SBA}

The protocol runs with party $P_1, \ldots, P_n$ in the $(\mathcal{F}_{\text{T-sig}}^{t+1}, \mathcal{F}_{\text{BA}})$-hybrid model, where $\mathcal{F}_{\text{T-sig}}^{t+1}$ has round complexity r for signing. On input $(baid, m_i)$, for $m_i \in \{0,1\}^*$, the party P_i proceeds as follows:

1. Input $(\texttt{sign}, (baid, m_i, -1))$ to $\mathcal{F}_{\text{T-sig}}^{t+1}$ and wait r rounds.
2. If $\mathcal{F}_{\text{T-sig}}^{t+1}$ outputs $(\texttt{signed}, (baid, m_i, -1))$, then input $(\texttt{transfer}, (baid, m_i, -1))$ to $\mathcal{F}_{\text{T-sig}}^{t+1}$.
3. If $\mathcal{F}_{\text{T-sig}}^{t+1}$ output $(\texttt{signed}, (baid, m_i, -1))$ in Step 2 and $\mathcal{F}_{\text{T-sig}}^{t+1}$ does not output $(\texttt{transfered}, (baid, m, -1))$ for $m \neq m_i$ in this step, then input $(\texttt{sign}, (baid, m_i, 0))$ to $\mathcal{F}_{\text{T-sig}}^{t+1}$ and wait r rounds.
4. If $\mathcal{F}_{\text{T-sig}}^{t+1}$ outputs $(\texttt{signed}, (baid, m_i, 0))$, then input $(\texttt{transfer}, (baid, m_i, 0))$ to $\mathcal{F}_{\text{T-sig}}^{t+1}$.
5. Let M be the set of m for which $\mathcal{F}_{\text{T-sig}}^{t+1}$ outputs $(\texttt{transfered}, (baid, m, 0))$. If $\mathcal{F}_{\text{T-sig}}^{t+1}$ output $(\texttt{signed}, (baid, m_i, 0))$ in the previous step, then set $G_i = 1$. Otherwise set $G_i = 0$. Then input $(baid, G_i)$ to $\mathcal{F}_{\text{BA}}$.
6. Let $(\texttt{decide}, baid, v)$ be the output from $\mathcal{F}_{\text{BA}}$. If $v = 0$, then output $(\texttt{decide}, baid, \perp)$, and if $v = 1$, then pick $m \in M$ and output $(\texttt{decide}, baid, m)$.

Figure 7.2: The multivalued BA protocol.

of inputs and using random bits $(r_1, \ldots, r_n)$ for the parties $(P_1, \ldots, P_n)$ is only a constant factor higher than the communication complexity respectively the round complexity of π on the same sequence of inputs and using random bits $(r_1, \ldots, r_n)$ for the parties $(P_1, \ldots, P_n)$, plus a term $O(n^2kB + n^2C)$ in communication complexity.

7.3.4 Efficient Reduction of Multivalued BA to Binary BA using Threshold Signatures

In this section we show that in the $\mathcal{F}_{\text{T-sig}}$-hybrid model, multivalued BA can be efficiently reduced to binary BA. That this is also the case in the unauthenticated model was proved by Turpin and Coan [TC84] and we will not review the construction here. The reduction of multivalued BA to binary BA is given in Fig. 7.2. Notice that we use $\mathcal{F}_{\text{BA}}$ as the underlying BA functionality.

Excluding the binary BA the round complexity is $2r + 2$ and the communication complexity in bits is $O(n^2 l)$ bits and $O(n^2)$ accesses to $\mathcal{F}_{\text{T-sig}}^{t+1}$, where l is a bound on the length of the m_i for honest P_i. We start the analysis by proving two lemmas about π_{SBA}.

Lemma 7.2 *If at most t parties are corrupted and two honest parties P_i and P_j output $(\texttt{decide}, baid, v_i)$ respectively $(\texttt{decide}, baid, v_j)$, then $v_i = v_j$ and P_i and P_j output in the same round.*

Proof. If $\mathcal{F}_{\text{BA}}$ outputs $(\texttt{decide}, baid, v)$ to some party P_j, then by specification it outputs $(\texttt{decide}, baid, v)$ at all honest parties P_j in the same round, as the parties are clearly syn-

chronized until the call of $\mathcal{F}_{\text{BA}}$ given that $\mathcal{F}_{\text{T-sig}}^{t+1}$ has round complexity r. Assume first that $v = 0$. Then all honest parties output $\perp$. Assume then that $v = 1$. This means that at least one honest party input $(baid, 1)$ to the binary BA, so some honest party had a signature on some $(baid, m, 0)$ in Step 4. This in turn implies that at least one honest party, P_i say, input $(\texttt{sign}, (baid, m, 0))$ in Step 3 as the construction threshold is $t + 1$ and at most t parties were corrupted. This means that P_i input $(\texttt{transfer}, (baid, m, -1))$ in Step 2, which implies that in Step 3 all honest parties received output $(\texttt{transfered}, (baid, m, -1))$. Therefore, by the requirements for inputting $(\texttt{sign}, (baid, m_i, 0))$ in Step 3, no honest party input $(\texttt{sign}, (baid, m', 0))$ for $m' \neq m$ in Step 3. Furthermore, no honest party inputs $(\texttt{sign}, (baid, m', 0))$ in any step but Step 3. Therefore, no party receives $(\texttt{transfered}, (baid, m', 0))$ for any $m' \neq m$, so $M = \{m\}$ for all honest parties, which proves the theorem. □

Lemma 7.3 *If at most $t \leq (n-1)/2$ parties are corrupted and all honest parties start with the same input $(baid, m)$, then all honest parties that output $(\texttt{decide}, baid, m')$ have $m' = m$.*

Proof. In Step 1 at least $n - t \geq t + 1$ parties input $(\texttt{sign}, (baid, m, -1))$ to $\mathcal{F}_{\text{T-sig}}^{t+1}$. So, in Step 2 all honest parties will see $\mathcal{F}_{\text{T-sig}}^{t+1}$ output $(\texttt{signed}, (baid, m, -1))$. Furthermore, no honest party input $(\texttt{sign}, (baid, m', -1))$ for $m' \neq m$. Therefore $\mathcal{F}_{\text{T-sig}}^{t+1}$ will not output $(\texttt{signed}, (baid, m', -1))$ for $m' \neq m$. Therefore, in Step 3 the honest parties, of which there are at least $t + 1$, will input $(\texttt{sign}, (baid, m, 0))$, and no honest party inputs $(\texttt{sign}, (baid, m', 0))$ for $m' \neq m$. Therefore, all honest parties will in Step 4 see $\mathcal{F}_{\text{T-sig}}^{t+1}$ output $(\texttt{signed}, (baid, m, 0))$ and will set $G_i = 1$. Therefore, the result of the BA will be 1 and all honest parties which output will output $(\texttt{decide}, baid, m)$. □

The following lemma follows from the above lemmas.

Lemma 7.4 *For $t \leq (n-1)/2$ the protocol π_{SBA}, with no communication, t-realizes* MultiRound$(\mathcal{F}_{\text{BA}})$ *in the $(\mathcal{F}_{\text{T-sig}}^{t+1}, \mathcal{F}_{\text{BA}})$-hybrid model, while giving only binary inputs to $\mathcal{F}_{\text{BA}}$.*

We can then use Theorem 3.11 on page 149 to get the desired result.

Theorem 7.4 *For $t \leq (n-1)/2$, if there exists a realization of $\mathcal{F}_{\text{T-sig}}^{t+1}$ with the properties in Theorem 7.2 on page 225, then there exists a protocol t-realizes $\mathcal{F}_{\text{SMBA}}$ in the $\mathcal{F}_{\text{SMBA}}$-hybrid model with initializing functionality $\mathcal{I}$, while giving only binary inputs to $\mathcal{F}_{\text{SMBA}}$. The communication complexity is $O(n^2kB + n^2C)$ and the active-round complexity is $O(A)$.*

Proof. As a starting point we take the protocol in Lemma 7.4. Then we replace $\mathcal{F}_{\text{T-sig}}^{t+1}$ with the realization for the real-life model with initializing functionality $\mathcal{I}$ and get a realization of MultiRound$(\mathcal{F}_{\text{BA}})$ in the $\mathcal{F}_{\text{BA}}$-hybrid model with initializing functionality $\mathcal{I}$. It is straightforward to check the claims about the complexities for this protocol. Using Theorem 3.11 on page 149 we then get that there exists a protocol t-realizing $\text{Stagger}^{\mathcal{IO}}(\text{MultiRound}(\mathcal{F}_{\text{BA}}), 1)$ in the $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{BA}}, 1)$-hybrid model with initializing functionality $\mathcal{I}$, with the same complexities up to a constant factor. Except for insignificant syntactical differences

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

ready:
A party P_i can input $(syncid, \texttt{ready})$, where *syncid* is a synchronizing id. The meaning of this message is that P_i is ready for the synchronization with id *sgid*. How the message affects the functionality is described below.

all ready:
A party P_i can input $(syncid, \texttt{all-ready})$. The meaning of this message is that P_i thinks that all honest parties have by now input $(syncid, \texttt{ready})$.

output:
If in some round 1) some honest party have input $(syncid, \texttt{all-ready})$ and the value $(syncid, \texttt{term})$ occurs on the SIT, or 2) all honest parties have input $(syncid, \texttt{all-ready})$, then output $(syncid, \texttt{term})$ to all honest parties.

incorrect inputs:
If some honest party inputs $(syncid, \texttt{all-ready})$ and the last honest party to into $(syncid, \texttt{ready})$ did so in the previous round or later or some honest party used the same synchronization id *syncid* twice, then break down.

Figure 7.3: A functionality for synchronizing.

$\text{Stagger}^{\mathcal{IO}}(\text{MultiRound}(\mathcal{F}_{\text{BA}}), 1)$, $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{BA}}, 1)$ and $\mathcal{F}_{\text{SMBA}}$ are identical, which proves the theorem. □

7.4 Realizing $\text{MultiRound}(\mathcal{F}_{\text{BA}})$ using $\mathcal{F}_{\text{MBA}}$ and $\mathcal{F}_{\text{sync}}$

In this section we show how to realize $\text{MultiRound}(\mathcal{F}_{\text{BA}})$ using a MBA functionality $\mathcal{F}_{\text{MBA}}$ with simultaneous activation and one-round staggered termination plus an ideal functionality $\mathcal{F}_{\text{sync}}$ for synchronizing.

Definition 7.1 *We define the functionality $\mathcal{F}_{\text{MBA}}$ to be the $\mathcal{F}_{\text{SMBA}}$ functionality with the modification that it requires simultaneous activation. I.e. if two honest parties input $(baid, b_i)$ and $(baid, b_j)$ is different rounds, it breaks down.*

The functionality $\mathcal{F}_{\text{MBA}}$ is the functionality which we discussed in the introduction, having the problem that though it can be efficiently realized it can *not* be efficiently used. This is not a problem here, as the section mainly serves as a technical step towards a higher goal.

The synchronizing functionality is given in Fig. 7.3. Notice that we require that when an honest party inputs `all-ready`, then all parties are not just ready, but were ready in the previous round. The reason for this requirement is that it guarantees that the `ready` and `all-ready` commands do not overlap in a valid input sequence of $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$. We return to this point in Section 7.5.

Consider the protocol in Fig. 7.4 on the next page. We clearly have that:

Theorem 7.5 *π_{SSBA} realizes $\text{MultiRound}(\mathcal{F}_{\text{BA}})$ in the $(\mathcal{F}_{\text{MBA}}, \mathcal{F}_{\text{sync}})$-hybrid model.*

Protocol π_{SSBA}

The protocol runs with party $P_1, \ldots, P_n$ in the $(\mathcal{F}_{\text{MBA}}, \mathcal{F}_{\text{sync}})$-hybrid model. On input $(baid, m_i)$, for $m_i \in \{0,1\}^*$, the party P_i proceeds as follows:

1. Input $(baid, m_i)$ to $\mathcal{F}_{\text{MBA}}$.
2. On output $(\texttt{decide}, m)$ from $\mathcal{F}_{\text{MBA}}$, input $(baid, \texttt{ready})$ to $\mathcal{F}_{\text{sync}}$.
3. Wait one round.
4. Input $(baid, \texttt{all-ready})$ to $\mathcal{F}_{\text{sync}}$.
5. On output $(baid, \texttt{term})$ from $\mathcal{F}_{\text{sync}}$, output $(\texttt{decide}, m)$.

Figure 7.4: A simple simultaneous BA protocol.

7.5 Realizing $\mathcal{F}_{\text{SGR}}$

Before we consider staggered multi BA, we consider the problem of reducing a known staggering gap.

In several of the following functionalities we allow the functionalities to have a staggered termination. We model this by letting the input on the SIT determine which parties terminate first. We make the following definition: Assume that a set of outputs $\{(id, y_i)\}_{i \in H}$ is defined for the honest parties. When we say "output $\{(id, y_i)\}_{i \in H}$, letting the adversary specify the output round and a one round termination gap" we mean the following: Output $(\texttt{specify}, id)$ on the SOT. Then in the first round where a value (id, S) occurs on the SIT, interpret S as a set $S \subseteq [n]$. Then output (id, y_i) to P_i for $i \in [n] \setminus S$ in the round where (id, S) arrived and output (id, y_i) to P_i for $i \in S$ in the following round. To model guaranteed termination we sometimes required that the adversary specifies the output round within r rounds, by which we mean that if r rounds after $(\texttt{specify}, id)$ was output no value (id, S) arrived, then behave as if $(id, \emptyset)$ arrived.

In Fig. 7.5 on the following page a functionality for reducing a staggering gap with known bound to a staggering gap of one round is given. In Fig. 7.6 on the next page a realization in the real-life model is given (appears in [BE03]) and in Fig. 7.5 on the following page a realization in the $\mathcal{F}_{\text{T-sig}}^{t+1,r}$-hybrid model is given, where $\mathcal{F}_{\text{T-sig}}^{t+1,r}$ is the $\mathcal{F}_{\text{T-sig}}$ functionality with construction threshold $t+1$ and round complexity r for signing.

Theorem 7.6 *For $t \leq (n-1)/3$ the protocol π_{USGR} t-realizes $\mathcal{F}_{\text{SGR}}$ in the real-life model with round-complexity $r = 1$.*

Proof. We consider one staggering gap id *sgid*. Define $R_{\min}$ to be the round where the last honest party inputs $(sgid, \texttt{ready})$ and let $R_{\max}$ be the round where the last honest party input $(sgid, \texttt{all-ready})$, plus 1. It is enough to argue that if the case incorrect inputs do not apply to *sgid*, then it holds for *sgid* that no honest party outputs $(\texttt{terminate}, sgid)$ before round $R_{\min}$ or after $R_{\max}$ and that the staggering gap is one round.

To see that no honest party outputs $(\texttt{terminate}, sgid)$ before round $R_{\min}$ observe that because at most t parties are corrupted, no party receives $(\texttt{terminate}, sgid)$ from $t+1$ distinct

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

The functionality $\mathcal{F}_{\text{SGR}}$ runs with parties $P_1, \ldots, P_n$ and is parameterized by a round complexity r. It proceeds as follows:

ready:
A party P_i can input $(sgid, \texttt{ready})$, where $sgid$ is a staggering gap id. The meaning of this message is that P_i is ready for the staggering gap with id $sgid$ to be closed. How the message affects the functionality is described below.

all ready:
A party P_i can input $(sgid, \texttt{all-ready})$. The meaning of this message is that P_i thinks that all honest parties have by now input $(sgid, \texttt{ready})$.[a]

output:
If in some round 1) all honest parties have input $(sgid, \texttt{ready})$ and the value $(sgid, \texttt{term})$ occurs on the SIT, or 2) all honest parties have input $(sgid, \texttt{all-ready})$, then output $(sgid, \texttt{term})$ to all honest parties, letting the adversary specify the output round (within r rounds) and a one-round termination gap.

incorrect inputs:
If some honest party inputs $(sgid, \texttt{all-ready})$ before all honest parties have input $(sgid, \texttt{ready})$ or some honest party uses the same staggering gap id $sgid$ twice, then and break down.

[a] We sometime use inputs of the form $(sgid, \Delta)$ to $\mathcal{F}_{\text{SGR}}$, for an integer Δ, as a 'macro' for the following input behavior: First input $(sgid, \texttt{ready})$, then wait Δ rounds and input $(sgid, \texttt{all-ready})$.

Figure 7.5: A functionality for reducing a staggering gap with known bound to a staggering gap of one round.

Protocol π_{USGR}

The protocol runs with parties $P_1, \ldots, P_n$ in the real-life model. Each party P_i executes the following rules in parallel:

1. On input $(sgid, \texttt{all-ready})$, send $(\texttt{terminate}, sgid)$ to all parties.
2. Upon receiving a message $(\texttt{terminate}, sgid)$ from $t + 1$ distinct parties, send $(\texttt{terminate}, sgid)$ to all parties.
3. Upon receiving $(\texttt{terminate}, sgid)$ from $n - t$ distinct parties, terminate with output $(\texttt{terminate}, sgid)$.

Figure 7.6: The unauthenticated staggering gap reduction protocol.

parties before an honest party sent $(\texttt{terminate}, sgid)$, because of Rule 1, and the first honest party sends $(\texttt{terminate}, sgid)$ according to Rule 1 no earlier then in round $R_{\min}$, as incorrect inputs would otherwise apply. In particular, no honest party receives $(\texttt{terminate}, sgid)$ from $n - t$ parties before round $R_{\min}$. We then argue that all parties terminates no later than in round $R_{\max}$. This follows by the fact that the last honest party sends $(\texttt{terminate}, sgid)$ in round $R_{\max} - 1$ and so all honest parties received at least $n - t$ such messages by round $R_{\max}$. Finally, we argue that the staggering gap is one round. Let R be the first round in which any

Protocol π_{ASGR}

The protocol runs with parties $P_1, \ldots, P_n$ in the $\mathcal{F}_{\text{T-sig}}^{t+1,r}$-hybrid model. Each party P_i executes the following rules in parallel:

1. On input $(sid, \texttt{all-ready})$, input $(\texttt{sign}, sid)$ to $\mathcal{F}_{\text{T-sig}}^{t+1,r}$.
2. Upon $\mathcal{F}_{\text{T-sig}}^{t+1,r}$ outputting $(\texttt{signed}, sid)$ or $(\texttt{transfered}, sid)$, input $(\texttt{transfer}, sid)$ to $\mathcal{F}_{\text{T-sig}}^{t+1,r}$ and output $(\texttt{terminate}, sid)$.

Figure 7.7: The authenticated staggering gap reduction protocol.

honest party P_i outputs $(\texttt{terminate}, sid)$. This means that P_i received $(\texttt{terminate}, sid)$ from $n - t$ distinct parties by round R. Since at most t of these parties are corrupted $n - t - t \geq t + 1$ of them sent $(\texttt{terminate}, sid)$ to all parties, so by round R all honest parties received $(\texttt{terminate}, sid)$ from at least $t + 1$ distinct parties. Therefore all honest parties sent $(\texttt{terminate}, sid)$ to all honest parties by the end of round R, implying that all honest parties terminate no later than in round $R + 1$. □

Theorem 7.7 *For $t \leq (n-1)/2$ the protocol π_{ASGR} t-realizes $\mathcal{F}_{\text{SGR}}$ in the $\mathcal{F}_{\text{T-sig}}^{t+1,r}$-hybrid model with round complexity r.*

Proof. Define $R_{\min}$ as in the above proof and let $R_{\max}$ be the round where the last honest party inputs $(sid, \texttt{all-ready})$, plus r. It is enough to argue that if the case incorrect inputs do not apply to sid, then it holds for sid that no honest party outputs $(\texttt{terminate}, sid)$ before round $R_{\min}$ or after $R_{\max}$ and that the staggering gap is one round.

To see that no honest party outputs $(\texttt{terminate}, sid)$ before round $R_{\min}$ observe that the first honest party inputs $(\texttt{sign}, sid)$ to $\mathcal{F}_{\text{T-sig}}^{t+1,r}$ in round $R_{\min}$ and the construction threshold is $t + 1 > t$, where t is the maximal number of corrupted parties. So, $\mathcal{F}_{\text{T-sig}}^{t+1,r}$ will not output $(\texttt{signed}, sid)$ or $(\texttt{transfered}, sid)$ before round $R_{\min}$ and no party terminates before receiving $(\texttt{signed}, sid)$ or $(\texttt{transfered}, sid)$ from $\mathcal{F}_{\text{T-sig}}^{t+1,r}$. We then argue that all parties terminate no later than in round $R_{\max}$. This follows by the fact that the last honest party inputs $(\texttt{sign}, sid)$ to $\mathcal{F}_{\text{T-sig}}^{t+1,r}$ in round $R_{\max} - r$ and so $\mathcal{F}_{\text{T-sig}}^{t+1,r}$ outputs $(\texttt{signed}, sid)$ to all parties no later than in round $R_{\max}$. That the staggering gap is one round is trivial, given that the transfer command on $\mathcal{F}_{\text{T-sig}}^{t+1,r}$ has round complexity 1. □

7.5.1 Realizing $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$ using $\mathcal{F}_{\text{SGR}}$

Before proceeding we observe that one can trivially realize $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$ using $\mathcal{F}_{\text{SGR}}$. The construction $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$ basically makes two changes to $\mathcal{F}_{\text{sync}}$: First of all, what was synchronous rounds before are allowed to have one-round staggering and the output round is now specified by the adversary. Since functionality $\mathcal{F}_{\text{sync}}$ only has few restrictions on the order of the inputs the first change does not matter much. Notice however, that because we have required that there is a margin of a round from the last honest party inputs `ready` until the first honest party inputs `all-ready`, the added one-round staggering will not

allow that some honest party inputs `all-ready` before all honest parties have input `ready`, or to be more precise: If any honest party do this, then $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$ breaks down, as would $\mathcal{F}_{\text{SGR}}$. The second change introduced by the one-round staggering is that what was before synchronous output rounds with guaranteed delivery are now an output round specified by the adversary along with a one-round staggering, as in $\mathcal{F}_{\text{SGR}}$. Using the above observations, it is straight-forward to verify the following theorem:

Theorem 7.8 *There exists a protocol with no communication realizing* $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$ *in the* $\mathcal{F}_{\text{SGR}}$*-hybrid model using one call to* $\mathcal{F}_{\text{SGR}}$ *per call to* $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$.

7.6 Realizing $\mathcal{F}_{\text{SMBA}}$

In the previous section we gave a protocol realizing the ideal functionality $\text{MultiRound}(\mathcal{F}_{\text{BA}})$ in the $(\mathcal{F}_{\text{MBA}}, \mathcal{F}_{\text{sync}})$-hybrid model. Assume then that we have a protocol π_{MBA} which t-realizes $\mathcal{F}_{\text{MBA}}$ in the real-life model (with initializing functionality $\mathcal{I}$) and we have that $\pi_{\text{SSBA}}[\pi_{\text{MBA}}/\mathcal{F}_{\text{MBA}}]$ t-realizes $\text{MultiRound}(\mathcal{F}_{\text{BA}})$ in the $\mathcal{F}_{\text{sync}}$-hybrid model (with initializing functionality $\mathcal{I}$). Using Theorem 3.11 on page 149 we then have that the composed protocol $\text{Stagger}^{\mathcal{IO}}(\pi_{\text{SSBA}}[\pi_{\text{MBA}}/\mathcal{F}_{\text{MBA}}], 1)$ t-realizes $\text{Stagger}^{\mathcal{IO}}(\text{MultiRound}(\mathcal{F}_{\text{BA}}), 1)$ in the $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$-hybrid model (with initializing functionality $\mathcal{I}$). Using Theorem 7.8 and the composition theorem we then get a protocol $\text{Stagger}^{\mathcal{IO}}(\pi_{\text{SSBA}}[\pi_{\text{MBA}}/\mathcal{F}_{\text{MBA}}], 1)$ t-realizing $\text{Stagger}^{\mathcal{IO}}(\text{MultiRound}(\mathcal{F}_{\text{BA}}), 1)$ in the $\mathcal{F}_{\text{SGR}}$-hybrid model (with initializing functionality $\mathcal{I}$).[14] Finally, let π_{SGR} be a protocol which t-realizes $\mathcal{F}_{\text{SGR}}$ in the real-life model (with initializing functionality $\mathcal{J}$)[15] and we have that $\text{Stagger}^{\mathcal{IO}}(\pi_{\text{SSBA}}[\pi_{\text{MBA}}/\mathcal{F}_{\text{MBA}}], 1)[\pi_{\text{SGR}}/\mathcal{F}_{\text{SGR}}]$ t-realizes $\text{Stagger}^{\mathcal{IO}}(\text{MultiRound}(\mathcal{F}_{\text{BA}}), 1)$ in the real-life model (with initializing functionality $[\mathcal{I}, \mathcal{J}]$). Since $\text{Stagger}^{\mathcal{IO}}(\text{MultiRound}(\mathcal{F}_{\text{BA}}), 1)$ and $\mathcal{F}_{\text{SMBA}}$ are identical except for some insignificant syntactical differences.

Notice that the construction $\text{Stagger}^{\mathcal{IO}}(\pi_{\text{SSBA}}[\pi_{\text{MBA}}/\mathcal{F}_{\text{MBA}}], 1)[\pi_{\text{SGR}}/\mathcal{F}_{\text{SGR}}]$ does not have much overhead. Compared to running π_{MBA} there will be one call to π_{SGR}, the complexity of which is within our stated goal for the complexity of one BA for the candidates in Fig. 7.6 on page 234 and Fig. 7.7 on the preceding page, and there is a small constant blowup in round complexity, but only the original messages of π_{MBA} and π_{SGR} are sent in the protocol. We already have sufficiently efficient candidates for π_{SGR}, so our goal from now on will be to realize $\mathcal{F}_{\text{MBA}}$ sufficiently efficient.

Theorem 7.9 *Assume that there exist protocols* π_{MBA} *and* π_{SGR} *which* t*-realizes* $\mathcal{F}_{\text{MBA}}$ *in the real-life model (with initializing functionality* $\mathcal{I}$*) respectively* t*-realize* $\mathcal{F}_{\text{SGR}}$ *in the real-life model (with initializing functionality* $\mathcal{J}$*). Then there exists a protocol* t*-realizing* $\mathcal{F}_{\text{SMBA}}$ *in the real-life model (with initializing functionality* $\mathcal{J}$*) with communication complexity and round complexity being the sum of the communication complexities and round complexities of* π_{MBA} *and* π_{SGR}*, within a constant factor.*

[14] Here we also use $\text{Stagger}^{\mathcal{IO}}(\pi_{\text{SSBA}}[\pi_{\text{MBA}}/\mathcal{F}_{\text{MBA}}], 1)$ to denote the version of $\text{Stagger}^{\mathcal{IO}}(\pi_{\text{SSBA}}[\pi_{\text{MBA}}/\mathcal{F}_{\text{MBA}}], 1)$ using $\mathcal{F}_{\text{SGR}}$ instead of $\text{Stagger}^{\mathcal{IO}}(\mathcal{F}_{\text{sync}}, 1)$.

[15] Possible candidates appear in Fig. 7.6 on page 234 and Fig. 7.7 on the preceding page

7.7 Special Phase-King BA Protocols

In this section we present the so-called special phase-king BA protocols which will be the basis for our MBA protocol. A special phase-king BA protocol proceeds in phases of `phase-len` rounds, and in round `king-round` in each phase the action of the protocol might depend on the index of some party, called the king. The 'quality' of this king will have an effect on the termination guarantee of the protocol, but not its correctness, so we shall not be concerned with how the king is chosen yet — we assume that in each round r, where (r mod `phase-len`) = `king-round`, some king $K_{ph,i}$ is supplied as input to each P_i — here ph denotes the phase (r div `phase-len`). The unauthenticated phase-king BA protocol in Fig. 7.9 on page 240 runs in the real-life model and the authenticated phase-king BA protocol in Fig. 7.9 on page 240 runs in the $\mathcal{F}_{\text{T-sig}}^{t+1}$-hybrid model. For both protocols we prove that they are correct BA protocols with the following three additional properties:

all-zero-non-participation-resilient:
: By which we mean that if all honest parties which enter the BA do so with value 0, then the protocol decides on 0 even if some honest parties do not enter the BA.

  In the following we say that the protocol was activated correctly if all honest parties that enter the BA do so in the same round and if some honest parties enter the BA with value 1, then all honest parties enter the BA. The following requirements are for BAs activated correctly.

decision-after-consensus-on-good-king:
: By which we mean that if for some phase ph it holds that $K_{ph,i} = K_{ph,j}$ for all honest parties P_i and P_j, and if $P_{K_{ph}}$ (where K_{ph} denotes the common value of the $K_{ph,i}$) was honest in round `king-round` $- 1$, then all honest parties that entered the BA already decided or will decide before the end of the current phase.

one-phase-staggering:
: By which we mean that if phase p is the first phase in which an honest party that entered the BA decides, then no honest party which entered the BA decides later than in phase $p + 1$.

agreement-cheap:
: By which we mean that if all honest parties that enter the BA do so with the same value, then all honest parties which entered the BA decides in the first phase.

To be precise we formalize these requirements by requiring that the protocols implement the functionality in Fig. 7.8 on the next page.

7.7.1 The Unauthenticated Protocol

The unauthenticated special phase-king BA protocol in Fig. 7.9 on page 240 is derived from the Feldman-Micali [FM97] protocol. The main reason for using the Feldman-Micali protocol is that it is trivial to make all-zero-non-participation-resilient. The observation is that if in the Feldman-Micali protocol all honest parties start with value 0, then all honest parties will send only 0-bits during the protocol. Therefore the convention that a silent processor

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

The functionality runs with parties $P_1, \ldots, P_n$ and is parameterized by a phase length `phase-len` $\in \mathbf{N}$, a phase length of the initial phase `init-len` $\in \mathbf{N}$ and a king round $0 \leq$ `king-round` $<$ `phase-len`.[a]

all-zero-non-participation-resilient, agreement-cheap:

Throughout the description, let H denote the set of indices of honest parties.

If in some round an honest party input $(baid, b_i)$ for $b_i \in \{0,1\}$, let I be the indices of those honest parties that input $(baid, b_j)$ for $b_i \in \{0,1\}$ in that round. If $I \neq H$ and $b_i = 1$ for some $i \in I$, then break down. Furthermore, if any honest party used the BA id $baid$ in an earlier round, then break down. Below we describe how the functionality deals with this one specific activation. All other activations are dealt with independently and in the same way.

Let $\texttt{round}_{baid} \leftarrow -\texttt{init-len}$ and let $\texttt{last-term-round}_{baid} \leftarrow \infty$ — for notational convenience we do not index variables by the BA id $baid$ in the following.

- If $b_i = 0$ for all $i \in I$, let `result` $\leftarrow 0$ and `last-term-round` $\leftarrow$ `phase-len`.
- If $b_i = 1$ for all $i \in I$, let `result` $\leftarrow 1$ and `last-term-round` $\leftarrow$ `phase-len`.[b]
- If none of the above rules apply, then let `result` $\leftarrow \perp$.

do round:

If in some round all P_i for $i \in I \cap H$ input (`do round`, $baid$), then let `round` $\leftarrow$ `round` $+ 1$.

If in some round some honest party inputs (`do round`, $baid$) and some P_i for $i \in I \cap H$ do not input (`do round`, $baid$), then break down.

decision-after-consensus-on-good-king:

If in some round, where `round` ≥ 0 and (`round` mod `phase-len`) $=$ `king-round`, a value (`king`, $baid, K$) is input to P_i for $i \in H \cap I$, and $K \in [n]$ and P_K were honest in the previous round, then let `last-term-round` $\leftarrow$ `phase-len` $\cdot\, ph$, where $ph = \lceil$`round`$/$`phase-len`$\rceil$ denotes the current phase.

decision rule, one-phase-staggering:

If in any round a value (`decide`, $baid, b$) is input on the SIT and `result` $= \perp$, then let `result` $\leftarrow b$.

If in any round (`term`, $baid, i$) is input on the SIT for $i \in I \cap H$, then proceed as follows: If `result` $= \perp$, then let `result` $\leftarrow 0$. Then output (`decide`, $baid$, `result`) to P_i, let $I \leftarrow I \setminus \{i\}$ and let `last-term-round` $\leftarrow \min\{$`round`$+$`phase-len`, `last-term-round`$\}$.

If in any round `round` $=$ `last-term-round`, then proceed as if (`term`, $baid, i$) was received for $i \in I$.

[a] Here `init-len` is the length of some possible initial phase, typically for an initial round for distributing the parties' inputs.

[b] As by definition $|I| > 0$, the last two rules are exclusive.

Figure 7.8: The special-phase king BA functionality with phase length `phase-len` and king-round `king-round`.

'sends' 0-bits makes the protocol all-zero-non-participation-resilient. To provide an intuition about the correctness of the protocol we describe how it is derived from the Feldman-Micali protocol. In the polling phase of the Feldman-Micali protocol party P_i holds a bit a_i, initially its input, which it sends to all parties. Then P_i tally incoming 1's, let A_i denote this tally. Then P_i computes its new bit as follows: If $A_i \in [0, n/3)$, then set $b_i = 0$; If $A_i \in [2n/3, n]$, then set $b_i = 1$; And if $A_i \in [n/3, 2n/3)$, then set $b_i = coin_i$, where $coin_i$ is a bit which P_i considers the coin value of the round where the new bit is computed. It is well-known and straight-forward to verify that if all honest parties start with the same a_i, then all honest parties end up with $b_i = a_i$. Furthermore, it is always the case that $|A_i - A_j| < n/3$ for all pairs of honest parties P_i and P_j, which in particular implies that if some honest party set $b_i = b$, in the update, then all honest parties set $b_i = b$ or set $b_i = coin_i$. This means that if $coin_i = b$ for all honest parties, then all honest parties will have $b_i = b$; We say that there is at least one good coin value, which, if the parties agree on it, will stabilize the network. The idea is then to replace the common coin-flip in [FM97] by a leader election. The elected leader, which we call the phase-king following [BGP89], will try to determine a good coin value and send it to all parties. If the phase-king, P_K say, has $A_K \in [0, n/3)$ or $A_K \in [2n/3, n]$ it is straight-forward to determine a good coin value, $coin_K = 0$ respectively $coin_K = 1$. If $A_K \in [n/3, 2n/3)$ the task is less straight-forward, but at least one of the coin values are good, so if another round of polling was done with this good coin value, let B_K be the tally, then the network would be stable and so $B_K \notin [n/3, 2n/3)$. Therefore the phase-king could pick a good coin value in the next phase. The rational is therefore to do a round of polling from the bits b_i, but using both the coin value 0 and the coin value 1 in parallel. This gives the phase-king two tallies B_K^0 and B_K^1, one from each of the parallel runs. In one of these, $B_K^{coin_{K,1}}$ say, the phase-king can determine a good coin value $coin_{K,2} \in \{0, 1\}$. The phase-king then sends $(coin_{K,1}, coin_{K,2})$ to all parties which set $B_i = B_i^{coin_{K,1}}$ and computes a bit c_i, using the coin $coin_{K,2}$ if $B_i \in [n/3, 2n/3)$. If the phase-king is honest this stabilizes the network. It is interesting to note that if the parties computed c_i by rerunning from the values a_i using two rounds of polling with coin $coin_{K,1}$ and then coin $coin_{K,2}$, then not even an honest phase-king would guarantee stabilizing the network. The two parallel runs serve as more than information gathering for the phase-king. They also commit the corrupted parties to a particular behavior on the first coin. With an eye on obtaining a randomized BA protocol with constant expected round complexity we let all parties act as phase-king before the phase-king is actually elected. This ensures that the adversary must corrupt a phase-king in the round before it is elected to avoid termination of the protocol, and this can be prevented by using a fair leader election protocol. We note that the protocol derived as described above is in some respects reminiscent of the ESPK protocol in [BDGK95].

We only give the part of the proof of correctness of π_{USPHAK} which differs essentially from the proof in [FM97]. The following lemma extends Claim T4-2 in [FM97].

Lemma 7.5 *If all honest parties are participating in phase p and have $a_i = a$ for $a \in \{0, 1\}$ at the beginning of the phase, then $e_i = a$ for all honest parties at the end of the phase.*

Proof. Assume first that $a_i = 0$ for all honest parties. This means that $A_i \leq (n-1)/3 < n/3$ for all honest parties, so $b_i = 0$ for all honest parties. This implies that $B_i^0 < n/3$ and

Protocol π_{USPHAK}

The protocol runs with parties $P_1, \ldots, P_n$ in the real-life model. The party P_i proceeds as follows for each *baid*:[a]

input:

Party P_i receives a bit a_i. In the following P_i executes one of the below rounds each time it receives input `do round`.

initial phase:

Let $a = a_i$ and send a to all parties.

polling:

Let A be the number of 1'a received.[b]

- If $A \in [0, n/3)$, then set $b = 0$.
- If $A \in [n/3, 2n/3)$, then set $b = \perp$.
- If $A \in [2n/3, n]$, then set $b = 1$.

Send b to all parties.

election:

king commit:

Let B^0 be the number of 1'a received and let B^1 be the number of 1'a and $\perp$'a received.[c] Then compute two coin values as follows:

1. If $B^0 \notin [n/3, 2n/3)$, then set $coin_1 = 0$. Otherwise set $coin_1 = 1$.[d]
2. If $B^{coin_1} \in [0, n/3)$, then set $coin_2 = 0$. Otherwise set $coin_2 = 1$.

Send $(coin_1, coin_2)$ to all parties.

king-round:

Expect an input $(\texttt{king}, K)$ from the environment. If such input does not arrive, then terminate.

[a] In the unauthenticated model the BA ids are only used to associate transfered data to specific instances of the protocol, and we drop the BA id *baid* in the description for clarity.

[b] Whenever P_i should receive a value from P_j according to the protocol and no value arrives, P_i will define the received value to equal the last bit ever received from P_j (0 if no bit was ever received).

[c] B^v would be the number of 1's received if before the last polling the coin value $\perp = v$ had been used.

[d] It is always the case that $B^{coin_1} \notin [n/3, 2n/3)$.

Figure 7.9(a) (cont. in Fig. 7.9(b) on the facing page): The Unauthenticated Special PHAse-King BA Protocol

$B_i^1 < n/3$ for all honest parties. In particular $B_i < n/3$ for all honest parties, which in turn gives us that $e_i = 0$ for all honest parties. The case where $a_i = 1$ for all honest parties follows equivalently. □

The following lemma is the equivalent to Claim T4-4 in [FM97].

Lemma 7.6 π_{USPHAK} *has decision-after-consensus-on-good-king.*

Protocol π_{USPHAK}

decision:

Let $(COIN_1, COIN_2)$ denote the two bits received from P_K in king commit.

Let $B = B^{COIN_1}$.

- If $B \in [0, n/3)$, then set $c = 0$.
- If $B \in [n/3, 2n/3)$, then set $c = COIN_2$.
- If $B \in [2n/3, n]$, then set $c = 1$.

Send c to all parties.

test for stable:

1. Let C be the number of 1's received.
   - If $C \in [0, n/3)$, then set $d = 0$.
   - If $C \in [n/3, 2n/3)$, then set $d = 1$.
   - If $C \in [2n/3, n]$, then set $d = 1$, output $(\texttt{decide}, 1)$ and terminate after having sent d to all parties.

   Send d to all parties.
2. Let D be the number of 1's received.
   - If $D \in [0, n/3)$, then set $e = 0$, output $(\texttt{decide}, 0)$ and terminate after having sent a to all parties.
   - If $D \in [n/3, 2n/3)$, then set $e = 0$.
   - If $D \in [2n/3, n]$, then set $e = 1$.

   Let $a = e$ and send a to all parties.
3. Go to Step polling.

Figure 7.9(b) (cont. from Fig. 7.9(a) on the facing page): The Unauthenticated Special PHAse-King BA Protocol

Proof. We first prove the following claims: 1) For all pairs (P_i, P_j) of honest parties $|\Sigma_i - \Sigma_j| < n/3$ for $\Sigma \in \{A, B, B^0, B^1, C, D\}$; 2) $B_i^{coin_{i,1}} \notin [n/3, 2n/3)$. To prove Claim 1 observe that since Σ_i and Σ_j are tallies of received 1s (and possibly $\perp$s) and all honest parties always send the same value to all parties it follows that $|\Sigma_i - \Sigma_j| \leq (n-1)/3 < n/3$. We note that honest parties sending the same bit to all honest parties includes the case where an honest party P_l is not participating because it terminated, in which case all honest parties defines the bit received from P_l to the same value. We then prove Claim 2. If $coin_{i,1} = 0$, then by the way $coin_{i,1}$ is defined $B_i^{coin_{i,1}} \notin [n/3, 2n/3)$. Assume then that $coin_{i,1} = 1$. If $A_j \notin [0, n/3)$ for all honest parties P_j, then $B_i^0 \geq n - (n-1)/3 > 2n/3$, a contradiction to $coin_{i,1} = 1$. So, we can assume that $A_j \in [0, n/3)$ for some honest party. By Claim 1 this implies that $A_j < 2n/3$ for all honest parties P_j, so all honest P_j sends $b_j \in \{0, \perp\}$. This implies that $B_i^1 \leq (n-1)/3 < n/3$, which completeness the proof of the claims.

Now, let K denote the common value of K_i for honest P_i. Since P_K was honest when $(coin_{K,1}, coin_{K,2})$ was sent and $K_i = K$, for some common value $K \in [n]$, for all honest

parties, it follows that $(COIN_{i,1}, COIN_{i,2}) = (COIN_1, COIN_2)$ for all honest parties, for some $(COIN_1, COIN_2) \in \{0,1\}^2$. Assume that $COIN_1 = 0$. By Claim 2 this means that $B_K^0 \notin [n/3, 2n/3)$. Assume first that $B_K^0 \in [0, n/3)$ (which implies that $COIN_2 = 0$). By Claim 1 this implies that $B_i^{COIN_1} = B_i^0 < 2n/3$ for all honest parties. Therefore $c_i \in \{0, COIN_2\} = \{0\}$ for all honest parties. Assume then that $B_K^0 \in [2n/3, n]$ (which implies that $COIN_2 = 1$). By Claim 1 this implies that $B_i^0 > n/3$ for all honest parties. Therefore $c_i \in \{1, COIN_2\} = \{1\}$ for all honest parties. It follows that if $COIN_1 = 0$, then $c_i = c_j$ for all pairs P_i and P_j of honest parties. Using Claim T4-5 of [FM97] this implies that by the end of the phase all parties will have terminated. Assume then that $COIN_1 = 1$. By Claim 2 this means that $B_K^1 \notin [n/3, 2n/3)$, and the proof then proceeds as for $COIN_1 = 0$. □

Theorem 7.10 *For $t \leq (n-1)/3$ the π_{USPHAK} protocol is a special phase-king BA protocol (t-realizes $\mathcal{F}_{\text{SPHAKIBA}}$) with the following complexities for one phase: 5 rounds, $6n^2$ bits of communication and message size 2 bits.*

Proof. Using Lemma 7.5 on page 239 and Lemma 7.6 on page 240 the proof in [FM97] can be extended to prove that π_{USPHAK} is a BA protocol with termination staggering gap at most one phase. Consider then the case where all honest parties have $a_i = 0$ as input. It is straight-forward to verify that in that case all honest parties will never send any value which is different from 0. By the way honest parties define a non-received bit it follows that if an honest party P_i is not participating in the protocol, then all participating honest parties will always defined the values received from P_i to be 0. Therefore π_{USPHAK} has all-zero-non-participation-resilience. It follows from inspection of the protocol that π_{USPHAK} is agreement-cheap. □

7.7.2 The Authenticated Protocol

We now describe an authenticated special phase-king protocol. It uses ideas from two previous BA protocols. As a basis we use the authenticated protocol from [Tou84]. As discussed in the introduction the parties sign the votes sent in the polling step, and a party has to send along signatures from enough other parties from the previous phase to justify its vote. There is one problem with this as we want the protocol to be all-zero-non-participation-resilient. We cannot as for the unauthenticated protocol just say that a non-participating party is virtually sending 0-bits, as the participating parties need the non-participating party's signature on these bits to justify its own vote. We therefore leave the polling framework and use the vote justifiers more in the spirit of how signatures are used in the BA protocol by Dolev and Strong from [DS82]. There, a party which holds a message m along with signatures from enough distinct honest parties can send the message and signatures to another party forcing that party to 'accept' the message as a potential outcome of the BA. We contrast this use to the justified-polling use of signatures by calling it 'influencing'.

The protocol is given in Fig. 7.10 on the next page. It is fairly straight-forward to check the agreement-cheap and the all-zero-non-participation-resilient properties. W.r.t. agreement and one-phase-staggering, the main observations to make is that for two honest parties P_i and P_j it holds that $|H_i - H_j| \leq 1$. This implies that if any honest party outputs 0

Protocol π_{ASPHAK}

The protocol runs with parties $P_1, \ldots, P_n$ in the $\mathcal{F}_{\text{T-sig}}^{t+1}$-hybrid model, where $\mathcal{F}_{\text{T-sig}}^{t+1}$ has round complexity `init-len` for initialization and round complexity r for signing. The party P_i proceeds as follows:

input:

Party P_i receives an input $(baid, b_i)$. In the following P_i executed one of the below rounds each time it receives input `do round`.

initial distribution:

If $b_i = 1$, then input $(\texttt{sign}, (baid, 0))$ to $\mathcal{F}_{\text{T-sig}}$. Set $p \leftarrow 0$.

influence:

Wait r rounds. If $\mathcal{F}_{\text{T-sig}}$ outputs $(\texttt{signed}, (baid, p))$, then input $(\texttt{transfer}, (baid, p))$ to $\mathcal{F}_{\text{T-sig}}$ and let $G \leftarrow 1$. Otherwise, let $G \leftarrow 0$.

election:

king commit:

1. If $\mathcal{F}_{\text{T-sig}}$ outputs $(\texttt{transfered}, (baid, p))$, then input $(\texttt{transfer}, (baid, p))$ to $\mathcal{F}_{\text{T-sig}}$.
2. Let J be the set of j for which $\mathcal{F}_{\text{T-sig}}$ output $(\texttt{transfered}, j, (baid, p))$.

king-round:

Expect an input $(\texttt{king}, K)$ from the environment. If such input does not arrive, then terminate.

decision:

If $K \in J$, then let $G \leftarrow G + 1$. If $G \geq 1$, then input $(\texttt{sign}, (baid, 2, p))$ to $\mathcal{F}_{\text{T-sig}}$.

test for stable:

1. If $\mathcal{F}_{\text{T-sig}}$ outputs $(\texttt{signed}, (baid, 2, p))$, then input $(\texttt{transfer}, (baid, 2, p))$ to $\mathcal{F}_{\text{T-sig}}$ and let $H \leftarrow 1$. Otherwise, let $H \leftarrow 0$.
2. Repeat twice: If $\mathcal{F}_{\text{T-sig}}$ outputs $(\texttt{transfered}, (baid, 2, p))$, then input $(\texttt{transfer}, (baid, 2, p))$ to $\mathcal{F}_{\text{T-sig}}$ and let $H \leftarrow H + 1$.
3. If $H \geq 2$, then input $(\texttt{sign}, (baid, p + 1))$ to $\mathcal{F}_{\text{T-sig}}$.
4. If $H = 0$, then output $(\texttt{decide}, baid, 0)$ and terminate.
5. If $H = 3$, then input $(\texttt{sign}, (baid, 2, p + 1))$ to $\mathcal{F}_{\text{T-sig}}$, output $(\texttt{decide}, baid, 1)$ and terminate.
6. Go to Step influence.

Figure 7.10: The Authenticated Special PHAse-King BA Protocol.

in phase p, then no honest party signed $(baid, p+1)$ and therefore all honest parties output 0 in the following phase, and if any honest party outputs 1 in phase p, then all honest parties signed $(baid, p + 1)$ and therefore, if all honest parties are participating, all honest parties output 1 in the following phase. To see that the protocol has decision-after-consensus-on-good-king, observe that if $K \in J$, then all parties will have $G \geq 1$ and will sign $(baid, 2, p)$, so all parties will decide on 1 in the current phase. If on the other hand $K \notin J$, then P_K did not receive $(\texttt{transfered}, (baid, p))$ in Step 2 in Step king commit, which implies that $G_i = 0$ for all honest parties before (and thus also after) Step decision. Therefore all parties will

decide on 0 in the current phase.

We introduce some notation used in the analysis. Let p' denote a possible value of the value of the variable p at P_i. For each variable Σ in the protocol we let $\Sigma_i^{p'}$ denote the value of Σ in P_i in the phase where $p_i = p'$. If $\Sigma_i^{p'}$ was not explicitly assigned we let $\Sigma_i^{p'} = \perp$.

Lemma 7.7 *π_{ASPHAK} is a valid BA protocol and has one-phase-staggering.*

Proof. Let p be the first phase in which an honest party decides and let P_i be a party that decides in that phase. It is straight-forward to verify that $|H_i^p - H_j^p| \leq 1$ for all pairs P_i and P_j of honest parties. Let p be the first phase in which an honest party outputs.

If some honest P_i decides on 0 in phase p, then $H_i^p = 0$, which implies that $H_j^p \leq 1$ for all honest parties P_j. Therefore no honest party decides on 1 in phase p and no honest party ever inputs $(\texttt{sign}, (baid, p{+}1))$ to $\mathcal{F}_{\text{T-sig}}$. Therefore $\mathcal{F}_{\text{T-sig}}$ will never output $(\texttt{signed}, (baid, p{+}1))$, which implies that all honest parties decide on 0 no later than in phase $p+1$.

If some honest P_i decides on 1 in phase p, then $H_i^p = 3$, which implies that $H_j^p \geq 2$ for all honest parties P_j. Therefore no honest party decides on 0 in phase p and all honest parties input $(\texttt{sign}, (baid, p{+}1))$ to $\mathcal{F}_{\text{T-sig}}$ in phase p. Therefore $\mathcal{F}_{\text{T-sig}}$ will output $(\texttt{signed}, (baid, p{+}1))$ in Step 1 of Step influence in phase $p{+}1$, which implies that all honest parties which reach that step will have $G_i^{p+1} \geq 1$ in Step decision and will input $(\texttt{sign}, (baid, 2, p+1))$ to $\mathcal{F}_{\text{T-sig}}$. As concluded above, all honest parties not reaching Step decision in phase $p+1$ terminated in phase p with output 1, and therefore also input $(\texttt{sign}, (baid, 2, p{+}1))$ to $\mathcal{F}_{\text{T-sig}}$. Therefore all honest parties reaching Step 1 in Step test for stable in phase $p+1$ sees $\mathcal{F}_{\text{T-sig}}$ output $(\texttt{signed}, (baid, 2, p+1))$. It is easy to verify that this implies that those parties terminate with output $(\texttt{decide}, baid, 1)$ in phase $p+1$. □

Lemma 7.8 *π_{ASPHAK} has decision-after-consensus-on-good-king.*

Proof. Assume that in phase p the king is good, i.e. $K_i^p = K_j^p = K$ for all honest parties P_i and P_j, and P_K was honest in Step 3 in Step king commit in phase p. We have to prove that honest parties terminate no later than by the end of phase p. If some honest party terminated before phase p, the claim follows by one-phase-staggering. We can therefore assume that all honest parties reach phase p.

Assume first that after Step decision in phase p we have that $G_i = 0$ for all honest parties P_i. This clearly implies that all honest parties terminate with output 0 in phase p. Assume therefore that some honest party P_i has $G_i > 0$ after Step decision in phase p. We look at two cases. *Case 1:* If P_i had $G_i = 0$ before Step decision in phase p, then $K_i^p \in J_j^p$. Since $K_i^p = K_j^p = K$ and P_K was honest in Step 3 in Step king commit in phase p it follows that $K_j^p \in J_j^p$ for all honest P_j. Therefore $G_j^p \geq 1$. *Case 2:* If P_i had $G_i \geq 1$ before Step decision in phase p, then P_i had $G_i \geq 1$ in Step 1 in Step king commit in phase p. Therefore P_i input $(\texttt{transfer}, (baid, p))$ to $\mathcal{F}_{\text{T-sig}}$ in that Step and P_K received $(\texttt{transfered}, (baid, p))$ in Step 2 in Step king commit in phase p. Since P_K was honest in that Step this implies that P_K input $(\texttt{transfered}, (baid, p))$ to $\mathcal{F}_{\text{T-sig}}$ which implies that $G_j^p \geq 1$ after Step decision for all honest P_j. In both cases we have that $G_j^p \geq 1$ after Step decision in phase p for all honest P_j. It is straight-forward to verify that this implies that all parties will terminate with output $(\texttt{decide}, baid, 1)$ in phase p. □

Theorem 7.11 *For $t \leq (n-1)/2$ the π_{ASPHAK} protocol is a special phase-king BA protocol (t-realizes $\mathcal{F}_{\text{SPHAKIBA}}$) with the following complexities for one phase (when $\mathcal{F}_{\text{T-sig}}$ is realized with a protocol with communication complexity k bits and round complexity r for signing): $6 + r$ rounds, $8kn^2$ bits of communication and message size k bits.*

Proof. That the π_{ASPHAK} protocol is a special phase-king authenticated BA protocol follows from Lemma 7.7 on the facing page and Lemma 7.8 on the preceding page and the observation that the π_{ASPHAK} protocol is trivially all-zero-non-participation-resilient and agreement-cheap. □

7.8 Realizing $\mathcal{F}_{\text{MBA}}$

We are now ready to realize the $\mathcal{F}_{\text{MBA}}$ functionality. We start by sketching the main idea behind the protocol, assuming that we have at our disposal any special phase-king BA protocol — i.e. we run in the $\mathcal{F}_{\text{SPHAKIBA}}$-hybrid model.

7.8.1 Introduction

Consider first the following protocol: Let all parties hold a round counter r. Each time a party receives input $(baid, b)$ it inputs $(baid, b)$ to $\mathcal{F}_{\text{SPHAKIBA}}$ and in the first round r, where $r + \texttt{init-len} \bmod \texttt{phase-len} = 0$, the party adds $baid$ to a set `Running`. Then in each round, input $(\texttt{do round}, baid)$ to $\mathcal{F}_{\text{SPHAKIBA}}$ for $baid \in \texttt{Running}$. In each round r, where $r \bmod \texttt{phase-len} = \texttt{king-round}$ the party inputs $(\texttt{king}, baid, K)$ to $\mathcal{F}_{\text{SPHAKIBA}}$, for some phase king K. The parties can use a fixed predetermined schedule of kings, say a round-robin schedule $1, 2, \ldots, n, 1, 2, \ldots$ to make sure to agree on the king in all phases. If $\mathcal{F}_{\text{SPHAKIBA}}$ ever outputs $(\texttt{decide}, baid, b)$ then the party outputs $(\texttt{decide}, baid, b)$ and removes $baid$ from `Running`.

This protocol already goes a long way. In each round where P_K is honest all BAs in `Running` will terminate in their current phase. This means that no BA runs for more than $f+1$ phases, where f is the number of corrupted parties in the execution. Unfortunately, the worst case is that *all* BAs run for $f+1$ phases. Assume namely that the parties $P_1, \ldots, P_f$ are corrupted and that all BAs are activated in a phase where $K = 1$. Then the adversary can force all BAs to run for $f + 1$ phases. There is no trivial solution to this attack. The parties cannot just start the round-robin schedule of kings from where they left of in the previous activation: Because of the staggered termination, they do not agree on which party was the last king. On the other hand, if the parties make too long skips in the schedule, exactly the honest parties might be skipped, as above. A solution to this problem is given in [BDGK95]: Simply run a BA after each activation to determine where to start the round-robin in the next activation.

Actually, a BA will not do as the parties do not hold a bit as input. When the parties terminate a given BA, each party P_i observed an integer p_i being the number of phases that the BA took to terminate at P_i. The parties know that $|p_i - p_j| \leq 1$ for all honest parties P_i and P_j and the parties essentially want to pick one of these integers. The problem is that even though P_i knows that $|p_i - p_j| \leq 1$ for all honest P_j, it does not know whether

$p_j = p_i - 1$ or $p_j = p_i + 1$. So, it is not clear from P_i's view which bit it is that should be input to the BA. And, it does not help to do a BA on say all three values $p_i - 1, p_i, p_i + 1$ — if $p_j = p_i + 1$ then P_j will be running other BAs. There seems to be a nasty regression here. The solution taken in [BDGK95] is based on the observation that $1 \leq p_i \leq f + 1$, as all parties are guaranteed to terminate in the phase where the first honest party is king, which happens no later than in phase $f + 1$. Since $f \leq t$, the parties can solve the problem by running $t + 1$ BAs, with id's $(baid, 0), (baid, 1), \ldots, (baid, t)$ say. We could e.g. let each party P_i input 1 to the BAs with index $(baid, p_i - 1)$ and $(baid, p_i)$ and then let each party pick as output the largest p for which $(baid, p)$ output 1. By the agreement condition on each BA the parties will agree on p. Let $p_{\min} = \min_i\{p_i\}$ and let $p_{\max} = \max\{p_i\}_i$. We know that $p_{\min} \geq p_{\max} - 1$. Therefore all parties input 1 to $(baid, p_{\min})$, so $p \geq p_{\min}$. Furthermore, no honest party input 1 to $(baid, p')$ for $p' > p_{\max}$, so $p \leq p_{\max}$. So, p was the input of one of the parties. Each party started out with an input p_i and the promise that $|p_i - p_j| \leq 1$ and picked an input of one of the honest parties. This problem was dubbed the Unknown Binary Set Consensus (UBC) problem in [BDGK95].

Given the outcome of the UBC the parties can determine a phase where an honest party terminated. Let P_i be the king of that phase; Clearly P_i was not too bad, so the parties start the next schedule from P_i, .

There is one issue with this, however: Running the UBC *after* the BA will not do, as the output of the UBC should then actually be a round where an honest party terminated *the UBC* as to not reuse the sequence of kings used in running the UBC when the next BA is run. In [BDGK95] this is handled by starting the UBC when the next BA is activated. Then two independent round robin schedules, R_0 and R_1, are used, running $R_{A \bmod 2}$ during the A'th activation, which includes a UBC resolving where $R_{(A-1) \bmod 2}$ was terminated. This is essentially the protocol from [BDGK95].

We present the protocol along with an optimization. The optimization is obtained by constructing a UBC protocol which runs only a constant number of BAs. To the best of the author's knowledge the previously most efficient UBC protocol for the optimal resiliences, $3t < n$ for the unauthenticated model and $2t < n$ for the authenticated model, was the one described above.[16] This is a significant improvement. First of all the communication complexity drops from being linear dependent on the size of the a priori set $\{1, \ldots, t + 1\}$, from which parties might receive inputs, to being constant in this set. Equally important, from a theoretical stand point it is interesting to resolve what is the lower bound on the complexity of the UBC problem, in BAs say. From the viewpoint of each party at most three outcomes are possible, so there is nothing which excludes a reduction from the UBC problem to two BAs. We use four.

7.8.2 Unknown Binary Set Consensus

In [BDGK95] the Unknown Binary Set Consensus (UBC) problem is introduced. Each honest party P_i starts with a value $m_i \in \mathbf{N}$ and it is guaranteed that $|m_i - m_j| \leq 1$ for all honest parties P_i and P_j. As in the BA problem we require that all honest parties output the same value, and that the output value was the input of some honest party. In [BDGK95] a solution

[16] This has been acknowledged by the third author of [BDGK95] in a private correspondence.

to the problem is given for $n > 4t$ and $n > 3t$. Both protocols are phase-king protocol; The solution for $n > 4t$, using techniques from [BGP92], uses messages of size $\log(m)$, where m is the largest input of an honest party, and the solution for $n > 3t$ uses messages of size b, where b is an a priori bound on the size of m_i. Both results are basically obtained by reducing the UBC problem to $\log(m)$ respectively b BA problems. In this section we improve on these solutions by presenting a reduction of the UBC problem to 4 BA problems.

Definition 7.2 *The functionality $\mathcal{F}_{\text{UBC}}$ is defined as $\mathcal{F}_{\text{BA}}$, see Fig. 3.17 on page 91, with the modification that it breaks down if two honest parties input $(baid, m_i)$ respectively $(baid, m_j)$, where $|m_i - m_j| > 1$ when m_i and m_j are interpreted as integers..*

Theorem 7.12 *$\mathcal{F}_{\text{UBC}}$ can be realized in the $\mathcal{F}_{\text{BA}}$-model with 4 calls to $\mathcal{F}_{\text{BA}}$ per call to $\mathcal{F}_{\text{UBC}}$.*

Proof. The party P_i proceeds as described below. Given input $(baid, m_i)$ for $m_i \in \{0,1\}^*$, let v_i be the value of m_i when interpreted as an integer. Compute $w_i = v_i \bmod 4$. Then the parties enter four BAs with indices $(baid, 0), \ldots, (baid, 3)$. Party P_i enters $(baid, w_i)$ and $(baid, w_i - 1 \bmod 4)$ with value 1 and enters $(baid, w_1 + 1 \bmod 4)$ and $(baid, w_1 + 2 \bmod 4)$ with value 0. Receive $(\texttt{decide}, (baid, l), d_l)$ for $l \in \{0, \ldots, 3\}$. Each party P_i computes the largest integer r for which $r_i \in \{v_i - 1, v_i, v_i + 1\}$ and $d_{r_i \bmod 4} = 1$, and outputs r_i.

Let V_i denote the set of $v \in \{v_i - 1, v_i, v_i + 1\}$ for which $d_{v \bmod 4} = 1$. Let $v_{\min}$ be the smallest integer such that some honest party has $v_i = v_{\min}$. We know that $v_i \in \{v_{\min}, v_{\min}+1\}$ for all honest P_i, so $v_{\min} \in \{v_i - 1, v_i\}$, which implies that each honest party P_i enters $(baid, v_{\min} \bmod 4)$ with input 1 and enters $(baid, v_{\min} + 2 \bmod 4)$ with input 0. So, $d_{v_{\min}} = 1$ and $d_{v_{\min} \bmod 2} = 0$. From $d_{v_{\min}} = 1$ and $v_{\min} \in \{v_i - 1, v_i\}$ it follows that $v_{\min} \in V_i$. Therefore $r_i \geq v_{\min}$. From $d_{v_{\min}+2 \bmod 4} = 0$ it then follows that $v_{\min} \leq r_i \leq v_{\min} + 1$. Assume that no party has $r_i = v_{\min} + 1$. Then $r_i = v_{\min}$ for all parties and agreement and validity follows. Assume then that some party has $r_i = v_{\min} + 1$. This means that $d_{v_{\min}+1} = 1$. Since all parties have $v_i \in \{v_{\min}, v_{\min} + 1\}$ it follows that $v_{\min} + 1 \in V_i$, so that $r_i \geq v_{\min} + 1$. Together with $r_i \leq v_{\min} + 1$ it then follows that $r_i = v_{\min} + 1$ for all honest P_i. From this agreement follows. Validity follows from the observation that if no honest party has $v_i = v_{\min} + 1$, then all honest parties have $v_i = v_{\min}$ and the result will trivially be $r_i = v_{\min}$. □

7.8.3 The MBA Protocol

In this section we present a protocol realizing $\mathcal{F}_{\text{MBA}}$, see Fig. 7.11 on the following page. It is similar to, and inspired by, the MULTI-CONSENSUS protocol in [BDGK95]. The changes made are to allow it to run also for $t = \lfloor (n-1)/2 \rfloor$, where the protocol in [BDGK95] assumes that $t = \lfloor (n-1)/3 \rfloor$.

Theorem 7.13 *π_{MBA} realizes $\mathcal{F}_{\text{MBA}}$ in the $(\mathcal{F}_{\text{SPHAKIBA}}, \mathcal{F}_{\text{SGR}})$-hybrid model and the round complexity is at most $4A + 2f$ phases in $\mathcal{F}_{\text{SPHAKIBA}}$, where A is the number of activating rounds and f is the number of corrupted parties.*

Proof. Following the rational given above for the protocol it is straight-forward to verify that it actually realizes $\mathcal{F}_{\text{MBA}}$. We verify that the round complexity is as claimed.

Protocol π_{MBA}

The protocol runs with parties $P_1, \ldots, P_n$ in the $(\mathcal{F}_{\text{SPHAKIBA}}, \mathcal{F}_{\text{SGR}})$-hybrid model, and we use $\mathcal{F}_{\text{SPHAKIUBC}}$ to denote a sub-protocol using four calls to $\mathcal{F}_{\text{SPHAKIBA}}$ for solving the UBC problem, see the proof of Theorem 7.12 on the page before for details. The party P_i proceeds as follows:

initializing:

On the first message:

1. Let $\texttt{LastKing}(-1) \leftarrow 1$ and $\texttt{LastKing}(0) \leftarrow 1$.
2. Let $\texttt{activation} \leftarrow 0$.

activation:

On input $\{(baid, b_{baid})\}$ let $\texttt{Running} \leftarrow \{(baid, b_{baid})\}$, let $\texttt{Decided} \leftarrow \emptyset$ and proceed as follows:

1. $\texttt{activation} \leftarrow \texttt{activation} + 1$.
2. $\texttt{round} \leftarrow -\texttt{init-len}$.
3. Let $\texttt{king}(\texttt{activation}) \leftarrow \texttt{LastKing}(\texttt{activation} - 2) + 1 \bmod n$.
4. Input $(\texttt{activation}, \texttt{king}(\texttt{activation} - 1))$ to $\mathcal{F}_{\text{SPHAKIUBC}}$ and for $(baid, b_i) \in \texttt{Running}$, input $(baid, b_i)$ to $\mathcal{F}_{\text{SPHAKIBA}}$.
5. Until $\mathcal{F}_{\text{SPHAKIUBC}}$ has output some $(\texttt{decide}, \texttt{activation}, K)$ and $|\texttt{Running}| = \emptyset$, do:
   (a) If $\texttt{round} \geq 0$ and $(\texttt{round} \bmod \texttt{phase-len}) = \texttt{king-round}$, then input $(\texttt{king}(\texttt{activation}) \bmod n)$ to $\mathcal{F}_{\text{SPHAKIUBC}}$ and $\mathcal{F}_{\text{SPHAKIBA}}$ and let $\texttt{king}(\texttt{activation}) \leftarrow \texttt{king}(\texttt{activation}) + 1$.
   (b) Input $(\texttt{do round}, baid)$ to $\mathcal{F}_{\text{SPHAKIBA}}$ for $(baid, \cdot) \in \texttt{Running}$ and input $(\texttt{do round}, \texttt{activation})$ to $\mathcal{F}_{\text{SPHAKIUBC}}$.
   If $\mathcal{F}_{\text{SPHAKIBA}}$ outputs $(\texttt{decide}, baid, b)$ for $(baid, \cdot) \in \texttt{Running}$, then let $\texttt{Decided} \leftarrow \texttt{Decided} \cup \{(baid, b)\}$ and remove $(baid, \cdot)$ from $\texttt{Running}$.
6. Let $\texttt{LastKing}(\texttt{activation} - 1) \leftarrow K$.
7. Input $(\texttt{activation}, \texttt{phase-len})$ to $\mathcal{F}_{\text{SGR}}$.
8. When $\mathcal{F}_{\text{SGR}}$ outputs $(\texttt{activation}, \texttt{term})$, then output $(baid, \texttt{decide}, b)$ for $(baid, b) \in \texttt{Decided}$.

Figure 7.11: The MBA Protocol.

We consider one of the two ongoing round-robin schedules. Let A' be the number of activations run with this schedule. Observe first that because of the one-phase-staggering of the special phase-king functionality $\mathcal{F}_{\text{SPHAKIBA}}$ and the fact that one-phase-staggering composes in parallel, the parties will input consecutive values to the UBC protocol, namely the king of the round in which they terminated. Notice that the value `king` is not reduced modulo n until after it was input to the UBC. This is to make sure that the values are indeed consecutive. By the definition of UBC it then follows that the output K of the UBC is the index of a party which was king in one of the two consecutive rounds in which parties

terminated two activations earlier.

If the parties terminated with a staggering, then call the party which was king in the last phase where parties terminated the last last king and call the previous king the first last king. Notice that there are at most one last last king per activation. Call a king which is king in a phase where there is a BA in which all parties are active an influential king. Notice that because of the one-phase staggering, a king which is not influential is a last last king. We will be interested in the sequence of influential kings, and want to argue that it is a round robin except for a few omissions called skipped last last kings.

Consider again the output K of the UBC. Assume first that it is the index of the first last king, including the case where all parties terminated the same round. Because the parties start from $K+1 \bmod n$ this means that the last last king start as king in the next activation. This means that the round-robin of influential kings is kept. If K is the index of the last last king, then the last last king is skipped in the round-robin of influential kings. We call such a king a skipped last last king.

Assume that r phases of $\mathcal{F}_{\text{SPHAKIBA}}$ is executed. If we look at the sequence of influential kings in these phases and insert the skipped last last kings, then it is of the form $1, 2, \ldots, n, 1, 2, \ldots, n, 1, \ldots$. Therefore, out of the total of r of these kings, at most $(r-f)/2 + f$ were corrupted, so at least $\frac{1}{2}r - \frac{1}{2}f$ were honest. Of these at most A' were skipped last last kings. The remaining $\frac{1}{2}r - \frac{1}{2}f - A'$ were influential honest kings. Because of the decision-after-consensus-on-good king, each of these terminated a distinct activation. Therefore $\frac{1}{2}r - \frac{1}{2}f - A' \leq A'$. This implies that $r \leq 4A' + f$. We have that $A'' = A - A'$ activations were run with the other round-robin schedule, giving rise to at most another $4A'' + f$ phases. A total of $4A + 2f$ phases, as claimed. □

7.8.4 Randomized Mode

Even though the above realization of the $\mathcal{F}_{\text{MBA}}$ protocol, when $f = O(A)$, has a constant amortized round-complexity, some activations might take $\Theta(f+2)$ rounds. In the worst case, this is close to optimal, but it might in some cases be desirable to have, in addition, a better *expected* round-complexity, especially because we consider reactive functionalities. This can be obtained by using a coin-flip for choosing the king in some phases, every second phase say. Running in the $\mathcal{F}_{\text{coin}}$-hybrid model the parties can generate the coin by all inputting `flip`. Because of the special property of the phase-king protocols that they terminate if the elected king was honest in the round *before* the round where the king is input — the round called king commit in Fig. 7.9 on page 240 and Fig. 7.10 on page 243 — it follows that the probability that the protocol terminates in a phase with a random king is at least $\frac{n-f}{n} > \frac{1}{2}$. Therefore the expected round complexity will indeed be constant.

For the authenticated model the coin-flip protocol from Section 6.6.3 can be used. Using this protocol, the round complexity of one coin-flip is constant, and the communication complexity is $O(n^2 k)$ bits. For the unauthenticated model the protocol from [FM97] can be used. Using this protocol, the round complexity of one coin-flip is constant, and the communication complexity is $O(n^4)$ bits.

Note however that if in π_{MBA} the parties input `flip` to $\mathcal{F}_{\text{coin}}$ in each phase, including those where no BA is active, then we cannot maintain the worst-case complexities — all

rounds will be active even when no BA is active. The obvious solution is to let the parties only input `flip` when they are active in a BA, and then continue where they left of when a BA is activated again. However, the parties must take care to input `flip` the same number of times to see the same coin. A simple solution is to use the multi-session functionality. In activation `activation` the parties use input (`flip`, `activation`). Since this gives session id's which is a sequence, the initialization and the coin-flip will both be constant round, see Section 6.6.3.

7.9 A Trivial Lower Bound on the Round Complexity of an MBA Protocol

Theorem 7.14 *Any MBA protocol for the unauthenticated or the authenticated model which tolerates t faults, will for all $f < t$ and all A have executions with A activating rounds and f faults which has* $\min(2A + f, 2A + t - 1)$ *active rounds.*

Proof. Consider the following adversarial strategy: Run the first $A-1$ BAs non-overlapping and without corrupting any parties ($f = 0$), but on carefully chosen inputs and random bits and then use full force in the last BA. By the results in [DRS90] we know that the adversary can pick a set of inputs and random bits so that the protocol runs for at least $\min(f + 2, t + 1) = 2$ active rounds in the first BA. Now save the state of all parties, denote the state by $(P_1^1, \ldots, P_1^n)$. We have that $(P_1^1, \ldots, P_1^n)$ defines a BA protocol: On any input party i will simply run from the saved state P_1^i. Therefore, by the results in [DRS90], we know that the adversary can pick a set of inputs and random bits so that this protocol runs for at least $\min(f + 2, t + 1) = 2$ active rounds. In particular the MBA protocol must in some cases use 4 rounds for the first 2 BAs. Continue this way to see that the first $A - 1$ BAs must use $2(A - 1)$ active rounds. Now use the same argument on the protocol defined by the parties' state after $A - 1$ BAs and get a last term $\min(f + 2, t + 1)$, and then add. □

Part III

Secure Multiparty Computation

CHAPTER 8

General Multiparty Computation, Static Security

The less secure a man is, the more likely he is to have extreme prejudice.
— Clint Eastwood

8.1 Introduction

In this chapter we present a solution to the so-called multiparty computation problem. The problem of multiparty computation (MPC) dates back to the papers by Yao [Yao82a] and Goldreich, Micali and Wigderson [GMW87]. What was proved there was basically that a collection of n players can efficiently compute the value of any n-input function, such that everyone learns the correct result, but no other new information. This is known as secure function evaluation. These protocols are proved secure against static PPT adversaries which corrupts a set of less than $n/2$ players. The protocols are proved secure in a weaker model of security than that in Chapter 3. In particular, they are not UC secure. Later, unconditionally secure MPC protocols were proposed by Ben-Or, Goldwasser and Wigderson and Chaum, Crépeau and Damgård [BGW88, CCD88] for the information theoretic model where private channels are assumed between every pair of players. The solution that we are going to present is for the cryptographic model.

Over the years, several protocols have been proposed which, under specific computational assumptions, improve the efficiency of general MPC, see for instance [BB89,CDM00,GRR98, CDD00]. Virtually all proposals have been based on some form of verifiable secret sharing (VSS), i.e., a protocol allowing a dealer to securely distribute a secret value s among the players as described in Section 6.3.1. But in the malicious case, where both the dealer and/or some of the players may be cheating. The protocol must then guarantee that after the sharing protocol terminates, a unique secret is *defined* and can be *reconstructed*. Then the parties manipulate these shared inputs to compute some function of them. The basic paradigm that has been used is to ensure that all inputs and intermediate values in the computation are VSS'ed. The idea of using VSS for computation has received an immense

attention, some of it mentioned above, but will not be covered further. See e.g. [GM95] as a starting point.

In all these earlier protocols, the total number of bits broadcast was $\Omega(n^2k|C|)$, where n is the number of players, k is a security parameter, and $|C|$ is the size of a circuit computing the desired function. Here, C may be a Boolean circuit, or an arithmetic circuit over a finite field, depending on the protocol. We note that all complexities mentioned here and in the next section are for computing deterministic functions. Handling probabilistic functions introduces some overhead for generating secure random bits, but this will be the same for all protocols we mention here, and so does not affect any comparisons we make.

In [FH96] Franklin and Haber propose a protocol for *passive* adversaries which achieves complexity $O(nk|C|)$. This protocol is not based on VSS (there is no need since the adversary is passive) but instead on a so-called joint encryption scheme, where a ciphertext can only be decrypted with the help of all players, but still the length of an encryption is independent of the number of players.

In this chapter we show how this new approach can be used to building multiparty computation protocols with *active* security. We start from any secure threshold encryption scheme with certain extra homomorphic properties. Input values are then encrypted under this scheme and the computation proceeds by manipulation of the ciphertexts. This allows us to avoid the need to VSS all values handled in the computation, and therefore leads to more efficient protocols, as detailed below. In [CDN01], some of the results presented in this chapter were presented in preliminary form.

Also Canetti and Gennaro [CG96] consider an approach to MPC where inputs are encrypted under a public key with the corresponding secret key shared among to parties. They do however not consider homomorphic schemes, but carry out the computation on ciphertexts using a general MPC. As a consequence their protocol is very inefficient compared to ours. However, their goal is not efficiency, but to demonstrate feasibility of so-called incoercible MPC.

The MPC protocols we construct here can be proved secure against an active and static adversary who corrupts any minority of the players. Like the protocol of [FH96], our construction requires once and for all an initial phase where keys for the threshold cryptosystem are set up. This can be done by a trusted party, or by any general purpose MPC. We stress, however, that unlike some earlier proposals for preprocessing in MPC, the complexity of this phase does not depend on the number or the size of computations to be done later. It is even possible to do a computation only for some subset of the players that participated in the first phase, provided the subset is large enough compared to the threshold that the cryptosystem was set up to handle. Moreover, since supplying input values to the computation consists essentially of just sending encryptions of these values, we can easily handle scenarios where one (large) group of players supply inputs, whereas a different (smaller) group of players does the actual computation.

In the following we therefore focus on the complexity of the actual computation. In our protocol the computation can be done only by broadcasting a number of messages, no encryption is needed to set up private channels. Most complexities we give in the following are therefore simply the number of bits broadcast, we refer to this as broadcast complexity. This does not invalidate comparison with earlier protocols because first, the same measure

was used in [FH96] and second, the earlier protocols with active security have complexity quadratic in n even if one only counts the bits broadcast. Our protocol has broadcast complexity $O(nkM + kI)$ bits, where M is the number of multiplication gates in the circuit and I is the number of input gates in the circuit. The protocol requires $O(d)$ rounds, where d is the multiplicative depth of C. To the best of our knowledge, this is the most efficient general MPC protocol tolerating up to $n/2$ active cheaters proposed to date.

Here, C is an arithmetic circuit over a ring R determined by the cryptosystem used, e.g., $R = \mathbf{Z}_n$ for an RSA modulus n, or $R = \mathrm{GF}(2^k)$. While such circuits can simulate any Boolean circuit with a small constant factor overhead, this also opens the possibility of building an ad-hoc circuit over R for the desired function, possibly exploiting the fact that with a large R, we can manipulate many bits in one arithmetic operation. In [LP01] Lysyanskaya and Peikert used this property of our protocol to construct an efficient adaptively secure threshold signature scheme, by replacing a previously more inefficient step in a known threshold signature scheme by our protocol.

The complexities given here assume existence of sufficiently efficient threshold cryptosystems. We give two examples of such systems with the right properties. One is based on Paillier's cryptosystem [Pai99], the other one is an extension of Franklin and Haber's cryptosystem [FH96]. It is secure assuming that both the QRA and the strong RSA assumption are true. Essentially the same scheme was also presented in [CDN01], but proved secure based on the QRA and the DDH assumption. This is essentially the same as the assumption made in [FH96]. While the first example is known (from [DJ01] and independently in [FPS00]), the second is new and may be of independent interest. In [KY02] Katz and Yung have presented a threshold homomorphic encryption scheme, which also supports our construction. Their scheme too can be based on the QRA assumption and the strong RSA assumption.

Franklin and Haber [FH96] left as an open problem to study the communication requirements for active adversaries. With the results presented in this chapter we can now say that under the same assumption as theirs, protection against active adversaries comes essentially for free. In Chapter 10 we then show how the protocol can be made adaptively secure. The main reason for this separation is that a complete proof of security is challenging enough without considering an adaptive adversary and will be used to build some intuition for the security of the approach. In Chapter 9 we then introduce an adaptively secure universally composable commitment scheme which will be an essential tool in making the protocol in this chapter adaptively secure.

We prove our protocols secure in the UC framework from Chapter 3. The framework allows to consider a more general problem than the secure function evaluation problem, as it allows to define the security of general reactive tasks. This allows us to prove that our protocol is equivalent to what we call an arithmetic black box (ABB). An ABB can be thought of as a secure general-purpose computer. Every player can in private specify inputs to the ABB, and any majority of players can ask it to perform any feasible computational task and make the result (and only the result) public. Moreover the ABB can be invoked several times and keeps an internal state between invocations. This point of view allows for easier and more modular proofs, and also makes it easier to use our protocols as tools in other constructions.

8.1.1 Informal Description of the Main Ideas

In this section, we give a completely informal introduction to some main ideas. All the concepts introduced here will be treated more formally in subsequent sections. We assume that from the start the following scenario has been established: We have an IND-CPA secure threshold public-key system given, i.e., there is a public encryption key pk known by all players, while the matching private decryption key has been shared among the players using e.g. a simple function sharing scheme (Definition 6.2 on page 194) for the decryption function.

The message space of the cryptosystem is assumed to be a ring R. In practice R might be $\mathbf{Z}_n$ for some RSA modulus n. For a plaintext $a \in R$, we let $\overline{a}$ denote an encryption of a. We then require certain homomorphic properties: From encryptions $\overline{a}, \overline{b}$, anyone can easily compute (deterministically) an encryption of $a+b$, which we denote by $\overline{a} \boxplus \overline{b}$. We also require that from an encryption $\overline{a}$ and a constant $\alpha \in R$, it is easy to compute an encryption of αa, which we denote by $\alpha \boxdot \overline{a}$. This immediately gives us an algorithm $\boxminus$ for subtracting.

Finally we assume that three secure (and sufficiently efficient) sub-protocols are available:

proving you know a plaintext:
: If P_i has created an encryption $\overline{a}$, he can give a zero-knowledge proof of knowledge that he knows a (or more accurately, that he knows a and a witness to the fact that the plaintext is a).

proving multiplications correct:
: Assume P_i is given an encryption $\overline{a}$, chooses a constant α, computes a random encryption $\overline{\alpha a}$ and broadcasts $\overline{\alpha}, \overline{\alpha a}$. He can then give a zero-knowledge proof that indeed $\overline{\alpha a}$ contains the product of the values contained in $\overline{\alpha}$ and $\overline{a}$.

threshold decryption:
: For the third sub-protocol, we have common input pk and an encryption $\overline{a}$. In addition every player also uses his share of the private key as input. The protocol computes securely a as output for everyone.

We can then sketch how to perform securely a computation specified as a circuit doing additions and multiplications in R. Note that this allows us to simulate a Boolean circuit in a straightforward way using 0/1 values in R.

The MPC protocol would simply start by having each player publish encryptions of his input values and give zero-knowledge proofs that he knows these values and also that the values are 0 or 1 if we are simulating a Boolean circuit.[1] Then any operation involving addition or multiplication by constants can be performed with no interaction: If all players know encryptions $\overline{a}, \overline{b}$ of input values to an addition gate, all players can immediately compute an encryption of the output $\overline{a+b}$. This leaves only the following problem:

Given encryptions $\overline{a}, \overline{b}$ (where it may be the case that no players knows a nor b), compute securely an encryption of $c = ab$. This can be done by the following protocol (which is a slightly optimized version of the protocol from [CDN01]):

[1] The purpose of having each party prove that he *knows* the plaintext is to prevent corrupted parties from copying the ciphertext of a honest party and thereby inputting the same value. This will be discussed i more detail in the technical section.

1. Each player P_i chooses at random a value $d_i \in R$ and broadcasts encryptions $\overline{d_i}$ and $\overline{d_i b}$.

2. All players prove, using the first and second sub-protocol, that they know their respective values of d_i, and that $\overline{d_i b}$ encrypts the correct value. Let N be the subset of the parties succeeding both proofs.

3. All parties can now compute $\overline{a} \boxplus (\boxplus_{i \in N} \overline{d}_i)$. This ciphertext is decrypted using the third sub-protocol, so all players learn $a + \sum_{i \in N} d_i$.

4. All parties set $\overline{c} = (a + \sum_{i \in N} d_i) \boxdot \overline{b} \boxminus (\boxplus_{i \in N} \overline{d_i b})$.

At the final stage we know encryptions of the output values, which we can just decrypt. Intuitively this is secure if the encryption is secure because, other than the outputs, only random values and values already known to the adversary are ever decrypted. We give proofs of this intuition in the following.

8.1.2 Related Work

In [SYY99] Sander Young and Yung also use homomorphic encryption for secure computation. What is considered there is the so-called crypto-computing, where an input party sends its encrypted inputs to a crypto-computing server which evaluates a circuit on the encrypted inputs and its own local inputs and deliver an encrypted version of the output back to the input party, which can decrypt to learn the output. In doing this the server learns nothing about the encrypted input of the input party. In [SYY99] a round optimal protocol for doing this (two moves) is presented for the class of circuits NC^1. Some of the work on using homomorphic encryption for crypto-computing is surveyed in [SYY99].

In concurrent independent work, Jacobson and Juels [JJ00] use an idea somewhat related to ours, the so-called mix-and-match approach which is also based on threshold encryption (with extra algebraic properties, similar to, but different from the ones we use). Beyond this, the techniques are completely different. For Boolean circuits and in the random oracle model, they get the same broadcast complexity as we obtain here (without using random oracles). The round complexity is larger than ours (namely $O(n + d)$). Another difference is that mix-and-match is inherently limited to circuits where all gates can be specified by constant size truth-tables. This introduces a considerable overhead when e.g. evaluating an arithmetic circuit over a large ring.

8.2 The Arithmetic Black-Box

In this section we define the arithmetic black-box (ABB). We think of the arithmetic black-box as a tamper resistant computer to which each party has a secure terminal. The computer allows parties to load their private values into variables and manipulate them using commands like $(x \leftarrow x_1 + x_2)$ to add variables and $(\texttt{output}, x)$ to reveal the value of a variable to all parties. The machine will not execute any command unless all honest parties have instructed it to execute that command. We then realize the ABB in the following sections.

This approach to studying secure general MPC has been chosen for several reasons. First of all it possesses several advantages over the function evaluation approach which has been the typical one applied when studying general secure MPC. Most importantly, it allows for more modular proofs and the ABB functionality is easier to use as part of another protocol, mostly due to the fact that the ABB functionality is stateful, whereas a function evaluation functionality is stateless. The ABB allows to do some computations on secret variables, reveal some partial knowledge about the variables and then continue to manipulate the secret variables based on this revealed information. The ABB does not force you to specify the sub-tasks to be handled by the functionality as functions. A similar approach is taken by Canetti, Lindell, Ostrovsky and Sahai in [CLOS02], where it is showed how to realize *any* well-formed functionality. Compared to our approach this is more general, as we realize a *specific* functionality. However, our functionality is easily seen to be universal — any functionality can be formulated using addition and multiplication over a ring and then evaluated via $\mathcal{F}_{\text{ABB}}$. Our reason for choosing the specific functionality $\mathcal{F}_{\text{ABB}}$ is that it generalizes the formulation of threshold homomorphic encryption scheme in a natural manner and therefore is closely coupled to the way we are going to realize to functionality. This simplifies the proofs considerable.

8.2.1 Specification of the ABB

Formally, the ABB is an ideal functionality. On initialization with security parameter k the ABB generates a uniformly random public key pk. The key defines the ring R_{pk} that the ABB does arithmetic over. In each activation it expects a command from each party and carries out the command if all honest parties agreed on the command to be carried out. Typically agreement means that all honest parties gave the same command, e.g. $(x \leftarrow x_1 + x_2)$, but this does not always make sense. If e.g. party P_i wants to load a secret value s using the command $(P_i, x \leftarrow s)$, then of course the other parties cannot acknowledge this by giving the same command. In this case they acknowledge by giving the command $(P_i, x \leftarrow S)$, where S is (a description of) some subset of R_{pk}. The intended meaning is that P_i is allowed to define x using a secret value from S.

Definition 8.1 *A PPT family of rings is a PPT algorithm gen, which on input k outputs $pk \in \{0,1\}^*$, where pk is a description of a ring $(R_{pk}, +_{pk}, \cdot_{pk})$. We require that we can do the following in PPT given $pk \in gen$: 1) Sample a uniformly random element from R_{pk}, 2) Compute strings representing 0_{pk} and 1_{pk}, 3) Compute $a +_{pk} b$ and $a \cdot_{pk} b$ given $a, b \in R_{pk}$, 4) Given $a \in \{0,1\}^*$, compute whether $a \in R_{pk}$, 5) Compute $a^{-1} \in R_{pk}^*$ for $a \in R_{pk}^*$. The arithmetic black-box (ABB) over a family of rings is the ideal functionality $\mathcal{F}_{\text{ABB}}$ given in Fig. 8.1 on the next page. The requirements in incorrect inputs defines an IO behavior $\mathcal{IO}_{\text{ABB}}$. By the basic arithmetic black-box (BABB) we mean the ideal functionality $\mathcal{F}_{\text{BABB}}$ which is $\mathcal{F}_{\text{ABB}}$ without the multiplication command.*

We give some clarifying remarks on the formulation of $\mathcal{F}_{\text{ABB}}$.

Remark 8.1 *The value r which is output on the SOT in initialization basically models that it is not a part of the functionality to be a secure distributed evaluation of gen. The value*

Functionality $\mathcal{F}_{\text{ABB}}^{gen}$

The functionality $\mathcal{F}_{\text{ABB}}^{gen}$ runs with parties $P_1, \ldots, P_n$ and is parameterized by a family of rings *gen*. All inputs to the parties are parsed in one of the forms `init` or $(cid, \cdots)$ for $cid \in \{0,1\}^*$. All input values are output on the SOT, except for s in load.

initialization:

If all honest parties input `init` in the same round, then generate a random key $pk \leftarrow gen(k; r)$ and output pk to all parties and output r to the adversary. Let val be a dictionary mapping from variable names $x \in \{0,1\}^*$ into $R_{pk} \cup \{\perp\}$. Initially $\text{val}(x) = \perp$ for all $x \in \{0,1\}^*$.

Output `ready` to all parties in a round specified by the adversary, and ignore all inputs until this has been done.

load:

On input $(cid, P_i, x \leftarrow S)$, possibly $S = R_{pk}$, in the same round from all honest parties except P_i, if P_i inputs $(cid, P_i, x \leftarrow s, S)$ for $s \in S$ in the same round or P_i is corrupted, then:

- If in some round $(cid, \texttt{term}, s')$ is input on the SIT for $s' \in S$, then proceed as follows: If P_i is honest in that round, then set $\text{val}(x) = s$. Otherwise, set $\text{val}(x) = s'$. Then output $(cid, \texttt{defined})$ to all parties
- If in some round P_i is corrupted and $(cid, \texttt{fail})$ is input on the SIT, then output $(cid, \texttt{fail})$ to all parties.

And then ignore all subsequent $(cid, \cdot)$ inputs on the SIT.

linear combination:

On input $(cid, x \leftarrow a_0 + \sum_{j=1}^{l} a_j x_j)$ in the same round from all honest parties, if $a_j \in R_{pk}$ for $j = 0, \ldots, l$ and $\text{val}(x_j) \neq \perp$ for $j = 1, \ldots, l$, then define $\text{val}(x) = a_0 + \sum_{j=1}^{l} a_j \, \text{val}(x_j)$ and output $(cid, \texttt{defined})$ to all parties.

Figure 8.1(a) (cont. in Fig. 8.1(b) on the next page): The Arithmetic Black-Box

pk is only output to specify the ring to do arithmetic over. Furthermore, outputting r in the initialization *command guarantees that $\mathcal{F}_{\text{ABB}}$ is a well-formed functionality. It might seem puzzling that the values pk and r occur as part of the initialization of the functionality $\mathcal{F}_{\text{ABB}}$, as these values are part of the* realization *of the functionality. This might in particular seem puzzling for the value r, which in the realization might specify also the secret key sk corresponding to pk (and in our realizations sk is not even known by the honest parties). The point, however, is that the ring that $\mathcal{F}_{\text{ABB}}$ computes over is specified by pk. In the case of Paillier e.g. the value N must be output to the parties using $\mathcal{F}_{\text{ABB}}$ to tell them that input values should be from $\mathbf{Z}_N$, that output values are from $\mathbf{Z}_N$ and that additions and multiplications are in $\mathbf{Z}_N$. If the plaintext ring is independent of the public key (as is the case when it is $\mathbf{Z}_2$) then it is true that pk and r can be dropped in the initialization. We will however not discuss this, simpler, case further. This discussion only argues why pk is part of the initialization. Why then is r (and thereby possibly sk) given to the adversary $\mathcal{S}$ on the SOT? If Paillier encryption is used r would e.g. contain the factorization of N.*

Functionality $\mathcal{F}_{\text{ABB}}^{gen}$

multiplication:

On input $(cid, x \leftarrow x_1 \cdot x_2)$ in the same round from all honest parties, if $\text{val}(x_1) \neq \perp$ and $\text{val}(x_2) \neq \perp$, then define $\text{val}(x) = \text{val}(x_1) \cdot \text{val}(x_2)$ and output $(cid, \texttt{defined})$ to all parties in a round specified by the adversary.

private multiplication:

On input $(cid, P_i, x \leftarrow x_1 \cdot x_2)$ in the same round from all honest parties, if P_i loaded x_1, i.e the command $(cid', P_i, x \leftarrow S)$ was executed at some point, and $\text{val}(x_2) \neq \perp$, then:

- If in some round $(cid, \texttt{term})$ is input on the SIT, then set $\text{val}(x) = \text{val}(x_1) \cdot \text{val}(x_2)$ and output $(cid, \texttt{defined})$ to all parties.
- If in some round P_i is corrupted and $(cid, \texttt{fail})$ is input on the SIT, then output $(cid, \texttt{fail})$ to all parties.

And then ignore all subsequent $(cid, \cdot)$ inputs on the SIT.

output:

On input $(cid, \texttt{output}, x)$ in the same round from all honest parties, if $\text{val}(x) \neq \perp$, then output $(cid, \texttt{output}, x, \text{val}(x))$ on the SOT. Then output $(cid, \text{val}(x))$ to all parties in a round determined by the adversary.

aligned output:

If more than one commands is activated in the same round, then the outputs are delivered when all the commands have terminated.

incorrect inputs:

If the first non-trivial input from all honest parties is not `init` or two honest parties input `init` in different rounds or any honest party inputs `init` twice, then break down. If in some round any honest party inputs a value of the form $(cid, \ldots)$ and some honest party does not, then break down. If any honest party uses the same command id *cid* twice, then break down. This allows to divide the inputs in all rounds into commands, where each honest party contributed a value to each command. If any of these commands cannot be parsed on one of the above forms, then break down. If a variable name is defined by several commands, then break down. Finally we require non-overlapping IO behavior, meaning that the next round of commands is not delivered until the last round of commands terminated, see Definition 3.28 on page 144 for a formalization.

Figure 8.1(b) (cont. from Fig. 8.1(a) on the preceding page): The Arithmetic Black-Box

The intuitive reason is that this renders pk 'insecure' outside the functionality when used in a $\mathcal{F}_{\text{ABB}}$-hybrid model: Since sk is allowed to leak to the environment, the parties cannot both use $\mathcal{F}_{\text{ABB}}$ and in addition use pk for doing, says, encryptions or commitments outside $\mathcal{F}_{\text{ABB}}$.[2] *It is necessary to prevent this as such encryptions or commitments could be replayed in the* realization *of $\mathcal{F}_{\text{ABB}}$ when the functionality is replaced by a realization. This could make the outer protocol and the realization of $\mathcal{F}_{\text{ABB}}$ interact in an adverse way, so that in particular it would not be secure to use any realization in any context. Since UC security*

[2] At least not if these encryptions or commitments depend on *sk* being hidden.

guarantees that it is indeed so, it is actually the strong UC security notion which forces us to reveal r in the hybrid model and render pk useless outside $\mathcal{F}_{\text{ABB}}$. The way this intuitive reason pops up in the technicalities of the proof is through the fact that if we do not reveal r on the SOT, then in the ideal process, where the simulator is given the output on the SOT, the simulator cannot simulate, see Footnote 8 on page 275

Remark 8.2 *An alternative formulation of the $\mathcal{F}_{\text{ABB}}$ functionality might let the adversary $\mathcal{S}$ choose pk and/or r to model that the functionality might specify any key pk and thereby any ring. Our realization would still be secure for this formulation, but the adversary $\mathcal{S}$ (being the simulator in the proof of security) would in all cases just pick pk at random anyway to get semantic security. Therefore nothing would be obtained by changing the formulation. On the other hand, when using $\mathcal{F}_{\text{ABB}}$ in the $\mathcal{F}_{\text{ABB}}$-hybrid model the parties would receive an adversely chosen pk instead of a random one. Therefore something might conceivably be lost. Choosing between a case where by guarantee nothing is lost and one where something* might *be lost we pick the first.*

Remark 8.3 *The* load *commands appears to be slightly more involved than one would expect. There are however good reasons for all the complications. The functionality says that an honest party must know from the beginning of the protocol what its input is, but that a corrupted party need not be bound to its input before the end of the load protocol and can of course start the protocol honestly and then abort. However, if the* load *command terminates, then honest parties will learn whether or not* $\text{val}(x)$ *was defined and if they receive* $(cid, \texttt{defined})$*, then P_i is committed to* $\text{val}(x)$*. The reason that honest parties must know the value at the beginning of the protocol is of course that the value is needed in the protocol realizing the command. The reason why we allow that that the corrupted parties are not committed until after the command is that it allows more, and more efficient, realizations.*

Remark 8.4 *The* private multiplication *command allows a party that loaded x_1 to multiply it with some other variable. The reason for having both* private multiplication *and* multiplication *is that* private multiplication *can be realized more efficiently than* multiplication *and that we will realize the* multiplication *command via* private multiplication *commands.*

8.2.2 General Multiplication

We later give several realizations of $\mathcal{F}_{\text{ABB}}$, but first we prove that the multiplication command can be reduced to the other commands in a black-box manner, i.e. $\mathcal{F}_{\text{ABB}}$ can be realized in the $\mathcal{F}_{\text{BABB}}$-hybrid model. This then leaves us with the problem of realizing $\mathcal{F}_{\text{BABB}}$ for the rings in question. The protocol is given in Fig. 8.2 on page 264.

Theorem 8.1 *π_{ABB} n-realizes $\mathcal{F}_{\text{ABB}}$ in the $\mathcal{F}_{\text{BABB}}$-hybrid model.*

Proof. That π_{ABB} uses $\mathcal{F}_{\text{BABB}}$ according to $\mathcal{IO}_{\text{ABB}}$ under $\mathcal{IO}_{\text{ABB}}$ follows by inspection. To prove the theorem it is therefore enough to prove that there exists a hybrid interface $\mathcal{T}$ such that $\text{IDEAL}^{\mathcal{F}_{\text{BABB}}}_{\mathcal{F}_{\text{ABB}},\mathcal{T},\mathcal{Z}_{|\mathcal{IO}_{\text{ABB}}}} = \text{HYB}^{\mathcal{F}_{\text{BABB}}}_{\pi_{\text{ABB}},\mathcal{Z}_{|\mathcal{IO}_{\text{ABB}}}}$ for all environments $\mathcal{Z}$.[3]

[3]Since $\mathcal{IO}_{\text{ABB}}$ is PPT we can consider the self-restricting environment $\mathcal{Z}_{|\mathcal{IO}_{\text{ABB}}}$.

We sketch how $\mathcal{T}$ can simulate the values seen during the protocol given access to the values on the SOT of $\mathcal{F}_{\text{ABB}}$. When the command given by $\mathcal{Z}$ is not the multiplication command the simulation is trivial. But, for the multiplication protocol $\mathcal{F}_{\text{ABB}}$ basically just tells us that a multiplication with command id *cid* took place, whereas what we have to simulate is the values of the multiplication protocol.

Observe that the only value which is not trivial to simulate is the value v output in Step 4 (it is output to the parties in Step 5, but should be simulated as output on the SOT in Step 4, when the `output`-command is given), all other values can be computed given just *cid*. If no parties are honest, there is nothing to simulate. Assume then that at least one party P_j is honest. Let A denote $\sum_{j\in J_2} a_j$, so that $v = \text{val}(x_1) + A$. We simulate $\text{val}(x_1) + A$ by giving a uniformly random value $v \in R_{pk}$ to $\mathcal{Z}$. We argue that this is a perfect simulation: Clearly $j \in J_2$, so in $\text{HYB}^{\mathcal{F}_{\text{BABB}}}_{\pi_{\text{ABB}},\mathcal{Z}_{|\mathcal{IO}_{\text{ABB}}}}$ we would have that the value A is uniformly random and independent of the view of the environment. This is so because the environment learns no information about a_j when it is loaded, so the a_i values loaded by the environment on behalf of the corrupted parties are independent of a_j. Since addition by an element in a ring is a bijection, the claim follows. So, if just one party is honest, then in $\text{HYB}^{\mathcal{F}_{\text{BABB}}}_{\pi_{\text{ABB}},\mathcal{Z}_{|\mathcal{IO}_{\text{ABB}}}}$ the value $v = \text{val}(x_1) + A$ is a uniformly random value in the view of the environment. If a honest party P_i is later corrupted, then simply simulate the internal a_i value of P_i honestly, using a uniformly random element from R_{pk}.[4] When the *last* honest party P_j is corrupted, we learn the true value of $\text{val}(x_1)$ from $\mathcal{F}_{\text{BABB}}$[5] and can simulate the a_j value by $a_j = v - \text{val}(x_2) - \sum_{i\in J_2\setminus\{j\}} a_i$. Since v was chosen as a uniformly random value it follows that a_j is a uniformly random value for which $v = \text{val}(x_2) + \sum_{j\in J_2} a_j$, as desired.

We have argued that we can simulate the view of the communication of protocol perfectly. What remains is to argue that the output $\mathcal{Z}$ sees from $\mathcal{F}_{\text{BABB}}$ in $\text{HYB}^{\mathcal{F}_{\text{BABB}}}_{\pi_{\text{ABB}},\mathcal{Z}_{|\mathcal{IO}_{\text{ABB}}}}$ is distributed as the output from $\mathcal{F}_{\text{ABB}}$ in $\text{IDEAL}^{\mathcal{F}_{\text{BABB}}}_{\mathcal{F}_{\text{ABB}},\mathcal{T},\mathcal{Z}_{|\mathcal{IO}_{\text{ABB}}}}$. For all commands except `output`, this is straight-forward. To see this we take multiplication as an example. Assume first that the simulated multiplication protocol does not terminate. This $\mathcal{T}$ simulates by not specifying an output round for the multiplication command on $\mathcal{F}_{\text{ABB}}$. Assume then that the multiplication protocol *does* terminate. This $\mathcal{T}$ simulates by inputting *cid* to $\mathcal{F}_{\text{ABB}}$, which causes it to define $\text{val}(x) = \text{val}(x_1) \cdot \text{val}(x_2)$ and output $(cid, \texttt{defined})$ to all parties, as desired. So, consider the `output`-command. To see that the output from the `output`-command is the same in the protocol and the simulation basically involves proving the multiplication protocol correct.

We consider $\text{HYB}^{\mathcal{F}_{\text{BABB}}}_{\pi_{\text{ABB}},\mathcal{Z}_{|\mathcal{IO}_{\text{ABB}}}}$. Let

$$A = \sum_{j\in J_2} \text{val}(a_j) \ ,$$

[4]The a_i value is not a value which is defined on the functionality $\mathcal{F}_{\text{ABB}}$ in $\text{IDEAL}^{\mathcal{F}_{\text{BABB}}}_{\mathcal{F}_{\text{ABB}},\mathcal{T},\mathcal{Z}_{|\mathcal{IO}_{\text{ABB}}}}$; It is a value internal to the simulated multiplication protocol. The fact that we separated the name spaces using the $(0, x)$ and $(1, x)$ convention ensures that $\mathcal{Z}$ has had no opportunity of viewing a_i before. Therefore $\mathcal{S}$ is now free to set a_i as it desires.

[5]When all parties are corrupted, $\mathcal{T}$ will know the inputs of all parties and can compute $\text{val}(x_1)$.

where $\mathrm{val}(a_j)$ denotes the values of a_j in $\mathcal{F}_{\mathrm{BABB}}$. Since a_j was defined for all $j \in J_1 \supset J_2$, A is well-defined. Let

$$B = \sum_{j \in J_2} \mathrm{val}(b_j) \ ,$$

where $\mathrm{val}(b_j)$ denotes the values of b_j in $\mathcal{F}_{\mathrm{BABB}}$. Since b_j was defined for all $j \in J_2$, B is well-defined. We have that

$$\mathrm{val}(c) = v = \mathrm{val}(x_1) + A \ ,$$

so

$$\begin{aligned} \mathrm{val}(x) &= v\,\mathrm{val}(x_2) - \sum_{j \in J_2} \mathrm{val}(a_j)\,\mathrm{val}(x_2) \\ &= (\mathrm{val}(x_1) + A)\,\mathrm{val}(x_2) - A\,\mathrm{val}(x_2) \\ &= \mathrm{val}(x_1)\,\mathrm{val}(x_2) \ , \end{aligned}$$

which concludes the proof. □

8.3 ABB Threshold Homomorphic Encryption Schemes

We present a static secure realization of $\mathcal{F}_{\mathrm{BABB}}$ based on threshold homomorphic encryption schemes with some special properties. In lack of a better name we call a scheme which has the properties that allow us to realize $\mathcal{F}_{\mathrm{ABB}}$ w.r.t static adversaries a static ABB threshold homomorphic encryption scheme.

Definition 8.2 *A static ABB t-threshold homomorphic encryption scheme $\mathcal{THE}$ is an IND-CPA secure encryption scheme (gen, E, D) with the following properties:*

plaintext ring:
The plaintext space is a ring. Formally there exists a PPT family of rings (Definition 8.1 on page 258) with the generator defined by running $(pk, sk) \leftarrow gen(k)$ and then outputting pk. We denote this generator by gen^{pk} and denote the ring by $(R_{pk}, +_{pk}, \cdot_{pk})$.

perfect correctness:
For all $(pk, sk) \in gen$ and all $m \in R_{pk}$ and $r \in \{0,1\}^*$ it holds that $m = D_{sk}(E_{pk}(m; r))$.

additive homomorphic:
There exists a PPT algorithm, which given public key pk and encryptions $\overline{m}_1 \in E_{pk}(m_1)$ and $\overline{m}_2 \in E_{pk}(m_2)$ outputs a uniquely determined encryption $\overline{m} \in E_{pk}(m_1 +_{pk} m_2)$. We write $\overline{m} \leftarrow \overline{m}_1 \boxplus_{pk} \overline{m}_2$.

multiplication by constant:
There exists a PPT algorithm, which on input pk, $m_1 \in R_{pk}$ and $\overline{m}_2 \in E_{pk}(m_2)$ outputs a uniquely determined encryption $\overline{m} \in E_{pk}(m_1 \cdot_{pk} m_2)$. We write $\overline{m} \leftarrow m_1 \boxdot_{pk} \overline{m}_2$.

Protocol π_{ABB}

The protocol runs in the $\mathcal{F}_{\text{BABB}}$-hybrid model with parties $P_1, \ldots, P_n$.

multiplication:

On input $(cid, x \leftarrow x_1 \cdot x_2)$ party P_i proceeds as described below. In the description the protocol keeps reusing cid, which would violate $\mathcal{IO}_{\text{ABB}}$. To avoid this, when cid is input in round r, instead input $(cid, 0, r)$, and for the load for P_j and the private multiplication command for P_j use (cid, j, r). Furthermore, for all variable names x input from the environment, use the name $(0, x)$ on $\mathcal{F}_{\text{BABB}}$ and for all internal variable names x use the name $(1, x)$ instead. This prevents the environment from ever accessing internal values via e.g. output-commands.

1. Generate uniformly random $a_i \leftarrow R_{pk}$ and input
$$(cid, P_i, a_i \leftarrow a_i, R_{pk})$$
to $\mathcal{F}_{\text{BABB}}$. Furthermore, input $(cid, P_i, a_j \leftarrow R_{pk})$ for $j \in [n] \setminus \{i\}$.
2. Wait until `defined` or `fail` is output by $\mathcal{F}_{\text{BABB}}$ for all $j \in [n]$. Let $J_1 \subseteq [n]$ be those indices j for which `defined` was output. Then input
$$(cid, b_j \leftarrow a_j \cdot x_2)$$
to $\mathcal{F}_{\text{BABB}}$ for $j \in J_1$.
3. Wait until `defined` or `fail` is output by $\mathcal{F}_{\text{BABB}}$ for all $j \in J_1$. Let $J_2 \subseteq J_1$ be those indices j for which `defined` was output. Then input
$$(cid, c \leftarrow x_1 + \sum_{j \in J_2} a_j)$$
to $\mathcal{F}_{\text{BABB}}$.
4. Wait until `defined` is output by $\mathcal{F}_{\text{BABB}}$. Then input
$$(cid, \texttt{output}, c)$$
to $\mathcal{F}_{\text{BABB}}$.
5. Wait until (cid, v) is output by $\mathcal{F}_{\text{BABB}}$. Then input
$$(cid, x \leftarrow v \cdot x_2 - \sum_{i \in J_2} b_i)$$
to $\mathcal{F}_{\text{BABB}}$.
6. Wait until `defined` is output by $\mathcal{F}_{\text{BABB}}$. Then output $(cid, \texttt{defined})$.

$\neq$multiplication:

On all other inputs of the form (cid, V) for some value V, input $((cid, 0, 0), V)$ to $\mathcal{F}_{\text{BABB}}$. Then wait until $\mathcal{F}_{\text{BABB}}$ outputs some value of the form $((cid, 0, 0), W)$, and output (cid, W). Translate all variable names x in the command to $(0, x)$.

Figure 8.2: Realizing $\mathcal{F}_{\text{ABB}}$ in the $\mathcal{F}_{\text{BABB}}$-hybrid model.

blinding:

There exists a PPT algorithm, called the blinding algorithm*, which on input pk and $c \in E_{pk}(m)$ and uniformly random bits r_2, called the* randomizer*, outputs a random encryption c' distributed exactly as $E_{pk}(m;r)$ for uniformly random bits r. We write $c' \leftarrow \mathrm{Blind}_{pk}(c;r)$.*

proof of plaintext knowledge:

There exists a Σ-protocol Σ_1 for the relation $R_1 \subseteq (\{0,1\}^*)^2 \times (\{0,1\}^*)^2$ given by

$$(pk, y) \sim (x, r) \Leftrightarrow x \in R_{pk} \wedge y = E_{pk}(x;r) \wedge r \in \mathcal{R}_{pk} \ .$$

Where $\mathcal{R}_{pk}$ is the domain from which random bits are chosen in the encryption. The set $S(R_1)$ for the relation is given by pk being a correct public key, i.e. $pk \in gen^{pk}$.

proof of conditioned plaintext:

The scheme might also include a number of Σ-protocols for proving that the plaintext of a ciphertext is in some subset S of R_{pk}. Formally, there exists a Σ-protocol Σ_S for the relation $R_S \subseteq (\{0,1\}^*)^2 \times (\{0,1\}^*)^2$ given by

$$(pk, y) \sim (x, r) \Leftrightarrow x \in S \wedge y = E_{pk}(x;r) \wedge r \in \mathcal{R}_{pk} \ .$$

The set $S(R_S)$ of the relation is given by $pk \in gen^{pk}$.

It is enough that the proof of plaintext knowledge and proof of conditioned plaintext Σ_1 has computational special knowledge soundness relative to pk being a correctly generated random public key,[6] but the membership soundness should not be relative to any PPT family of sets.[7] Furthermore we allow for $r \in \{0,1\}^*$ in the extraction relation and that the Σ-protocols are private reference string Σ-protocols.

proof of correct multiplication:

There exists a Σ-protocol Σ_2 for the relation $R_2 \subseteq (\{0,1\}^*)^4 \times (\{0,1\}^*)^3$ given by

$$(pk, x, y, z) \sim (d, r_1, r_2) \Leftrightarrow y = E_{pk}(d; r_1) \wedge z = \mathrm{Blind}_{pk}(d \boxdot_{pk} x; r_2) \ .$$

It is enough that the Σ-protocol has special membership soundness and it is allowed to be a private reference string Σ-protocol. The set $S(R_2)$ of the relation is given by $pk \in gen^{pk}$.

threshold decryption:

There exists a protocol π_{THE} t-realizing $\mathcal{F}_{\mathrm{THE}}^{gen,E,D}$ from Fig. 8.3 on the following page (possibly in a $\mathcal{G}$-hybrid model) under $\mathcal{IO}_{\mathrm{static}}$.

We present two examples of static threshold homomorphic encryption schemes. The first scheme is based on the QRA and the strong RSA assumption. The second scheme is based on Paillier's cryptosystem. Katz and Yung [KY02] have presented a scheme which can be verified to also meet Definition 8.2 on page 263. Their scheme can also be based on the QRA and strong RSA, using techniques described below.

[6] Formally specified by the generator $pk \leftarrow gen^{pk}$ and the set of instances where the public key equals pk.

[7] It is discussed in Footnote 11 on page 282, exactly where in the proof it is used that the membership soundness in unconditional.

Functionality $\mathcal{F}_{\text{THE}}^{gen,E,D}$

The functionality runs with parties $P_1, \ldots, P_n$ and is parameterized by a public-key encryption scheme (gen, E, D). The functionality proceeds as follows:

key generation:

On input `init` from all honest parties in the same round, generate $(pk, sk) \leftarrow gen(k)$ and output pk to all parties and output pk on the SOT.

decryption:

If in some round all honest parties input (did, C), then output $(did, C, D_{sk}(C))$ on the SOT and output $(did, C, D_{sk}(C))$ to all honest parties in a round specified by the adversary.

incorrect inputs:

If the first non-trivial input from all honest parties is not `init` or two honest parties input `init` in different rounds or any honest party inputs `init` twice, then break down. If in some round any honest party inputs a value (did, C) and some honest party does not input (did, C), then break down. If any honest party uses the same decryption id did twice, then break down. If any honest party inputs (did, C), where C is not a valid ciphertext, then break down. Notice that this last check cannot necessarily be checked in PPT given just the inputs and outputs of $\mathcal{F}_{\text{THE}}$, as we have not required that we can check whether a string is a ciphertext given just the public key.[a]

[a] An alternative approach would be to realize the function $\{0,1\}^* \to \{0,1\}^*, C \mapsto D_{sk}(C)$ which e.g. outputs `invalid` when the input is not a valid ciphertext. Let's call this the strong functionality. The reason why we do not realize the strong functionality is that it might be harder and that we do not need this extended functionality. To see that realizing $\{0,1\}^* \to \{0,1\}^*, C \mapsto D_{sk}(C)$ might be harder than realizing the function $\mathcal{C}_{sk} \to \{0,1\}^*, C \mapsto D_{sk}(C)$, where $\mathcal{C}_{sk}$ is the set of valid ciphertexts, consider a set of ciphertexts $\mathcal{C}_{sk}$ and extend it to be, by definition, some set which cannot be recognized in PPT, even if sk is given. Then let all ciphertexts in the extension which is not also in the original set decrypt to a special symbol $\perp$, which is different from `invalid`. Then a realization of the strong functionality is not possible as the problem of determining whether to return $\perp$ or `invalid` is hard even given sk. Since in all contexts where we use the functionality we know that the input is a correct ciphertext when the decryption functionality is called it therefore makes sense to realize only the weaker functionality. Since in the cases where there is a difference the set $\mathcal{C}_{sk}$ must be hard to recognize we have to use the IO restriction to do the test, as the core functionality can only do PPT computations.

Figure 8.3: The threshold functionality for a public-key encryption scheme (gen, E, D).

8.3.1 Based on Paillier's Cryptosystem

In this section we present a static ABB threshold homomorphic encryption scheme based on Paillier's cryptosystem.

key generation:

Let $(p, q) \leftarrow gen(k)$ be an RSA generator for which $p' = (p-1)/2$ and $q' = (q-1)/2$ are primes and $\gcd(N, \lambda) = 1$, where $\lambda = 2p'q'$.

To generate a key-pair (pk, sk), run $(p, q) \leftarrow gen(k)$. Let $N = pq$ and let $d = \lambda(\lambda^{-1} \bmod N)$. Let $pk = N$ and let $sk = (N, d)$. The plaintext space is $\mathbf{Z}_N$ and the ciphertext

space is $\mathbf{Z}_{N^2}^*$.

encryption:
To encrypt, generate uniformly random $r \in \mathbf{Z}_N^*$ and let

$$E_N(m;r) = g^m r^N \bmod N^2 \ ,$$

where $g = N + 1$.

decryption:
Given $c \in \mathbf{Z}_{N^2}^*$, let

$$m = \frac{(c^d \bmod N) - 1}{N} \ .$$

See Section 2.7.2 for details, from where it also follows that the scheme has perfect correctness.

additive homomorphic:
For $C_1 = E_N(m_1; r_1)$ and $C_2 = E_N(m_2; r_2)$ let $C_1 \boxplus_N C_2 = C_1 C_2 \bmod N^2$. Then by Section 2.7.2 we have that $C_1 \boxplus_N C_2 = E_N(m_1 + m_2 \bmod N; r_1 r_2 \bmod N)$.

multiplication by constant:
For $C_1 = E_N(m_1; r_1)$ and $m \in \mathbf{Z}_N$, let $m \boxdot_N C_1 = C_1^m \bmod N = E_N(m m_1 \bmod N; r_1^m \bmod N)$.

blinding:
For $C_1 = E_N(m_1; r_1)$ and $r \in \mathbf{Z}_N^*$, let $\mathrm{Blind}_N(C_1; r) = C_1 r^N \bmod N^2 = E_N(m; r_1 r \bmod N)$.

proof of plaintext knowledge, correct multiplication:
To construct the proof of plaintext knowledge and correct multiplication we use that E_N is an N-invertible homomorphism as defined in Section 2.10. For the proof of plaintext knowledge one can use the proof of known K representation (with $K = g$). To prove correct multiplication one has to prove that $y = g^d r_1^N \bmod N^2$ and $z = x^d r_2^N \bmod N^2$ for which one can use the proof of linear relation between K representations, with $K_1 = g$, $c_1 = y$, $K_2 = x$ and $c_1 = z$ and $a_0 = 0$, $a_1 = 1$ and $a_2 = -1$.

proof of Boolean plaintext:
Finally we construct a proof of conditioned plaintext for all small sets $S \subset \mathbf{Z}_N$ — we are mostly interested in $S = \{0, 1\}$. Given a ciphertext $c = g^m r^N \bmod N^2$ a Σ-protocol Σ_{m_0} for proving knowledge of m and proving that m has a fixed constant value m_0, is given by the proof of linear relation between K representations, with $K_1 = g$, $c_1 = c$ and $a_0 = m_0$ and $a_1 = 1$. Then $\bigvee_{m_0 \in S} \Sigma_{m_0}$ is a Σ-protocol for proving restricted plaintext for S. See Section 2.9.8 for details.

threshold decryption:
A realization of the threshold functionality can be constructed using Theorem 6.1 on page 196 and Theorem 6.4 on page 204. Let $(p, q, e, d, v, w, sk_0) \leftarrow gen'$ denote the generator from Definition 6.3 on page 202, letting $s = 2$ and letting e be the empty string and letting $d = \lambda(\lambda^{-1} \bmod N)$. The public key output to the parties from

$\mathcal{F}_{\text{thresh}}^{gen'}$ in Fig. 6.1 on page 192 is (N, v, w), where v is a random generator of Q_N and $w = v^{\Delta^2 d} \bmod N$. We only need N, as the public key.

On input $c = E_N(m; r) \in \mathbf{Z}_{N^2}^*$ to $\mathcal{F}_{\text{thresh}}^{gen'}$ it returns $c^{4\Delta^2 d} \bmod N^2$. Since $c^{4\Delta^2} \in E_N(m4\Delta^2 \bmod N; r^{4\Delta^2} \bmod N)$ we can compute $m4\Delta^2 \bmod N = \frac{c^{4\Delta^2 d} \bmod N^2 - 1}{N}$, from which we can then compute m, as $4\Delta^2 \in \mathbf{Z}_N^*$ except for small values of k.

Theorem 8.2 *The Paillier scheme is a static ABB threshold homomorphic encryption scheme assuming the DCRA assumption for moduli N distributed as in* key generation *above.*

Proof. Except for the IND-CPA security of the encryption scheme all required properties were argued in the description of the scheme, or are straight-forward.

Proving that the IND-CPA security of the Paillier scheme reduces to the DCRA assumption for moduli N distributed as in key generation above is trivial, as the IND-CPA security is almost by definition equivalent to the DCRA.

Note that in the threshold decryption the ideal functionality $\mathcal{F}_{\text{THE}}$, that we claim to realize, outputs the public key N, whereas the ideal functionality $\mathcal{F}_{\text{thresh}}$, which we use to realize $\mathcal{F}_{\text{THE}}$, outputs (N, v, w), where v is a random generator of Q_{N^2} and $w = v^d \bmod N^2$. Therefore, proving the security of the reduction involves showing how to simulate values v and w given N. Notice however that the random generator of Q_{N^2} can be sampled, statistically close, by $v = E_N(m; r^2 \bmod N)$ for uniformly random $m \in \mathbf{Z}_N$ and uniformly random $r \in \mathbf{Z}_N^*$, and then $w = E_N(m; r^2 \bmod N)^d \bmod N^2 = mN + 1$. The side information from evaluations is simulated in the same way. □

8.3.2 Based on QRA and Strong RSA

As our second example of a static ABB threshold homomorphic encryption scheme we present a scheme based on the QRA (Assumption 2.7 on page 26) and the strong RSA assumption (Assumption 2.5 on page 25). The encryption scheme is a simplified variant of Franklin and Haber's system [FH96], extended with various protocols that we need for our purposes. A somewhat similar (but non-threshold) variant was suggested by Cramer and appears in [FH96]. In [CDN01] it was argued that a variant of the present scheme could be based on the QR and DDH assumptions. We feel that the version we give here based on the QRA and the strong RSA assumption is somewhat cleaner because we only use assumptions related to factoring and because the semantic security can be proved based only on the QRA. This follows from observations made by Cramer and Shoup [CS01] which was not known when [CDN01] was written. The strong RSA assumption is used to argue the computational soundness of the Σ-protocols. Let us call the scheme QRSRSA. It basically encrypts a bit b by the quadratic residuosity of the ciphertext in $\mathbf{Z}_N^*$. One is given a random generator h of Q_N and computes an encryption $\beta = (-1)^b h^r \bmod N$ for uniformly random r. Since -1 is not a quadratic residue, this is a quadratic residue iff $b = 0$. To be able to do a threshold decryption the value $g^r \bmod N$ is given as part of the ciphertext, where $h = g^x \bmod N$ for uniformly random x. The value x is the secret key. To decrypt $(\alpha, \beta) = (g^r, (-1)^b h^r) \bmod N$ one computes $\beta(\alpha^x)^{-1} \bmod N$, computing α^x using a function sharing of $\alpha \mapsto \alpha^x \bmod N$.

This scheme is slightly different from the scheme in [CDN01], where the form of an encryption was $(g^r \bmod N, h^{4\Delta^2 x}) \bmod N$ to allow for threshold decryption using a function sharing of $\alpha \mapsto \alpha^{4\Delta^2 x} \bmod N$ using Theorem 6.4 on page 204. This however turns out not to be secure in our setting, because of the side information leaked to the adversary. Given a ciphertext $(\alpha, \beta) = (g^r, (-1)^b h^{4\Delta^2 r}) \bmod N$, knowing $\alpha^{4\Delta^2 x} \bmod N = h^{4\Delta^2 r} \bmod N$ is equivalent to knowing b as $h^{4\Delta^2 r} \bmod N = (-1)\beta \bmod N$. If one in addition is given $h^r \bmod N$, then one learns additional information, because $h^r \bmod N$ cannot be computed efficiently from $h^{4\Delta^2} \bmod N$. In fact, given $\beta' = h^r \bmod N$ and b one can check that $\beta = (-1)^b \beta'^{4\Delta^2} \bmod N$, which is a *proof* that (α, β) encrypts b. The reason being that $\beta'^{4\Delta^2} \bmod N$ trivially is a quadratic residue, so that b is guaranteed to be the quadratic residuosity of β, which is in turn the plaintext value. Therefore one has in addition to the plaintext value obtain a proof of what the plaintext value is. Therefore the protocol is not zero-knowledge relative to its task. This would actually turn out to be a problem in the proof of Theorem 8.4 on page 275, and the author does not know a proof of security for the protocol using the original scheme in [CDN01].

key generation:

Let $(p, q) \leftarrow gen(k)$ be a generator for which the QR assumption and the strong RSA assumption hold, and for which $p' = (p-1)/2$ and $q' = (q-1)/2$ are $\lceil k/2 - 2 \rceil$-bit primes. Let $J_1 = Q_N \cup (-1)Q_N$ and let $M = p'q'$.

To generate a key-pair (pk, sk), run $(p, q) \leftarrow gen(k)$. Let $N = pq$, let $g = y^2 \bmod N$ for a uniformly random element $y \in \mathbf{Z}_N^*$, so that except with negligible probability g is a random generator of Q_N and let $h = g^x \bmod N$ for a uniformly random even number $x \in 2\mathbf{Z}_{2^{2k}}$. Let $pk = (N, g, h)$ and let $sk = (N, g, x)$.

encryption:

To encrypt $b \in \{0, 1\}$, generate uniformly random $r \in 2\mathbf{Z}_{2^{2k}}$ and let

$$E_{N,g,h}(b; r) = (g^r \bmod N, (-1)^b h^r \bmod N) \ .$$

Notice that because M is odd, $r \bmod M$ is statistically close to uniformly random in $\mathbf{Z}_M$, which is the reason for picking r from a large domain. The only reason for picking r even is to given the encryptor a witness that $\alpha = g^r \bmod N$ and $h^r \bmod N$ are quadratic residues.

decryption:

Given $(\alpha, \beta) = E_{N,g,h}(b; r)$, let

$$D_{N,g,x}(\alpha, \beta) = \frac{1}{2}(1 - (\beta \alpha^{-x} \bmod N)) \ .$$

additive homomorphic:

Given ciphertexts (α_1, β_1) and (α_2, β_2), let $(\alpha_1, \beta_1) \boxplus_N (\alpha_2, \beta_2) = (\alpha_1 \alpha_2 \bmod N, \beta_1 \beta_2 \bmod N)$. Then clearly $E_{N,g,h}(b_1; r_1) \boxplus_N E_{N,g,h}(b_2; r_2) = E_{N,g,h}(b_1 + b_2 \bmod 2; r_1 + r_2)$.

multiplication by constant:

Given a ciphertext $C = (\alpha, \beta)$, let $1 \boxdot C = C$ and let $0 \boxdot C = (1, 1)$.

blinding:

Given a ciphertext $C = (\alpha, \beta)$, let $\text{Blind}_{N,g,h}(C; r) = C \boxplus_N E_{N,g,h}(0; r)$ for uniformly random $r \in 2\mathbf{Z}_{2^{2k}}$.

proof of plaintext knowledge:

To prove plaintext knowledge for (α, β) it suffice to prove knowledge of a value r for which $\alpha \equiv g^r \pmod N$ and $\beta^2 \equiv h^{2r} \pmod N$, as this proves that $\alpha = g^r \bmod N$ and $\beta = \pm h^r \bmod N$. Given a witness for this, one can quickly find $b \in \{0, 1\}$ such that $(\alpha, \beta) = (g^r, (-1)^b h^r) \pmod N$. Since r is even the proof can be done using the Σ-protocol `equality of even RSA discrete logarithms`.

proof of correct multiplication:

First notice that proving equality of even RSA discrete logarithms for the instance $(N, 1, g, \alpha, h, \beta, 2^{2k})$ proves that (α, β) encrypts 0, and proving equality of even RSA discrete logarithms for $(N, s, g, \alpha, h, -\beta, 2^{2k})$ proves that (α, β) encrypts 1. Now assume that $y = E_{N,g,h}(d; r)$ and $z = \text{Blind}(d \boxdot x; r_2)$. Notice that if $d = 0$, then $z = \text{Blind}(0 \boxdot x; r_2) = E_{N,g,h}(0; r_2)$; And, if $d = 1$, then $z = \text{Blind}(1 \boxdot x; r_2) = x \boxplus_N E_{N,g,h}(0; r_2)$. To prove knowledge of (d, r, r_2) it is therefore enough to prove that either $y = E_{N,g,h}(0; r)$ and $z = E_{N,g,h}(0; r_2)$ or $y = E_{N,g,h}(1; r)$ and $z \boxminus_N x = E_{N,g,h}(0; r_2)$. This can be done with the above proofs and the $\wedge$-construction and the $\vee$-construction. See Section 2.9.8 for details.

threshold decryption:

A realization of the threshold functionality can be constructed using Theorem 6.1 on page 196 and Theorem 6.5 on page 205. Let $(p, q, e, d, v, w, sk_0) \leftarrow gen'$ denote the generator from Definition 6.4 on page 204, letting $s = 1$ and letting e be the empty string and letting $d \in 2\mathbf{Z}_{2^{2k}}$ be uniformly random. The public key output to the parties from $\mathcal{F}_{\text{thresh}}^{gen'}$ in Fig. 6.1 on page 192 is (N, v, w), where v is a random generator of Q_N and $w = v^d \bmod N$. Let $g = v$, $x = d$ and $h = w \bmod N$ and (N, g, h) is exactly a public key of the here scheme with private key x. Furthermore, on input α the functionality $\mathcal{F}_{\text{thresh}}^{gen'}$ returns $\alpha^d \bmod N$ from which one can trivially compute the decryption $b = \frac{1}{2}(1 - (\beta\alpha^{-x} \bmod N))$.

Notice that this way we only obtained a realization for the construction threshold $c = n$ (corruption threshold $t = n-1$), meaning that all parties have to participate to decrypt. Using the so-called share backup technique by Rabin [Rab98] one can however obtain any other construction threshold. Since we have a separate section dedicated to this technique, (Section 10.4), we do not give the details here, but simply claim that we can obtain a realization of $\mathcal{F}_{\text{THE}}$ for all c.

Independent of the work in [CDN01], Katz and Yung proposed another threshold homomorphic encryption scheme which can also be based on the QRA and the strong RSA. In the terminology we have set up here, they use a function sharing of $\beta \mapsto \beta^x \bmod N$ over $\mathbf{Z}_N^*$, where $x = M = p'q'$. Therefore $\beta^x \bmod N = \beta^M \bmod N \in \{-1, 1\}$, depending on the quadratic residuosity of β. Therefore the element α is not needed. In [KY02] an additive sharing is used and validity of shares is proved using `equality of RSA discrete logarithms` or a technique by Gennaro, Jarecki, Krawczyk and Rabin [GJKR00] based on

the so-called information checking technique. However, both techniques are insufficient as they only prove that the shares are correct up to a factor ± 1. As an example `equality of RSA discrete logarithms` proves knowledge of s_i such that $y_i^2 \bmod N = x^{2s_i} \bmod N$, where s_i is the secret share. This proves that $y_i = \pm x^{s_i} \bmod N$. But, that shares are correct up to a factor of ± 1 is clearly not sufficient when the result is ± 1. It is therefore necessary to use a proof of discrete RSA logarithms which also gets the quadratic residuosity correct, as e.g. the proof of even RSA discrete logarithms used above. This issue and has been acknowledged by the first author of [KY02] who also verifies that using `even RSA discrete logarithm` will take care of this issue for the scheme in [KY02]. The scheme in [KY02] can be extended to meet Definition 8.2 on page 263 using the techniques above.

Theorem 8.3 *The QRSRSA scheme is a static ABB threshold homomorphic encryption scheme assuming the QRA for moduli N distributed as described in* key generation *for QRSRSA and assuming the strong RSA assumption.*

Proof. Except for the correctness and the IND-CPA security of the encryption scheme all required properties were argued in the description of the scheme, or are straight-forward.

Notice that for the Σ-protocol `equality of even RSA discrete logarithms` to have computational knowledge soundness we have to assume the strong RSA assumption for moduli N and g as generated by *gen*, see Corollary 2.3 on page 56. Notice furthermore that it is not a problem that the Σ-protocol for `equality of even RSA discrete logarithms` has a larger extraction relation. The relation in proof of plaintext knowledge in Definition 8.2 on page 263 allows for a larger extraction relation.

Observe furthermore that having access to $\mathcal{F}_{\text{thresh}}^{gen'}$ we do not need to simulate any side information. For a ciphertext $(\alpha, \beta) = (g^r, (-1)^b h^r)$, in the simulation the simulator is given b from $\mathcal{F}_{\text{THE}}$ and must simulate $\mathcal{F}_{\text{thresh}}$ outputting $\alpha^x \bmod N$. But $\alpha^x = (-1)^b \beta \bmod N$, so knowing b is equivalent to knowing α^x. Notice that it would be a completely difference story if $\alpha^{x(2\Delta^2)^{-1}} \bmod N$ was leaked to the adversary as for the generator in Definition 6.3 on page 202. Knowing b is equivalent to knowing α^x, but one cannot efficiently compute $\alpha^{x(2\Delta^2)^{-1}} \bmod N$ from $\alpha^x \bmod N$ and therefore cannot simulate $\mathcal{F}_{\text{thresh}}^{gen'}$ given access to $\mathcal{F}_{\text{THE}}$.

We now prove that the IND-CPA security of the QRSRSA encryption scheme reduces to the QRA for moduli N distributed as for the key generator of QRSRSA. Assume that we are given a random element $B \in J_1$, and that we are to compute a bit $b \in \{0, 1\}$, where $c = 0$ iff $B \in Q_N$. The idea is to place B as part of an encryption (A, B), letting the quadratic residuosity of B determine the plaintext value.

Pick uniformly random $A \in Q_N$ and let

$$g = A^2 \bmod N \ ,$$

$$h = B^2 \bmod N \ .$$

Except with an exponentially small probability A generates Q_N, in which case g generates Q_N. So, assume that g generates Q_N and *define* $x \in \mathbf{Z}_M$ by

$$h = g^x \bmod N \ .$$

By the choice of A, the defined x is uniformly random in $\mathbf{Z}_M$. Therefore the defined key (N, g, h, x) is distributed statistically close to a real key.

Furthermore, using $h = g^x \bmod N$, $h = B^2 \bmod N$ and $g = A^2 \bmod N$ we get that

$$B^2 \equiv A^{2x} \pmod{N} .$$

Since $A^x \bmod N \in Q_N$ and $B \in J_1 = Q_N \cup (-1)Q_N$ it follows that

$$B \equiv (-1)^b A^x .$$

Then *define* $r \in \mathbf{Z}_M$ by

$$A = g^r \bmod N .$$

It follows that

$$B \equiv (-1)^b A^x \equiv (-1)^b g^{rx} \equiv (-1)^b h^r \pmod{N} ,$$

so it follows that $(A, B) \equiv (g^r, (-1)^b h^r) \pmod{N}$ is an encryption of b under (N, g, h). Then let $(\alpha, \beta) = \text{Blind}((A, B); r)$ to get a uniformly random encryption of b. It follows that the semantic security reduces to the QRA. □

8.4 From Threshold Homomorphic Encryption to Black-Box Arithmetic

We now show that if we are given any static ABB threshold homomorphic encryption scheme $\mathcal{THE} = (gen, E, D)$ we can realize $\mathcal{F}_{\text{BABB}}^{gen^{pk}}$, where the PPT family of rings gen^{pk} is given by running $(pk, sk) \leftarrow gen(k)$ and outputting pk.

We describe the protocol in the $(\mathcal{F}_{\text{ZK-PM}}^{R_1,R_S,R_2}, \mathcal{F}_{\text{THE}}^{\mathcal{THE}})$-hybrid model, where R_1, R_S, R_2 denote the relations of $\mathcal{THE}$. During the execution each party P_i maintains a dictionary enc_i mapping from variable names $x \in \{0,1\}^*$ into encryptions under pk. Initially $\text{enc}_i(x) = \perp$ for all $x \in \{0,1\}^*$. The dictionary is maintained such that for any two honest parties P_i and P_j, we have that $\text{enc}_i = \text{enc}_j$. This invariant will be evident from the implementation and we therefore skip the sub-fix on enc in most cases. The dictionary $\text{enc}_i(x)$ is basically an encrypted version the dictionary $\text{val}(x)$ held by $\mathcal{F}_{\text{BABB}}$. We say that the variable x is defined if $\text{enc}(x) \neq \perp$. For each defined variable x which P_i loaded, it keeps the information $\text{val}_i(x) \in R_{pk}$ and $\text{rnd}_i(x) \in \{0,1\}^*$ such that $\text{enc}(x) = E_{pk}(\text{val}_i(x); \text{rnd}_i(x))$. In the description of the protocols we assume that the inputs are according to $\mathcal{IO}_{\text{ABB}}$. What happens when this is not the case is not essential. The protocol is given in Fig. 8.4 on the facing page.

We would like to argue that if $\mathcal{THE}$ is a static ABB t-threshold homomorphic encryption scheme, according to Definition 8.2 on page 263, then $\pi_{\text{BABB}}^{\mathcal{THE}}$ t-realizes $\mathcal{F}_{\text{BABB}}^{gen^{pk}}$ in the $(\mathcal{F}_{\text{ZK-PM}}^{R_1,R_S,R_2}, \mathcal{F}_{\text{THE}}^{\mathcal{THE}})$-hybrid model and then plug in the realization of $\mathcal{F}_{\text{ZK-PM}}^{R_1,R_S,R_2}$ from Chapter 5 and the realization of $\mathcal{F}_{\text{THE}}^{\mathcal{THE}}$, from Definition 8.2 on page 263. But we can not do this. The problem is that we are using the zero-knowledge proof of membership functionality $\mathcal{F}_{\text{ZK-PM}}^{R_1,R_S,R_2}$, and a proof of membership is in general not enough in the load command. We need a proof of *knowledge* to guarantee independence of inputs. I.e. to guaranteed that a corrupted party does not copy a ciphertext broadcast by an honest party and broadcasts

Protocol $\pi^{\mathcal{THE}}_{\text{BABB}}$

The protocol runs with parties $P_1, \ldots, P_n$ in the $(\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PM}}, \mathcal{F}^{\mathcal{THE}}_{\text{THE}})$-hybrid model where R_1, R_S and R_2 are the relations for plaintext knowledge, conditioned plaintext respectively correct multiplication for $\mathcal{THE}$. The protocol proceeds as follows:

initialization:

On input `init`, input `init` to $\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PM}}$ and $\mathcal{F}^{\mathcal{THE}}_{\text{THE}}$. Wait until $\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PM}}$ outputs `ready` and $\mathcal{F}^{\mathcal{THE}}_{\text{THE}}$ outputs pk and ignore all inputs until this happens. Then output `ready`.

load:

1. On input $(cid, P_i, x \leftarrow s, S)$ party P_i sets $\text{val}(x) = s$, generates uniformly random bits $\text{rnd}(x)$ for encryption and sets $\text{enc}(x) = E_{pk}(\text{val}(x); \text{rnd}(x))$. It then broadcasts $\text{enc}(x)$.
2. The other parties, of which the honest by assumption all have received the input $(P_i, x \leftarrow S)$, receive and store $\text{enc}(x)$.
3. Then all parties input $(cid, R_S, P_i, (pk, \text{enc}(x)))$ to $\mathcal{F}_{\text{ZK-PM}}$ and P_i inputs $(cid, R_S, P_i, (\text{val}(x), \text{rnd}(x)))$ to $\mathcal{F}_{\text{ZK-PM}}$.
4. The parties wait for output (cid, b) from $\mathcal{F}_{\text{ZK-PM}}$.

If party P_j receives $(P_i, x \leftarrow S)$, but does not receive a broadcast message from P_i in the next round or $b = 0$, then set $\text{enc}(x) \leftarrow \perp$ and output $(cid, \texttt{fail})$. Otherwise, output $(cid, \texttt{defined})$.

linear combination:

If P_i receives the input $(cid, x \leftarrow a_0 + \sum_{j=1}^{l} a_j x_j)$, then it computes $\text{enc}(x) \leftarrow a_0 \boxplus (\boxplus_{j=1}^{l} a_j \boxdot \text{enc}(x_j))$.

Figure 8.4(a) (cont. in Fig. 8.4(b) on the next page): The basic arithmetic black-box protocol for a threshold homomorphic encryption scheme $\mathcal{THE}$

that encryption as his own. Notice that the $\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PM}}$-hybrid model actually allows a corrupted party to do this without being caught. If a ciphertext c is broadcast by an honest party, a corrupted party P_j can randomize c into c' and broadcast c'. When P_j has to prove plaintext 'knowledge' with R_1 the environment will simply instruct $\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PM}}$ to accept c'. It can safely do this: The value c was broadcast by an honest party and is therefore guaranteed to be a ciphertext. Therefore $\mathcal{Z}$ would not violate $\mathcal{IO}_{\text{ZK-PM}}$ by letting $\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PM}}$ accept c' for the relation R_1. This allows a corrupted party to always input the same value to the computation as a given honest party. This is a behavior which we do not want to be possible, and it is certainly a behavior which is not possible in the ideal process with $\mathcal{F}_{\text{BABB}}$. Therefore the ability to copy inputs would be an accomplishment of the real-life corrupted parties extra to that of any ideal process adversary, showing that such a protocol would be insure.

If on the other hand we used the zero-knowledge proof of knowledge functionality $\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PK}}$ for realizing $\pi^{\mathcal{THE}}_{\text{BABB}}$, then we could indeed prove the protocol secure, but we would not know how to efficiently realize $\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PK}}$. Unfortunate we are caught somewhere in-between. What we will do is to plug in the realization of $\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PM}}$ from Chapter 5 and then prove that the

Protocol $\pi^{\mathcal{THE}}_{\text{BABB}}$

private multiplication:

1. If P_i receives the input $(cid, P_i, x \leftarrow x_1 \cdot x_2)$, where $\text{enc}(x_1) = E_{pk}(\text{val}_i(x_1); \text{rnd}_i(x_1))$, then it computes $\text{enc}(x) \leftarrow \text{Blind}_{pk}(\text{val}(x_1) \boxdot \text{enc}(x_2); r_2)$ for uniformly random bits r_2 for the blinding algorithm. It then broadcasts $\text{enc}(x)$.
2. The other parties, of which the honest parties under $\mathcal{IO}_{\text{ABB}}$ have received the input $(cid, P_i, x \leftarrow x_1 \cdot x_2)$, receive and store $\text{enc}(x)$.
3. Then P_j for $j \neq i$ inputs $(cid, R_2, P_i, (pk, \text{enc}(x_1), \text{enc}(x_2), \text{enc}(x)))$ to $\mathcal{F}_{\text{ZK-PM}}$ and P_i inputs $(cid, R_2, P_i, (\text{val}(x_1), \text{rnd}(x_1), r_2))$ to $\mathcal{F}_{\text{ZK-PM}}$. This is possible by the invariant as $\text{val}(x_1)$ and $\text{rnd}(x_1)$ are known to P_i.
4. The parties wait for output (cid, b) from $\mathcal{F}_{\text{ZK-PM}}$.

If party P_j receives $(cid, P_i, x \leftarrow x_1 \cdot x_2)$ and does not receive a broadcast message from P_i or $b = 0$, then P_j sets $\text{enc}(x) \leftarrow \perp$ and outputs $(cid, \texttt{fail})$. Otherwise P_j outputs $(cid, \texttt{defined})$.

output:

If P_i receives the input $(cid, \texttt{output}, x)$ it inputs $(cid, \text{enc}(x))$ to $\mathcal{F}_{\text{THE}}$. If $\mathcal{F}_{\text{THE}}$ ever outputs (cid, m) then P_i outputs (cid, m).

aligned output:

If more than one commands is activated in the same round, then the outputs are delivered when all the commands have terminated.

Figure 8.4(b) (cont. from Fig. 8.4(a) on the page before): The basic arithmetic black-box protocol for a threshold homomorphic encryption scheme $\mathcal{THE}$

composed protocol t-realizes $\mathcal{F}^{gen^{pk}}_{\text{BABB}}$ in the $\mathcal{F}^{\mathcal{THE}}_{\text{THE}}$-hybrid model. The main reason why we can do this when we cannot prove $\pi^{\mathcal{THE}}_{\text{BABB}}$ secure in the $(\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PM}}, \mathcal{F}^{\mathcal{THE}}_{\text{THE}})$-hybrid model is that the realization of $\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PM}}$ from Chapter 5 has a salient property extra to realizing $\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PM}}$, namely off-line extraction.

We give the main idea of how we are going to exploit the off-line extraction. Recall that $\pi^{\mathcal{THE}}_{\text{BABB}}$ only having off-line extraction, as opposed to on-line extraction, means that whereas the simulator from the proof of Theorem 5.1 on page 180, let us call it S, can simulate the proof of honest parties without knowing the witnesses, it cannot extract the witnesses of the accepted proofs of corrupted parties, unless it is allowed to rewind the environment. Therefore the simulator is not sufficient for a case as the load command, where we need to extract the plaintext. And, we *do* need the plaintext: The simulator that we construct, let us call it $\mathcal{S}$, will be running in the ideal process with $\mathcal{F}_{\text{BABB}}$ and must simulate π_{BABB}. This means that if a corrupted party P_i broadcasts an encryption c and gives an acceptable proof of plaintext knowledge, then the environment $\mathcal{Z}$ expects to see the honest parties output $(cid, \texttt{defined})$. To make them do that the simulator has to input some value s to $\mathcal{F}_{\text{BABB}}$, on behalf of P_i, to define $\text{val}(x) = s$. If the simulation and $\mathcal{F}_{\text{BABB}}$ is to be consistent, then this value should indeed be the plaintext of c. If we input any other value $s' \neq D_{sk}(c)$ and the environment $\mathcal{Z}$ later inputs $(cid', \texttt{output}, x)$, then $\mathcal{F}_{\text{BABB}}$ will output (cid', s') to $\mathcal{Z}$ from all

honest parties and it will be obvious to $\mathcal{Z}$ that it is viewing the simulation — in the protocol the parties would have decrypted c and output $D_{sk}(c)$.

A very simple observation lets us do without the on-line extraction: We are proving statements about ciphertexts and the witnesses that we are interested in are the plaintexts. There is a more prosaic approach to obtaining plaintexts than rewinding zero-knowledge proofs, namely decryption. So, that is what we will do. Each time the environment broadcast a ciphertext c on behalf of a corrupted party and gives an acceptable proof of knowledge, we decrypt c to obtain the value s to input to $\mathcal{F}_{\text{BABB}}$ on behalf of P_i.

This approach leaves two technical questions: First of all, where does the simulator get the private key from? And second, when the simulator is using the private key, how can we rely on the semantic security of the encryption scheme? The answer to the first question is simple: Because the whole setup was carefully designed to reveal sk to the simulator.[8] The simulator has access to the SOT of $\mathcal{F}_{\text{BABB}}^{gen^{pk}}$, and the first thing $\mathcal{F}_{\text{BABB}}^{gen^{pk}}$ ever does is to generate a public key $pk \leftarrow gen^{pk}(k;r)$ which is used to define the ring which $\mathcal{F}_{\text{BABB}}$ does arithmetic over and then output r on the SOT. By definition, the generator gen^{pk} runs $(pk, sk) \leftarrow gen(k;r)$ and outputs pk. So, $\mathcal{S}$ takes r, reruns $(pk, sk) \leftarrow gen(k;r)$ and has sk. The answer to the second question is to use a hybrid, the clue being that the view which $\mathcal{Z}$ sees in $\text{IDEAL}_{\mathcal{F}_{\text{BABB}}^{R_1,R_S,R_2},\mathcal{S},\mathcal{Z}}$ can be generated without using the private key. Define namely a hybrid distribution H as follows: Take $\text{IDEAL}_{\mathcal{F}_{\text{BABB}}^{R_1,R_S,R_2},\mathcal{S},\mathcal{Z}}$ and start running it, but without giving sk to $\mathcal{S}$. Each time $\mathcal{Z}$ gives an acceptable proof $\mathcal{S}$ will then be stuck, but we can then rewind the whole experiment $\text{IDEAL}_{\mathcal{F}_{\text{BABB}}^{R_1,R_S,R_2},\mathcal{S},\mathcal{Z}}$ and *extract* a witness from the accepted proof instead. We then give the witness to $\mathcal{S}$, which continues the simulation where it gave up. The view which $\mathcal{Z}$ has in H is indistinguishable from its view in $\text{IDEAL}_{\mathcal{F}_{\text{BABB}}^{R_1,R_S,R_2},\mathcal{S},\mathcal{Z}}$, so it is enough to prove that H is indistinguishable from the protocol. And, since H does not use sk we can now appeal to the IND-CPA security of the encryption scheme. We also use this trick in the following chapters, so we might as well give it a name. We dub it pushing the rewinding to the analysis.

The proof of the following theorem will be carried out in great detail. This is because it is the first time we use the *pushing the rewinding to the analysis* techniques. We use the technique again in the two following chapters. Throughout the proof, the reader should appreciate that the intuition build by the detailed application of the *pushing the rewinding to the analysis* technique will allow us to use the technique in a more colloquial way in later chapters and therefore focus on the extra challenges that we encounter in this chapters.

Theorem 8.4 *Let $\mathcal{THE} = (gen, E, D)$ be a static ABB t-threshold homomorphic encryption scheme, with Σ-protocols Σ_1, Σ_S and Σ_2 for the relations R_1, R_S respectively R_2. Let $\pi_{\text{DAM-ZK}}^{\Sigma_1,\Sigma_S,\Sigma_2}$ be the realization of $\mathcal{F}_{\text{ZK-PM}}^{R_1,R_S,R_2}$ from Fig. 5.1 on page 180, being run as a proof of knowledge. For all $t < (n-1)/2$, the protocol $\pi_{\text{BABB}}^{\mathcal{THE}}[\pi_{\text{DAM-ZK}}^{\Sigma_1,\Sigma_S,\Sigma_2}/\mathcal{F}_{\text{ZK-PM}}^{R_1,R_S,R_2}]$ t-realizes $\mathcal{F}_{\text{BABB}}^{gen^{pk}}$ in the $(\mathcal{F}_{\text{THE}}^{\mathcal{THE}}, \mathcal{F}_{\text{PRS}}^{\text{commit},\mathtt{prs}})$-hybrid model under $\mathcal{IO}_{\text{static}}(t)$, where $\mathtt{prs}$ is the possible private reference string for Σ_1, Σ_S and Σ_2.*

Proof. In the following we use π_{BABB} as a replacement for the delightful, but slightly clumsy

[8] This is the issue discussed in Remark 8.1 on page 258.

Interface $\mathcal{S}_{\text{BABB}}$

Before anything else, parties are corrupted. Let C denote the set of corrupted parties.

initialize:

Initialize the simulator $\mathcal{S}_{\text{DAM-ZK}}$, given by Fig. 5.2 on page 182, for the $\pi_{\text{DAM-ZK}}$ protocol, by inputting the security parameter k and some random bits for its execution. I.e. $\mathcal{S}_{\text{BABB}}$ will be of the form $\mathcal{S}_{\text{BABB}}'[\mathcal{S}_{\text{DAM-ZK}}]$, see Fig. 3.13 on page 85.

All parties have received the `init` command and $\mathcal{F}_{\text{BABB}}$ has output r on its SOT. Compute $(pk, sk) = gen(k; r)$. Simulate that all honest parties input `init` to $\mathcal{F}_{\text{THE}}$ and then simulate the output pk on the SOT of $\mathcal{F}_{\text{THE}}$ and the output pk from all parties.

load:

On the load command, if $i \in C$, then all honest parties have input $(cid, P_i, x \leftarrow S)$. We simulate the protocol to $\mathcal{Z}$ as follows: First $\mathcal{Z}$ broadcasts a message $\text{enc}(x)$ (on behalf of P_i). We use ${\mathcal{S}_{\text{DAM-ZK}}}^{R_S}$ to simulate the run of $\pi^{R_S}_{\text{DAM-ZK}}$, with input $(cid, (pk, \text{enc}(x)))$ to all honest parties. If ${\mathcal{S}_{\text{DAM-ZK}}}^{R_S}$ accepts, then let $\text{val}(x) = D_{sk}(\text{enc}(x))$ and store $\text{enc}(x)$ and $\text{val}(x)$. Then input $(cid, \texttt{term}, \text{val}(x))$ on the SIT of $\mathcal{F}_{\text{BABB}}$ on behalf of P_i to define $\text{val}(x)$ in $\mathcal{F}_{\text{BABB}}$. If P_i fails the protocol, then input $(cid, \texttt{fail})$ on the SIT of $\mathcal{F}_{\text{BABB}}$. This simulates successfully, **except** in the case where $s \notin S$, where the load is rejected.

If $i \in H$, then P_i, in the ideal evaluation, have received $(cid, P_i, x \leftarrow s, S)$ for some $s \in S$. Therefore we should set $\text{enc}(x)$ to be a random encryption of s in $\mathcal{S}_{\text{BABB}}$, and show this encryption to $\mathcal{Z}$ to simulate a broadcast. However, since $\mathcal{S}_{\text{BABB}}$ does not learn the value of s, it cannot do this. Instead it sets $\text{enc}(x)$ to be a random encryption of s_0, where $s_0 \in S$ is some fixed dummy value, e.g. $s_0 = 0$ if $S = R_{pk}$. Then $\mathcal{S}_{\text{BABB}}$ shows $\text{enc}(x)$ to $\mathcal{Z}$ to simulate a broadcast. Then use ${\mathcal{S}_{\text{DAM-ZK}}}^{R_S}$ to simulate the run of $\pi^{R_S}_{\text{DAM-ZK}}$, with input $(cid, (pk, \text{enc}(x)))$ to all honest parties P_j. Provide this input to $\mathcal{S}_{\text{DAM-ZK}}$ simulating that it came from an ideal functionality $\mathcal{F}_{\text{ZK-PM}}$ — notice that indeed ${\mathcal{S}_{\text{DAM-ZK}}}^{R_S}$ does *not* expect to receive the witness from the honest P_i and that $(pk, \text{enc}(x)) \in L(R_S)$. This simulates successfully **except** that $\text{enc}(x) \leftarrow E_{pk}(s_0)$ instead of $\text{enc}(x) \leftarrow E_{pk}(s)$.

linear combination:

On input $(cid, x \leftarrow a_0 + \sum_{j=1}^{l} a_j x_j)$, compute $\text{enc}(x)$ as in the protocol. This simulates successfully.

Figure 8.5(a) (cont. in Fig. 8.5(b) on the facing page): The interface $\mathcal{S}_{\text{BABB}}$ used in the proof of Theorem 8.4 on the page before

symbol $\pi^{\mathcal{THE}}_{\text{BABB}}[\pi^{\Sigma_1,\Sigma_S,\Sigma_2}_{\text{DAM-ZK}}/\mathcal{F}^{R_1,R_S,R_2}_{\text{ZK-PM}}]$, and we use $\mathcal{F}_{\text{BABB}}$ to denote $\mathcal{F}^{gen^{pk}}_{\text{BABB}}$. We will not carry the IO behavior $\mathcal{IO}_{\text{static}}$ with us in the notation either.

To prove the theorem we construct a PPT simulator $\mathcal{S}_{\text{BABB}}$ such that

$$\text{IDEAL}^{\mathcal{F}_{\text{THE}},\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{BABB}},\mathcal{Z}} \overset{\text{c}}{\approx} \text{HYB}^{\mathcal{F}_{\text{THE}},\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\pi_{\text{BABB}},\mathcal{Z}} \tag{8.1}$$

for all environments $\mathcal{Z}$ for which $\Pr[\text{IDEAL}^{\mathcal{F}_{\text{THE}},\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{BABB}},\mathcal{Z}} = \text{``H} \rightsquigarrow \mathcal{F}_{\text{BABB}}\text{''}] \overset{\text{c}}{\approx} 0$. The simulator is given in Fig. 8.5. It is straight-forward to verify that $\mathcal{S}_{\text{BABB}}$ is PPT, using that $\mathcal{S}_{\text{DAM-ZK}}$ is PPT.

Interface $\mathcal{S}_{\text{BABB}}$

private multiplication:

On input $(cid, P_i, x \leftarrow x_1 \cdot x_2)$, if $i \in C$, then receive $\text{enc}(x)$ from $\mathcal{Z}$. Then use ${\mathcal{S}_{\text{DAM-ZK}}}^{R_2}$ to simulate the run of $\pi^{R_2}_{\text{DAM-ZK}}$. If ${\mathcal{S}_{\text{DAM-ZK}}}^{R_2}$ accepts the proof, then store $\text{enc}(x)$ and input $(cid, \texttt{term})$ on the SIT of $\mathcal{F}_{\text{BABB}}$ to define $\text{val}(x) = \text{val}(x_1) \cdot \text{val}(x_2)$ in $\mathcal{F}_{\text{BABB}}$. If $\mathcal{Z}$ fails the protocol, then input $(cid, \texttt{fail})$ on the SIT of $\mathcal{F}_{\text{BABB}}$. This simulates successfully.

If $i \in H$ then handle it similar to the load command by setting $\text{enc}(x) = \text{Blind}_{pk}(0 \boxdot \text{enc}(x_2); r_2)$. This simulates successfully **except** that $\text{enc}(x)$ encrypts 0 instead of $D_{sk}(\text{enc}(x_1)) \cdot D_{sk}(\text{enc}(x_2))$. Notice that indeed $(pk, \text{enc}(x_1), \text{enc}(x_2), \text{enc}(x)) \in L(R_2)$.

output:

On seeing the value $(cid, \texttt{output}, x, \text{val}(x))$ on the SOT of $\mathcal{F}_{\text{BABB}}$, simulate the input of $(cid, \text{enc}(x))$ to $\mathcal{F}_{\text{THE}}$ from all honest parties. If $\mathcal{Z}$ terminates the call to $\mathcal{F}_{\text{THE}}$ in the simulation, then simulate this by outputting $(cid, \text{val}(x))$ to all parties from $\mathcal{F}_{\text{THE}}$. This simulates successfully **except** if $val(x) \neq D_{sk}(\text{enc}(x))$.

Figure 8.5(b) (cont. from Fig. 8.5(a) on the facing page): The interface $\mathcal{S}_{\text{BABB}}$ used in the proof of Theorem 8.4 on page 275

The simulator $\mathcal{S}_{\text{BABB}}$ holds internally a list of defined variables and stores at each variable the value $\text{enc}(x)$, and at some variables the value $\text{val}(x)$. The goal of the simulation is that after each activation the variables defined in $\mathcal{S}_{\text{BABB}}$ are the same as those defined in the copy of $\mathcal{F}_{\text{BABB}}$ in $\text{IDEAL}^{\mathcal{F}_{\text{THE}}, \mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}_{\text{BABB}}, \mathcal{S}_{\text{BABB}}, \mathcal{Z}}$. Besides this, if P_i loaded x, i.e. the command $(cid, P_i, x \leftarrow S)$ was carried out at some point, and P_i is corrupted, then $\mathcal{S}_{\text{BABB}}$ also knows the value $\text{val}(x)$ which x is defined to in $\mathcal{F}_{\text{BABB}}$. We now prove Eq. 8.1 on the preceding page using a hybrids argument.

We first exploit that $\mathcal{S}_{\text{BABB}}$ can be divided into $\mathcal{S}_{\text{DAM-ZK}}$ and the part using $\mathcal{S}_{\text{DAM-ZK}}$, let us call it ${\mathcal{S}_{\text{BABB}}}'$. For proof technical reasons, arrange these parts on a form such that $\mathcal{S}_{\text{BABB}} = {\mathcal{S}_{\text{BABB}}}'[\mathcal{S}_{\text{DAM-ZK}}]$ for the composition operation in Fig. 3.13 on page 85. This is possible because ${\mathcal{S}_{\text{BABB}}}'$ provides the instances to $\mathcal{S}_{\text{DAM-ZK}}$ simulating that they come from an ideal functionality $\mathcal{F}_{\text{ZK-PM}}$. We use this to construct an environment $\mathcal{Y}$ for which

$$\text{IDEAL}^{\mathcal{F}_{\text{THE}}, \mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}_{\text{BABB}}, {\mathcal{S}_{\text{BABB}}}'[\mathcal{S}_{\text{DAM-ZK}}], \mathcal{Z}} \overset{\text{c}}{\approx} \text{IDEAL}^{\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}_{\text{ZK-PM}}, \mathcal{S}_{\text{DAM-ZK}}, \mathcal{Y}} \quad . \tag{8.2}$$

The environment $\mathcal{Y}$ runs as follows: On input (k, z) it runs $\text{IDEAL}^{\mathcal{F}_{\text{THE}}, \mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}_{\text{BABB}}, {\mathcal{S}_{\text{BABB}}}'[\mathcal{S}_{\text{DAM-ZK}}], \mathcal{Z}}(k, z)$ with the difference that whenever ${\mathcal{S}_{\text{BABB}}}'$ is about to access its copy of $\mathcal{S}_{\text{DAM-ZK}}$, instead the copy of $\mathcal{S}_{\text{DAM-ZK}}$ in $\text{IDEAL}^{\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}_{\text{ZK-PM}}, \mathcal{S}_{\text{DAM-ZK}}, \mathcal{Y}}$ is used. We therefore write $\text{IDEAL}^{\mathcal{F}_{\text{THE}}, \mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}_{\text{BABB}}, {\mathcal{S}_{\text{BABB}}}'[\cdot], \mathcal{Z}}$ to stress the ${\mathcal{S}_{\text{BABB}}}'$ does not have direct access to $\mathcal{S}_{\text{DAM-ZK}}$. Notice that the value provided to $\mathcal{S}_{\text{DAM-ZK}}$ by ${\mathcal{S}_{\text{BABB}}}'$ in initialize in Fig. 8.5 on the preceding page is provided by the execution in $\text{IDEAL}^{\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}_{\text{ZK-PM}}, \mathcal{S}_{\text{DAM-ZK}}, \mathcal{Y}}$. So far so good. When $\mathcal{S}_{\text{DAM-ZK}}$ is simulating honest parties receiving proofs, $\mathcal{Y}$ simply observes the communication in $\text{IDEAL}^{\mathcal{F}^{\text{commit,prs}}_{\text{PRS}}}_{\mathcal{F}_{\text{ZK-PM}}, \mathcal{S}_{\text{DAM-ZK}}, \mathcal{Y}}$ and merges this communication (which is simulated by $\mathcal{S}_{\text{DAM-ZK}}$) with that simulated by ${\mathcal{S}_{\text{BABB}}}'$ in

$\text{IDEAL}^{\mathcal{F}_{\text{THE}},\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{BABB}}{}'[\cdot],\mathcal{Z}}$. This is done as in Fig. 3.13 on page 85. In the load command, when $\mathcal{S}_{\text{BABB}}{}'$ would use $\mathcal{S}_{\text{DAM-ZK}}{}^{R_S}$ to simulate the proof with instance $(cid,(pk,\text{enc}(x)))$, the environment $\mathcal{Y}$ instead inputs $(cid,(pk,\text{enc}(x)),(s_0,r_0))$ to P_i in $\text{IDEAL}^{\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}$, where r_0 is the random bits used when $\text{enc}(x)$ was computed. Since $((pk,\text{enc}(x)),(s_0,r_0)) \in R_S$ this results in $\mathcal{S}_{\text{DAM-ZK}}$ receiving $(cid,(pk,\text{enc}(x)))$, exactly as in $\text{IDEAL}^{\mathcal{F}_{\text{THE}},\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{BABB}}{}'[\mathcal{S}_{\text{DAM-ZK}}],\mathcal{Z}}$. The private load command is dealt with in the same way, using that we know a witness of $\text{enc}(x_1)$ as x_1 was loaded by P_1 — and if not, then P_i simply terminates the execution of the command. When $\text{IDEAL}^{\mathcal{F}_{\text{THE}},\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{BABB}}{}'[\cdot],\mathcal{Z}}$ outputs a guess, then output the same guess. This ends the description of $\mathcal{Y}$ and establishes Eq. 8.2 on the page before. The reason why we do not get $\text{IDEAL}^{\mathcal{F}_{\text{THE}},\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{BABB}}{}'[\mathcal{S}_{\text{DAM-ZK}}],\mathcal{Z}} = \text{IDEAL}^{\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}$ is that there is a negligible probability that $\text{IDEAL}^{\mathcal{F}_{\text{THE}},\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{BABB}}{}'[\mathcal{S}_{\text{DAM-ZK}}],\mathcal{Z}}$ outputs ``$\texttt{H} \rightsquigarrow \mathcal{F}_{\text{BABB}}$''.[9] We then prove that

$$\text{IDEAL}^{\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}} \overset{\text{c}}{\approx} \text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}} \tag{8.3}$$

or that with a non-negligible probability $\text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}$ outputs a break of the extractor for an instance input by $\mathcal{Y}$. This follows from Corollary 5.3 on page 186 and the fact that $\Pr[\text{IDEAL}^{\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}} \in$ ``$\texttt{H} \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$''$] = 0$, by construction of $\mathcal{Y}$.

Assume first that with a non-negligible probability $\text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}$ outputs a break of the extractor for an instance input by $\mathcal{Y}$. We use this to reach a contradiction to the computational special knowledge soundness of Σ_1 or Σ_S. The Σ-protocol Σ_2 is only required to have membership soundness, and is not extracted. To reach a contradiction we must be able to take as input a random public key, see proof of conditioned plaintext in Definition 8.2 on page 263, and produce a break of the extractor for an instance containing that key. This means that we should be able to run $\text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}$ on a given public key pk and a given private reference string s. Notice that the simulator already does not use the trapdoor t_s of s, so running on a given s is no problem. However, the secret key sk of pk *is* used by the simulator, so we cannot simply run on a given public key pk. To facilitate this we run the ideal process in a particular way: In the following when we say that $\text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}$ is to be run on a given public key pk we mean that we run $\text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}$ with the following changes:

1. Do not run the check of the IO behavior of $\mathcal{F}_{\text{ZK-PK}}$ as the test whether the inputs are ciphertexts not necessarily is PPT. It does not matter either whether the test is run, as $\Pr[\text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}} =$ ``$\texttt{H} \rightsquigarrow \mathcal{F}_{\text{ZK-PK}}$''$] = 0$. This modification does not affect the distribution, neither the view of $\mathcal{Z}$ or the probability that a break of the extractor is produced.

2. When $\mathcal{F}_{\text{BABB}}$, in the copy of $\text{IDEAL}^{\mathcal{F}_{\text{THE}},\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{BABB}}{}'[\cdot],\mathcal{Z}}$ inside $\mathcal{Y}$, is about to run *gen* to

[9] By construction of $\mathcal{S}_{\text{DAM-ZK}}$ we have that $\Pr[\text{IDEAL}^{\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}} =$ ``$\texttt{C} \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}$''$] = 0$.

create a key-pair, instead use the given public key pk, and in $\mathcal{S}_{\text{BABB}}$ use the same key. Now that we do not know sk, when the execution gets to the place in load where $D_{sk}(\text{enc}(x))$ is computed, find $D_{sk}(\text{enc}(x))$ as follows instead: Since $\mathcal{S}_{\text{BABB}}{}'$ needs to compute $D_{sk}(\text{enc}(x))$, it accepted a proof of plaintext knowledge for $\text{enc}(x)$ from a corrupted party. We are running $\text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}(k,z)$, so by definition of $\mathcal{F}_{\text{ZK-PK}}$ the proof was accepted because $\mathcal{S}_{\text{DAM-ZK}}$ input a correct witness (m,r) on the SIT of $\mathcal{F}_{\text{ZK-PK}}$ to make it accept. The adversary S simply reads this witness off the SIT of $\mathcal{F}_{\text{ZK-PK}}$ instead of decrypting. We then have $m \in R_{pk}$ and $\text{enc}(c) = E_{pk}(m;r)$. By the perfect decryption requirement, see Definition 8.2 on page 263, we have found $m = D_{sk}(\text{enc}(x))$ as required. This modification does not affect the view of $\mathcal{Z}$ or the probability that a break of the extractor is produced.

Now we can generate the distribution $\text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}$ without using sk. Consider then the following adversary A against the computational special knowledge soundness of R_1 and R_S under random keys. See Section 2.9 for details. The adversary A is given (k,z,i) where i specified the legal instances. In our case $i = pk$ is a random public key, specifying the set of instances with this key in them. Then start running $\text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}(k,z)$ on the given public key pk as described above. Because the probability that a break of the extractor is produced was not changed, with non-negligible probability we receive a break of the extractor for an instance input by $\mathcal{Y}$, which by construction of $\mathcal{Y}$ all contain pk. So, we output this break. This means that we break the computational special knowledge soundness of R_1 and R_S under random keys. A contradiction. This allow us to conclude Eq. 8.3 on the preceding page. Notice that the adversary A was *expected* PPT as it ran $\text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}$. We show how to deal with this in Footnote 10 on the following page.

We then prove that

$$\text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}} \overset{\text{c}}{\approx} \text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}} , \tag{8.4}$$

where $\mathcal{X}$ behaves exactly as $\mathcal{Y}$ except for the following modification: During the execution of $\text{IDEAL}^{\mathcal{F}_{\text{THE}},\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{BABB}}{}'[\cdot],\mathcal{Z}}$, when at the second "**except**" $\mathcal{S}_{\text{BABB}}{}'$ is about to set $\text{enc}(x)$ to be a random encryption of s_0, instead inspect the communication between $\mathcal{Z}$ and $\mathcal{F}_{\text{BABB}}$ in $\text{IDEAL}^{\mathcal{F}_{\text{THE}},\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{BABB}}{}'[\cdot],\mathcal{Z}}$ to determine the correct value s. Then let $\text{enc}(x)$ be an encryption of s, as in the protocol. Also, at the third "**except**" in the private multiplication, use the correct value instead of 0. The correct value is defined to be $D_{sk}(\text{enc}(x_1))$, which can be found *without using sk* as follows: When P_i is given the command $(cid, P_i, x \leftarrow x_1 \cdot x_2)$, then P_i also loaded x_1 and therefore knows the decryption of $\text{enc}(x_1)$ by the above argument. And if P_i did not load x_1, and the private multiplication command is invalid according to $\mathcal{IO}_{\text{ABB}}$, then P_i terminates and the plaintext of $\text{enc}(x_1)$ is not needed. Notice that by construction of $\mathcal{X}$ we have that $\Pr[\text{IDEAL}^{\curvearrowright,\mathcal{F}_{\text{PRS}}^{\text{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}} = \text{``H} \rightsquigarrow \mathcal{F}_{\text{ZK-PK}}\text{''}] = 0$.

We use the IND-CPA security (Definition 2.9 on page 16) of $\mathcal{THE}$ to prove Eq. 8.4. Consider the following IND-CPA adversary A against $\mathcal{THE}$, see Fig. 2.2 on page 16 for details on the IND-CPA game. The adversary A receives as input $k \in \mathbf{N}$, $z \in \{0,1\}^*$ and a

public-key pk. It then picks test messages 0 and 1 and receives an encryption $Y = E_{pk}(b)$. Then it starts running $\mathrm{IDEAL}^{\curvearrowleft,\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}(k,z)$ with the following modifications:

1. Run on the given public key pk.
2. When $\mathcal{S}_{\mathrm{BABB}}'$, in load and private multiplication is about to compute $\mathrm{enc}(x) = E_{pk}(s_0)$ respectively $E_{pk}(0)$, determine the correct values as in $\mathrm{IDEAL}^{\curvearrowleft,\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}}(k,z)$. Let m denote the correct value and then let $\mathrm{enc}(x) = \mathrm{Blind}_{pk}(m \cdot Y \boxplus s_0 \boxdot (1 \boxminus Y); r_2)$ for a uniformly random randomizer. This change surely affects the distribution of the view of $\mathcal{Z}$.
3. Now that A do not know a witness for $\mathrm{enc}(x)$ anymore, it cannot input it to $\mathcal{F}_{\text{ZK-PK}}^{R_S}$ along with $(pid, (pk, \mathrm{enc}(x)))$. Instead, just input $(pid, (pk, \mathrm{enc}(x)))$ and simulate an output $(pid, 1)$ to all parties. The private multiplication is dealt with in the same way. This does not affect the view of $\mathcal{Z}$.

Notice that of all the modifications only the switch from dummy to correct plaintexts affect the view of $\mathcal{Z}$. When $\mathcal{Z}$ outputs a guess c, output the same guess. Let $H^{b'}$ denote the distribution of c when $b = b'$ in the IND-CPA game. By the definition of IND-CPA security it follows that $H^0 \overset{\mathrm{c}}{\approx} H^1$.[10] But, we have that $Y = E_{pk}(b)$, so $\mathrm{enc}(x)$ is a uniformly random encryption of $mb + s_0(1-b)$. This means that if $b = 0$, then $\mathrm{enc}(x)$ is a uniformly random encryption of s_0 as in $\mathrm{IDEAL}^{\curvearrowleft,\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}(k,z)$, and if $b = 1$, then $\mathrm{enc}(x)$ is a uniformly random encryption of m, as in $\mathrm{IDEAL}^{\curvearrowleft,\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}}(k,z)$. So, $H^0 = \mathrm{IDEAL}^{\curvearrowleft,\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{Y}}(k,z)$ and $H^1 = \mathrm{IDEAL}^{\curvearrowleft,\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}}(k,z)$. So, Eq. 8.4 on the page before follows from $H^0 \overset{\mathrm{c}}{\approx} H^1$.

We now use correct inputs from honest parties, so we got rid of the second and the third "**except**" in Fig. 8.5 on page 276. From Corollary 5.3 on page 186 and the observation that $\Pr[\mathrm{IDEAL}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}} = \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}\text{''}] = 0$ we now get that

$$\mathrm{IDEAL}^{\curvearrowleft,\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PK}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}} \overset{\mathrm{c}}{\approx} \mathrm{IDEAL}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}} \ . \tag{8.5}$$

Using again that $\Pr[\mathrm{IDEAL}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}} = \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}\text{''}] = 0$, we get from Theorem 5.1 on page 180 that

$$\mathrm{IDEAL}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}} \overset{\mathrm{c}}{\approx} \mathrm{HYB}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\text{DAM-ZK}},\mathcal{X}} \ . \tag{8.6}$$

[10] If not we have from $\mathrm{IND}^{\mathtt{cpa},0}_{E,A}(k,z) = H^0(k,z)$ and $\mathrm{IND}^{\mathtt{cpa},1}_{E,A}(k,z) = H^1(k,z)$ that there exists a polynomial $q(k)$ and an auxiliary input $z \in \{0,1\}^*$ such that $|\Pr[\mathrm{IND}^{\mathtt{cpa},0}_{E,A}(k,z) = 1] - \Pr[\mathrm{IND}^{\mathtt{cpa},1}_{E,A}(k,z) = 1]| > \frac{1}{q(k)}$ for infinitely many k. Since A is expected PPT there exists a polynomial $p(k)$ bounding the expected running time of A. Let $A'(k)$ be the adversary which runs $A(k)$ for $q(k)p(k)$ step and outputs as follows: If $A(k)$ outputs within $q(k)p(k)$ steps, then use the output of $A(k)$. Otherwise output a uniformly random bit. Let $r(k)$ be the probability that $A(k)$ do not output in $q(k)p(k)$ steps. Since $r(k) \cdot q(k)p(k) \leq p(k)$ it follows that $r(k) \leq \frac{1}{q(k)}$, so $|\Pr[\mathrm{IND}^{\mathtt{cpa}}_{E,A'}(k,z) = 1] - \Pr[\mathrm{IND}^{\mathtt{cpa}}_{E,A}(k,z) = 1]| = \frac{1}{2}r(k) \leq \frac{1}{2q(k)}$, so by the triangle inequality $|\Pr[\mathrm{IND}^{\mathtt{cpa},0}_{E,A'}(k,z) = 1] - \Pr[\mathrm{IND}^{\mathtt{cpa},1}_{E,A'}(k,z) = 1]| > \frac{1}{2q(k)}$ for infinitely many k, proving that $\mathrm{IND}^{\mathtt{cpa},0}_{E,A'}(k,z) \overset{\mathrm{c}}{\not\approx} \mathrm{IND}^{\mathtt{cpa},1}_{E,A'}(k,z)$. This contradicts Definition 2.9 on page 16 as A' is clearly PPT.

Our last move is to prove that

$$\mathrm{HYB}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\text{DAM-ZK}},\mathcal{X}} \overset{\mathrm{c}}{\approx} \mathrm{HYB}^{\mathcal{F}_{\mathrm{THE}},\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\mathrm{BABB}},\mathcal{Z}} \quad . \tag{8.7}$$

Recall that $\mathcal{X}$ is running $\mathrm{IDEAL}^{\mathcal{F}_{\mathrm{THE}},\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\mathrm{BABB}},\mathcal{S}_{\mathrm{BABB}'}[\cdot],\mathcal{Z}}(k,z)$, but using correct input for honest parties, and when $\mathcal{X}$ is run in $\mathrm{HYB}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\text{DAM-ZK}},\mathcal{X}}$ it is run with the protocol $\pi_{\text{DAM-ZK}}$ instead of the simulator. This means that in $\mathrm{HYB}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\text{DAM-ZK}},\mathcal{X}}$, $\mathcal{S}_{\mathrm{BABB}}$ is actually running the protocol. Outputs to $\mathcal{Z}$ are still delivered by $\mathcal{F}_{\mathrm{BABB}}$ though. Therefore $\mathrm{HYB}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\text{DAM-ZK}},\mathcal{X}}$ is distributed identical to $\mathrm{HYB}^{\mathcal{F}_{\mathrm{THE}},\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\mathrm{BABB}},\mathcal{Z}}$, except that in $\mathrm{HYB}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\text{DAM-ZK}},\mathcal{X}}$, in the output command, the result is $\mathrm{val}(x)$ from the copy of $\mathcal{F}_{\mathrm{BABB}}$ in $\mathrm{IDEAL}^{\mathcal{F}_{\mathrm{THE}},\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\mathrm{BABB}},\mathcal{S}_{\mathrm{BABB}'}[\mathcal{S}_{\text{DAM-ZK}}],\mathcal{Z}}(k,z)$ (in $\mathcal{X}$), and in $\mathrm{HYB}^{\mathcal{F}_{\mathrm{THE}},\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\mathrm{BABB}},\mathcal{Z}}$ the result is the decryption of $\mathrm{enc}(x)$ (this is the mis-simulation anticipated by the fourth "**except**" in Fig. 8.5 on page 276). In case the first "**except**" from Fig. 8.5 on page 276 should apply, the parties of the protocol will output `defined`. whereas $\mathcal{F}_{\mathrm{BABB}}$ outputs nothing.

Therefore, proving that $\mathrm{HYB}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\text{DAM-ZK}},\mathcal{X}} \overset{\mathrm{c}}{\approx} \mathrm{HYB}^{\mathcal{F}_{\mathrm{THE}},\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\mathrm{BABB}},\mathcal{Z}}$ comes down to proving that in $\mathrm{HYB}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\text{DAM-ZK}},\mathcal{X}}$ these two possibilities of error do not occur. Observe first that honest parties always will have their loads and private multiplications accepted, using that $\pi_{\text{DAM-ZK}}$ is correct. And that $\mathrm{val}(x) = D_{sk}(\mathrm{enc}(x))$ for the x's loaded by corrupted parties follows by the fact that $\mathrm{val}(x)$ is computed as $D_{sk}(\mathrm{enc}(x))$, by definition of $\mathcal{X}$. If therefore there exists x such that $\mathrm{val}(x) \neq D_{sk}(\mathrm{enc}(x))$ then there is a first point in the simulation where this occurs and at that point $\mathcal{Z}$ broadcast a ciphertext $\mathrm{enc}(x)$ with $\mathrm{val}(x) \neq D_{sk}(\mathrm{enc}(x))$. This either happens in a load command, where $D_{sk}(\mathrm{enc}(x)) \notin S$ will result in $\mathrm{val}(x) = \perp$, or it happened in a private multiplication command with $D_{sk}(\mathrm{enc}(x)) \neq \mathrm{val}(x) = \mathrm{val}(x_1) \cdot \mathrm{val}(x_2)$. In both cases $\mathcal{Z}$ proved a false instance. In the first case because $\mathcal{Z}$ proved that there exists m and r such that $m \in S$ and $\mathrm{enc}(x) = E_{pk}(m;r)$, which by the perfect correctness implies that $D_{sk}(\mathrm{enc}(x)) \notin S$. In the second case, because we look at the first point where $D_{sk}(\mathrm{enc}(x)) = \mathrm{val}(x)$ fails we have that $\mathrm{val}(x_1) = D_{sk}(\mathrm{enc}(x_1))$ and $\mathrm{val}(x_2) = D_{sk}(\mathrm{enc}(x_2))$, which by $D_{sk}(\mathrm{enc}(x)) \neq \mathrm{val}(x) = \mathrm{val}(x_1) \cdot \mathrm{val}(x_2)$ and the perfect correctness implies that there does not exist at witness for $(pk, \mathrm{enc}(x), \mathrm{enc}(x_1), \mathrm{enc}(x_2)) \in L(R_2)$. So, if Eq. 8.7 does not hold, then with non-negligible probability $\mathcal{Z}$ proves a false instance in $\mathrm{HYB}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\text{DAM-ZK}},\mathcal{X}}$. Because $\mathcal{X}$ knows sk it can detect this: Look at a version of $\mathcal{X}$, let us call it $\mathcal{X}'$, which runs like $\mathcal{X}$ but decrypts all instances successfully proved by $\mathcal{Z}$ and outputs 1 as soon as $\mathcal{Z}$ is detected proving a false instance. Otherwise $\mathcal{X}'$ outputs 0. We have that $\mathrm{HYB}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\text{DAM-ZK}},\mathcal{X}'} \overset{\mathrm{c}}{\not\approx} 0$. By definition of $\mathcal{F}_{\text{ZK-PM}}$ we clearly have that $\mathrm{IDEAL}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}'} \overset{\mathrm{c}}{\approx} 0$. So,

$$\mathrm{IDEAL}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}'} \overset{\mathrm{c}}{\not\approx} \mathrm{HYB}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\pi_{\text{DAM-ZK}},\mathcal{X}'} \quad . \tag{8.8}$$

Using that $\mathcal{X}'$ and $\mathcal{X}$ are identical until $\mathcal{X}'$ outputs it follows from the above observation that $\Pr[\mathrm{IDEAL}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}} = \text{``}\texttt{H} \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}\text{''}] = 0$ that also $\Pr[\mathrm{IDEAL}^{\mathcal{F}_{\mathrm{PRS}}^{\mathrm{commit,prs}}}_{\mathcal{F}_{\text{ZK-PM}},\mathcal{S}_{\text{DAM-ZK}},\mathcal{X}'} =$

$\text{``}\mathtt{H} \rightsquigarrow \mathcal{F}_{\text{ZK-PM}}\text{''}] = 0$. Therefore Eq. 8.8 on the preceding page contradicts Theorem 5.1 on page 180.[11] This concludes the proof of Eq. 8.7 on the preceding page.

It is fairly straight-forward to verify that if $\mathcal{Z}$ lets the ideal functionalities $\mathcal{F}_{\text{THE}}^{\mathcal{THE}}$ and $\mathcal{F}_{\text{PRS}}^{\text{commit},\mathtt{prs}}$ terminate, then each command terminates in constant round. The most essential observation to make is that it is not a problem that $\mathcal{F}_{\text{ZK-PM}}$ is run as a proof of knowledge. The non-overlapping IO behavior of $\mathcal{IO}_{\text{ABB}}$ will prevent the situation where new proofs keep arriving (a situation which would prevent $\mathcal{F}_{\text{ZK-PM}}$ from terminating). □

8.5 What is New?

We have showed that it is possible to do general secure MPC by broadcasting encryptions of the inputs under a threshold homomorphic encryption (THE) scheme and then computing on these encryptions, while keeping a consistent view of this *encrypted state* of the computation. This is an essentially different approach to MPC than the secret sharing approach, where the parties compute on a *shared state*. In the THE approach only the key is shared. Conceptually, this is of course interesting in its own right, as the techniques are essentially different. But, does it buy us anything to compute on an encrypted state instead of a shared state, in terms of efficiency say? The discussion of the efficiency of our realization is deferred to Chapter 10, but here we mention one example where computing on an encrypted state buys us at lot in terms of efficiency. If we are satisfied with passive security, we can obtain a protocol with communication complexity $O(nk)$ bits per operation.

In fact, we only need that we are given one party, P_S say, which is only passively corrupted. We can then simply let P_S, drive the computation. When a party loads a values, instead of broadcasting $\text{enc}(x)$ it sends it to P_S and proves plaintext knowledge. If the proof is accepted, then P_S sends $\text{enc}(x)$ to all parties. Private multiplication is done in the same way and the linear combination is done as in the current protocol. If the decryption protocol is of the form in Fig. 6.3 on page 195, then output can be realized by P_S sending $\text{enc}(x)$ to all parties which respond by sending their decryption shares to P_S along with proofs of correctness. Then P_S combines and sends the result to the parties. This realizes $\mathcal{F}_{\text{BABB}}$ if P_S follows the protocol. Adding robustness to this more efficiently than by running it over a broadcast channel, which is basically what π_{BABB} does, is the subject of current research.

The communication complexity $O(nk)$ is a factor n better than what it seems any approach based on a secret sharing scheme can obtain — collecting the shares at one party would completely jeopardize the security.

[11] Notice that the definition of $\mathcal{X}'$ uses sk. It is therefore essential to the proof that the special membership soundness of Σ_1, Σ_S and Σ_2 is not relative to the PPT family of sets defined by a random public key.

CHAPTER 9

Universally Composable Commitment Schemes

The follies which a man regrets the most in his life are those which he didn't commit when he had the opportunity.
— Helen Rowland

9.1 Introduction

In this section we show how to realize the commitment functionality efficiently based on various standard complexity assumptions. The notion of a universally composable commitment scheme was first published, and realized, by Canetti and Fischlin [CF01]. The notion is very strong: It guarantees that security is maintained even when an unbounded number of copies of the scheme are running concurrently and asynchronous. It also guarantees non-malleability [DDN00, DCIO98, DG02], independence of inputs and maintains security even if an adversary can decide adaptively to corrupt some of the players and make them cheat.

9.1.1 Related Work

It is clearly important for practical applications to have solutions where only the two main players need to be active. However, in [CF01] it is proved that universal composability is so strong a notion that no universally composable commitment scheme for only two players exist. However, if one assumes the common reference model, then two-player solutions do exist and two examples are given in [CF01] based on specific complexity theoretic assumptions. Later [CLOS02] gave a solution based only on the existence of trapdoor permutations.

The commitment scheme from [CF01, CLOS02] use $\Omega(k)$ bits to commit to one bit, where k is a security parameter, and they guarantee only computational hiding and binding. In fact, as detailed later, one might even get the impression that perfect hiding, respectively binding cannot be achieved. Here, by perfect, we mean that an unbounded receiver gets zero information about m, respectively an unbounded committer can change his mind about m with probability zero.

Our contribution is a new construction of universally composable commitment schemes, which uses $O(k)$ bits of communication to commit to k bits. The scheme can be set up such that it is perfectly binding, or perfectly hiding, without loosing efficiency.[1] The construction is based on a new primitive which we call a mixed commitment scheme, which we can in turn construct from the N-invertible homomorphisms from Section 2.10. Our commitment protocol has three moves, where the protocols in [CF01, CLOS02] are non-interactive. So, with our current knowledge about universally composable commitments, the price for a low communication complexity in bits is a higher round complexity.

In addition to assuming the common reference string we use the PRS model, which we introduced in Section 3.8.8. I.e. we assume that the parties have a mechanism for learning the other parties' public keys and that the owner of the public key is guaranteed to know the corresponding private key. Opposed to the CRS model, no guarantee is given on the *distribution* of the public key. Alternatively, it can be done without the PRS model, but then the CRS will be of length $O(nk)$, as discussed in Section 3.8.8.

As a final contribution we show that if a mixed commitment scheme comes with non-erasure Σ-protocols for proving relations among committed values, the resulting UC commitment scheme 'inherits' these protocols, such that usage of these is also universally composable. For our concrete schemes, this results in efficient protocols for proving binary Boolean relations among committed values and also (for the version based on Paillier encryption) additive and multiplicative relations modulo N.

Our commitment scheme commits to k bits, which also means that when the commitment is opened, all k bits are reveal. This in some applications makes it impossible to exploit the fact that the expansion factor is constant for efficiency. Some protocols requires that individual bits can be opened independently. Very prominent examples are many zero-knowledge proof protocols, e.g. the zero-knowledge proof protocol for Hamiltonian-Cycle (HC) given and proved secure in the $\mathcal{F}_{\text{COM}}$-hybrid model in [CF01]. This problem can however be solved using a standard technique, presented by Kilian, Micali and Ostrovsky [KMO89], for transforming a multi-bit commitment scheme into a multi-bit commitment scheme with the property that individual bits can be opened independently, without any loss of efficiency, except for a small constant factor. This allow us to improve the communication complexity of the HC protocol in [CF01] by a factor k.

9.1.2 Informal Description of the Main Ideas

Recall that in a standard commitment scheme, both committing and opening are non-interactive, so that committing just consists of running an algorithm commit_K, keyed by a public key K, taking as input the message m to be committed to and a uniformly random string r. The committer computes $c \leftarrow \text{commit}_K(m;r)$ and sends c to the receiver. To open, the committer sends m and r to the receiver, who checks that $c = \text{commit}_K(m;r)$. For this type of scheme, hiding means that given just c the receiver does not learn m and binding means that the committer cannot change his mind by computing m', r', where $c = \text{commit}(m';r')$ and $m' \neq m$.

[1] [CF01] also contains a scheme which is statistically binding and computationally hiding, the scheme however requires a new setup of the common reference string per commitment.

In a trapdoor scheme however, to each public key K a piece of trapdoor information t_K is associated which, if known, allows the committer to change his mind. We call such schemes equivocable. One may also construct schemes where a different type of trapdoor information d_K exists, such that given d_K, one can efficiently compute m from $\text{commit}_K(m;r)$. We call such schemes extractable. Note that equivocable schemes cannot be perfect binding and that extractable schemes cannot be perfect hiding.

As mentioned, the scheme in [CF01] guarantees only computational binding and computational hiding. Actually, this is important to the construction: To prove security, we must simulate an adversary's view of the real scheme with access to the idealized functionality only. Now, if the committer is corrupted by the adversary and sends a commitment c, the simulator must find out which message was committed to, and send it to the idealized functionality. As discussed several times we cannot rewind in the UC framework. So, we cannot use rewinding for extracting the message. A solution is to use an extractable scheme, have the public key K in the reference string, and set things up such that the simulator knows the trapdoor d_k. A similar consideration leads to the conclusion that if instead the receiver is corrupt, the scheme must be equivocable with trapdoor t_K known to the simulator, because the simulator must generate a commitment on behalf of the honest committer before finding out from the idealized functionality which value was actually committed to. So, to build universally composable commitments it seems we must have a scheme that is simultaneously extractable *and* equivocable, and all without rewinding. This is precisely what Canetti's and Fischlin's ingenious construction provides.

In this chapter, we propose a different technique for universally composable commitments based on what we call a mixed commitment scheme. A mixed commitment scheme is basically a commitment scheme which on some of the keys is perfectly hiding and equivocable, we call these keys the E-keys, and on some of the keys is perfectly binding and extractable, we call these keys the X-keys. Clearly, no key can be both an X- and an E-key, so if we were to put the entire key in the common reference string, either extractability or equivocability would fail and the simulation could not work. We remedy this by putting only a part of the key, the so-called system key, in the reference string. The rest of the key is set up once per commitment using a two-move coin-flip protocol. This allows the simulator to force the key used for each commitment to be an E-key or an X-key depending on whether equivocability or extractability is needed.

Our basic construction is neither perfectly binding nor perfectly hiding because the set-up of keys is randomized and is not guaranteed to lead to any particular type of key. However, one may add to the reference string an extra key that is guaranteed to be either an E-key or an X-key. Using this in combination with the basic scheme, one can obtain either perfect hiding or perfect binding.

9.2 Mixed Commitment Schemes

We now give a more formal description of mixed commitment schemes. The most important difference from the intuitive discussion above is that the system key N comes with a trapdoor t_N that allows efficient extraction for all X-keys. The E-keys, however, each come with their own trapdoor for equivocability.

Definition 9.1 *By a mixed commitment scheme we mean a commitment scheme* commit_K *with some global system key* N*, which determines the message space* $\mathcal{M}_N$ *and the key space* $\mathcal{K}_N$ *of the commitments. The key space contains two sets, the E-keys and the X-keys, for which the following holds:*

key generation:

One can in PPT generate a system key N*, defining a message space* $\mathcal{M}_N$*, along with the so-called X-trapdoor* t_N*. One can given the system key* N *generate random keys from* $\mathcal{K}_N$ *in PPT and given* t_N *one can sample random X-keys in PPT. Given the system key, one can in PPT generate an E-key* K *along with the so-called E-trapdoor* t_K*.*

key indistinguishability:

Random E-keys and random X-keys are both computationally indistinguishable from random keys from $\mathcal{K}_N$ *as long as the X-trapdoor* t_N *is not known.*

equivocability:

Given E-key K *and E-trapdoor* t_K *one can given any commitment* $c = \text{commit}_K(m; r)$*, along with* m *and* r*, and any message* $m' \in \mathcal{M}_N$ *compute uniformly random* r' *for which* $c = \text{commit}_K(m'; r')$*.*

extraction:

Given a commitment $c = \text{commit}_K(m; r)$*, where* K *is an X-key, one can given the X-trapdoor* t_N *compute* m *in PPT.*

We furthermore require that given N *one can sample random elements from* $\mathcal{M}_N$ *and recognize* $\mathcal{M}_N$ *and* $\mathcal{K}_N$ *in PPT.*

Note that the indistinguishability of random E-keys, random X-keys, and random keys from $\mathcal{K}_N$ implies that as long as the X-trapdoor is not known the scheme is computationally hiding for all keys and as long as the E-trapdoor is not known either the scheme is computationally binding for all keys.

For the construction in the next section we need a few special requirements on the mixed commitment scheme. First of all we assume that the message space $\mathcal{M}_N$ and the key space $\mathcal{K}_N$ are finite groups in which we can compute in PPT. We denote the group operation by $+$. Second we need that the number of X-keys over the total number of keys is negligibly close to 1. Finally we need that the key space $\mathcal{K}_N$ can be sampled in PPTIS, a requirement derived from the fact that we are considering adaptive security. We call a mixed commitment scheme with these properties a special mixed commitment scheme.

We say that the commitment scheme has a Σ-protocol for a given $(a+1)$-ary relation R if $(m_1, \ldots, m_a, N) \in R$ can be checked given $(m_1, \ldots, m_a, N) \in (\{0,1\}^*)^{a+1}$ and if there exists a non-erasure Σ-protocol for the relation with instances $x = (N, L, c_1, \ldots, c_a)$ and witnesses $w = ((m_1, r_1), \ldots, (m_a, r_a))$ given by N being a system key, L being a commitment key and $c_i = \text{commit}_L(m_i; r_i)$ for $i = 1, \ldots, a$ and $(m_1, \ldots, m_l) \in R$.[2] To say that the commitment scheme has a Σ-protocol for any relation at all we also require that there exists

[2]It is not a mistake that we did not require that $m_i \in \mathcal{M}_N$. If this is guaranteed by $c_i = \text{commit}_K(m_i; r)$, that is fine, otherwise one can add the requirement to R if one needs it.

a non-erasure Σ-protocol for the relation with instances $x = (N, L, c_L, K, c_K)$ and witnesses $w = ((m_L, r_L), (m_K, r_K))$ given by N being a system key, L and K being commitment keys and $c_L = \text{commit}_L(m_L; r_L)$ and $c_K = \text{commit}_K(m_K; r_K)$ and $m_L = m_K$. We call this relation equality of committed values.

9.2.1 Examples of Special Mixed Commitment Schemes

We give three examples of special mixed commitment schemes, all based on N-invertible homomorphisms which can encrypt with cosets. Specifically, we can construct mixed commitment schemes from the examples in Definition 2.31 on page 58, excluding the exponentiation example.

Let $(G, H, f, N, b', g, b, t) \leftarrow gen(k)$ be an N-invertible homomorphisms which can encrypt with cosets. The system key is $N = (G, H, f, N, b', g, b)$ and $t_N = t$. The key space is $\mathcal{K}_N = H$ and the message space is $\mathcal{M}_N = \mathbf{Z}_b$; Both can be sampled in PPT as required by Definition 9.1 on the preceding page. We need that H has invertible sampling, as Definition 9.1 on the facing page requires that the key space has invertible sampling. That this is the case for the Paillier and the Okamoto-Uchiyama examples from Definition 2.31 on page 58 is trivial. For the Diffie-Hellman example we need that $\langle \alpha \rangle$ can we invertibly sampled.

If we let p be a random k-bit prime for which $q = (p-1)/2$ is also prime and let $\alpha = 4$ be a generator of the quadratic residues, then a random element from $\langle \alpha \rangle$ can be sampled as $x = y^2 \bmod p$ for uniformly random $y \in \mathbf{Z}_p^*$. That this is an invertible sampling follows from the fact that square roots can be computed in Q_p in PPT.

We commit as $\text{commit}_K(m, r) = K^m f(r)$, where r is uniformly random in G — actually r is the uniformly random bits used to sample an element from G, but since G has invertible sampling we will not distinguish.

The E-keys will be the set $F = f(G)$ and the E-trapdoor will be $f^{-1}(K)$. Clearly a random E-key can be sampled along with its trapdoor in PPT as require, namely as $K = f(t_K)$ for uniformly random $t_K \in G$; Since f is a homomorphism is follows that K is uniformly random in F.

For equivocability assume that we have $K = f(r_K)$ and $c = K^m f(r)$ and that we are given $m' \in \mathbf{Z}_b$. Then compute $r' = r_K^{m-m'} r$ and r' is a uniformly random element from G and $\text{commit}_K(m'; r') = K^{m'} f(r_K^{m-m'} r) = K^{m'} f(r_K)^{m-m'} f(r) = K^{m'} K^{m-m'} f(r) = K^m f(r) = c$, as desired — again, formally we should use the random bits faking algorithm for the sampler of G to construct random bits making it look like r' was sampled from G.

If the group homomorphism has trapdoor coset determination, then the X-keys will be the keys of the form $K = g^i f(r)$, where i is invertible modulo $\text{ord}_{H/F}(g)$. We need that this is the entire key space except for a negligible fraction. That this is indeed the case for the Paillier and the Okamoto-Uchiyama examples in Definition 2.31 on page 58 is straight-forward to verify. Definition 9.1 on the preceding page requires that one can sample X-keys in PPT when $t_N = t$ is known, that this is possible follows from the fact that $\text{ord}_{H/F}(g) = |H/F|$ is computable from N and t_N and that $\mathbf{Z}^*_{\text{ord}_{H/F}(g)}$ can be sampled in PPT by Theorem 2.3 on page 28. To see that $|H/F|$ is computable from N and t_N, observe that N-invertibility allows us to compute $x' \in G$ such that $f(x') = g^N$. Therefore the order of g divides N. So, use the trapdoor coset determination to determine the coset of g^{-1} to obtain $\text{ord}_{H/F}(g) - 1$.

Using trapdoor coset determination we extract a commitment $c = K^m f(s) = (g^i f(r))^m f(s)$ as follows: From c compute $im \bmod \mathrm{ord}(g)$ and from K compute $i \bmod \mathrm{ord}(g)$. Since i is invertible modulo $\mathrm{ord}(g)$ we can then compute $m \bmod \mathrm{ord}(g) = m$.

If the group homomorphism has trapdoor image distinguishability, then the X-keys will be $H \setminus F$. For this to work we need that the size of H/F is super-polynomial. Notice that this is indeed the case for the Diffie-Hellman example in Definition 2.31 on page 58. By the trapdoor image distinguishability the X-keys can be efficiently sampled given t_N. For extraction, note that $\mathrm{commit}_K(0;r) \in F$ and $\mathrm{commit}_K(1;r) \notin F$ and use the trapdoor image distinguishability.

9.2.1.1 Proofs of Relation

We now construct some proofs of relation for our examples of mixed commitment schemes. Since our examples are all based on N-invertible homomorphisms we can use the proofs from Section 2.10.4.

A commitment is of the form $c = K^m f(r)$, i.e. the witness is a K representation, which immediately gives us efficient Σ-protocols for proving linear and multiplicative relations between committed values. For the Paillier example, where the message space is also $\mathbf{Z}_N$, this is fine, as we can then e.g. prove knowledge of committed values $m_1, \ldots, m_l \in \mathbf{Z}_N$ in a given linear relation $\sum_{i=1}^{l} a_i m_i \equiv a_0 \pmod{N}$. This in particular allows to prove that $m_1 = m_2$, which was required to say that the commitment scheme has any proof of relation at all.

It is however less obvious what use we have of proving relation modulo N in the Okamoto-Uchiyama example and the Diffie-Hellman example, where $N = p^2 q$ respectively $N = q$, but where the message space is $\mathbf{Z}_{2^{\lceil k/2 \rceil - 1}}$ respectively small subsets of $\mathbf{Z}_q$. We have however not required that the proofs of relations also prove that the message of which one proves knowledge is from the message space, so it is at least sound to use these proofs. And, there is one important application of the proof of linear relation, namely with $a_1 = 1$, $a_2 = N - 1$ and $a_0 = 0$, which proves knowledge of $m_1, m_2 \in \mathbf{Z}_N$ for which $m_0 = m_1$. This gives us the proof that we need to say that the commitment scheme has any proof of relation at all.

We can also construct proofs for small Boolean relations. We can do use the proof of linear relation with $a_1 = 1$ and $a_0 = 0$ or $a_1 = 1$. This allows to prove knowledge that a committed value is 0 and to prove knowledge that a committed value is 1. Using the $\wedge$-construction from Section 2.9 we can then for l commitments $(c_1, \ldots, c_l)$ prove knowledge of $(m_1, \ldots, m_l)$ for which $(m_1, \ldots, m_l) = (b_1, \ldots, b_l)$ for some constant $(b_1, \ldots, b_l) \in \{0,1\}^l$. Using the $\vee$-construction this then allows to prove knowledge of $(m_1, \ldots, m_l) \in \{0,1\}^l$ for which $(m_1, \ldots, m_l) \in R$ for any relation $R \subseteq \{0,1\}^l$. In particular, we can prove $a = b \wedge c$ for three committed values, by using $R = \{(0,0,0), (0,0,1), (0,1,0), (1,1,1)\}$ and $a = \neg b$, using $R = \{(0,1), (1,0)\}$.

9.3 The Commitment Functionality

We now specify the task that we want to realize as an ideal functionality. We look at a slightly different version of the commitment functionality than the one in [CF01]. The functionality in

Functionality $\mathcal{F}_{\text{COM-PR}}^{\texttt{com}}$

The functionality runs with parties $P_1, \ldots, P_n$ and is parameterized by a special mixed commitment scheme `com`. It proceeds as follows:

initialize:

On input `init` from all parties, generate a uniformly random system key N for `com` along with the X-trapdoor t_N and output (N, t_N) on the SOT; We assume that t_N contains all random bits used by the generator to guarantee that the functionality is well-formed. Then let the adversary specify an output round, and output N to all parties in the specified round.

commit:

Upon receiving $(\texttt{commit}, cid, P_i, P_j, m)$ from P_i, where $m \in \mathcal{M}_N$, record (cid, P_i, P_j, m) and output $(\texttt{commit}, cid, P_i, P_j)$ on the SOT, let the adversary specify an output round and, in the specified round, output $(\texttt{receipt}, cid, P_i, P_j)$ to P_j. If P_i is corrupted before the output round and the SIT contains a value $(\texttt{fail}, cid)$, then do not output anything to P_j and record nothing. If P_i is corrupted before the output round and the value $(\texttt{change}, cid, m')$ is on the SIT, then record (cid, P_i, P_j, m').

opening:

Upon receiving a message $(\texttt{open}, cid, P_i, P_j)$ from P_i, where (cid, P_i, P_j, m) has been recorded, output (cid, P_i, P_j, m) on the SOT, let the adversary specify an output round and, in the specified round output $(\texttt{open}, cid, P_i, P_j, m)$ to P_j. If before the output round P_i is corrupted and the SIT contains a value $(\texttt{fail}, cid)$, then output nothing to P_j.

proof of relation:

If P_j is honest and inputs $(\texttt{prove}, cid, P_i, P_j, R, cid_1, \ldots, cid_a)$ and P_i inputs the same value in the same round or P_i is corrupted, and if furthermore $(cid_1, P_i, P_j, m_1), \ldots, (cid_a, P_i, P_j, m_a)$ have been recorded and R is an $(a+1)$-ary relation of the special mixed commitment scheme, then output $(\texttt{prove}, cid, P_i, P_j, R, cid_1, \ldots, cid_a, b)$ on the SOT, where $b \in \{0, 1\}$ and $b = 1$ iff $(N, m_1, m_2, \ldots, m_a) \in R$. Then let the adversary specify an output round, and in the specified round output (cid, b) to P_j. If before the output round P_i is corrupted and the SIT contains a value $(\texttt{fail}, cid)$, then output $(cid, 0)$ to P_j.

incorrect inputs:

If an honest party uses the same commitment id cid for committing or proving more than once or some honest party inputs $m \notin \mathcal{M}_N$ or some honest party inputs $(id, P_i, P_j, \ldots)$, in proof of relation or opening, for which no values are recorded or where the corresponding party is honest and did not input the corresponding value, then break down.

Figure 9.1: The functionality for COMmitment with Proof of Relation.

[CF01] is only for committing to one bit. Here we generalize. The domain of our commitments will be the domain of the special mixed commitment used in the implementation. Therefore the ideal functionality must specify the domain by initially giving a system key N. In addition, the X-trapdoor of N is revealed to the simulator. This is no problem in the ideal process since here the X-trapdoor cannot be used to find committed values — the ideal functionality stores committed values internally and reveals nothing before opening

time. The simulator, however, needs the X-trapdoor in order to do the simulation of our implementation — for pushing some rewinding to the analysis.[3] The implementation on the other hand will of course keep the X-trapdoor of N hidden. An additional extension of the functionality is the addition of the proof of relation command, which allows the committer to prove relations between committed values. A similar functionality was independently presented by Canetti, Lindell, Ostrovsky and Sahai [CLOS02]. They consider a version where the proof of relation takes as input a constant known to both the committer and the receiver. This allows them to do away with the opening command, as it can be realized by a proof of relation (with the relation being the identity relation and the known constant being the value committed to). One could consider simplifying $\mathcal{F}_{\text{COM-PR}}$ using this idea. However, since the opening command can be realized non-interactively and our realization of the proof of relation is interactive, it would make a difference in efficiency which solution is used to reveal a committed value. We have therefore chosen to maintain both commands. The ideal functionality for commitments with proofs of relation is given in Fig. 9.1 on the page before.

It should be noted that a version of the functionality where N and t_N are not specified by the ideal functionality could be used. We would then let the domain of the commitments be a domain determined by k and which is contained in (or easy to encode in) the domain of all the system keys for security parameter k. In the description it is not a mistake that we do not require $m' \in \mathcal{M}_N$ when the environment inputs on behalf of corrupted parties in the `change` command. The adversary can commit to any value. He will of course be caught when he opens the commitment; If the application needs that any opened commitment opens to a value in $\mathcal{M}_N$, then the parties can define any value outside $\mathcal{M}_N$ to be some fixed value from $\mathcal{M}_N$ or define that an opening to a value outside $\mathcal{M}_N$ does not count as a correct opening.

9.4 UCC with Constant Expansion Factor

Given a special mixed commitment scheme we can construct a universally composable commitment scheme. The construction is given in Fig. 9.2 on the facing page. It is a simplified version of a protocol first published in [DN02b]. The main simplifying difference is that in [DN02b] the special mixed commitment scheme was transformed as to equip it with two independent trapdoors before it was used to construct the universally composable commitment scheme. Without having to go into details, this is avoided in the current version by sending the c_L commitment in the initial round. This commitment has however not been added for that purpose. The reason for adding it is to bind the committing party from the first message. Without the c_L commitment we do not know how to prove the protocol secure. We discuss this during the analysis when we use that c_L is there (see Footnote *a* in Fig. 9.3(c) on page 294). The initial commitment was not part of the protocol in [DN02b], and as a consequence we do *not* know how to prove the protocol in [DN02b] secure, opposed to what is claimed in that paper. The flaw in the proof of [DN02b] was pointed to by Groth [Gro02].

[3] See Remark 8.1 on page 258 for a more detailed discussion of this type of functionality.

Protocol π_{COMPR}

The protocol runs with parties $P_1, \ldots, P_n$ in the $(\mathcal{F}_{\text{CRS}}^{N}, \mathcal{F}_{\text{ZK-PM}}, \mathcal{F}_{\text{PRS}}^{gen, R_{gen}})$-hybrid model, where $\mathcal{F}_{\text{CRS}}^{N}$ generates a random system key N along with a random key L, and gen is the key-generator for a trapdoor commitment scheme.[a]

initialization:

On input `init`, input `init` to $\mathcal{F}_{\text{CRS}}$ and $\mathcal{F}_{\text{ZK-PM}}$ and request the key of all parties from $\mathcal{F}_{\text{PRS}}$. Then wait for a value (N, L) from $\mathcal{F}_{\text{CRS}}$, the value `ready` from $\mathcal{F}_{\text{ZK-PK}}$ and a value $\overline{K}_i$ for each P_i from $\mathcal{F}_{\text{PRS}}$. Then output `ready`. Until outputting `ready`, ignore all inputs but the first `init` input.

committing:

C.1 On input $(\texttt{commit}, cid, P_i, P_j, m)$ party P_i generates a random commitment key K_1 for system key N and commits to it as $c_1 = \text{commit}_{\overline{K}_j}(K_1; r_1)$. Then P_i computes $c_L = \text{commit}_L(m; r_L)$ and sends $(\texttt{com}_1, cid, c_1, c_L)$ to P_j.

R.1 P_j replies with $(\texttt{com}_2, cid, K_2)$ for a random commitment key K_2.

C.2 P_i computes $K = K_1 + K_2$ and $c_2 = \text{commit}_K(m; r_2)$, records $(cid, P_j, K, m, r_2, r_L)$ and sends the message $(\texttt{com}_3, cid, K_1, r_1, c_2)$ to P_j.

R.2 P_j checks that $c_1 = \text{commit}_{\overline{K}_j}(K_1; r_1)$, and if so computes $K = K_1 + K_2$, records (cid, P_i, K, c_2, c_L), and outputs $(\texttt{receipt}, cid, P_i, P_j)$.

opening:

C.3 On input $(\texttt{open}, cid, P_i, P_j)$, P_i sends $(\texttt{open}, cid, m, r_2, r_L)$ to P_j.

R.3 P_j checks that $c_2 = \text{commit}_K(m; r_2)$ and $c_L = \text{commit}_L(m; r_2)$, and if so outputs $(\texttt{open}, cid, P_i, P_j, m)$.

proving relation:

On input $(\texttt{prove}, cid, P_i, P_j, R, cid_1, \ldots, cid_a)$, where $(cid_1, P_j, K_1, m_1, r_1, r_{L,1}), \ldots, (cid_a, P_j, K_a, m_a, r_a, r_{L,a})$ are recorded commitments, if $(m_1, \ldots, m_a) \in R$, party P_i inputs (cid, P_i, P_j, R, x, w) to $\mathcal{F}_{\text{ZK-PM}}^{R}$, where $x = (N, L, \text{commit}_L(m_1; r_{L,1}), \ldots, \text{commit}_L(m_a; r_{L,a}))$ and $w = (m_1, r_{L,1}, \ldots, m_a, r_{L,a})$.

If party P_j receives input $(\texttt{prove}, cid, P_i, P_j, R, cid_1, \ldots, cid_a)$, where for $l = 1, \ldots, a$ a commitment $(cid_l, P_i, K_l, c_{2,l}, c_{L,l})$ was stored, then P_j inputs (cid, P_i, P_j, R, x) to $\mathcal{F}_{\text{ZK-PM}}$, where $x = (N, L, c_{L,1}, \ldots, c_{L,a})$. Then P_j waits for output (cid, b) from $\mathcal{F}_{\text{ZK-PM}}^{R}$ and outputs (cid, b).

The first time a given commitment (cid, P_i, K, c_2, c_L) is used in any proof it is also proved that (L, c_L) and (K, c_2) commits to the same value.

[a] This could be the mixed commitment scheme, but any trapdoor commitment scheme will do.

Figure 9.2: The protocol for COMmitment with Proof of Relation.

Theorem 9.1 *If com is a special mixed commitment scheme, then $\pi_{\text{COMPR}}^{\mathsf{com}}[\pi_{\text{DAM-ZK}}^{R} / \mathcal{F}_{\text{ZK-PM}}^{R}]$ realizes $\mathcal{F}_{\text{COM-PR}}^{\mathsf{com}}$ in the $(\mathcal{F}_{\text{CRS}}^{N}, \mathcal{F}_{\text{PRS}}^{\text{commit}})$-hybrid model.*

Proof. We do the proof using the simulator $\mathcal{S}_{\text{COM-PR}}$ in Fig. 9.3 on the following page. We would now like to argue that

$$\Pr[\text{IDEAL}^{\mathcal{F}_{\text{CRS}}^{\mathsf{com}}, \mathcal{F}_{\text{PRS}}^{\text{commit}}}_{\mathcal{F}_{\text{COM-PR}}^{\mathsf{com}}, \mathcal{S}_{\text{COM-PR}}, \mathcal{Z}} \in \text{``H} \rightsquigarrow \mathcal{F}_{\text{CRS}}\text{''} \cup \text{``H} \rightsquigarrow \mathcal{F}_{\text{PRS}}\text{''}] \overset{\text{c}}{\approx} 0$$

Interface $\mathcal{S}_{\text{COM-PR}}$

The interface $\mathcal{S}_{\text{COM-PR}}$ tries to run internally a copy of π_{COMPR} on the inputs of $\mathcal{Z}$. For the inputs not known to $\mathcal{S}_{\text{COM-PR}}$ this is done in a non-committing way, so that all commitments can later be opened to arbitrary values. The communication of $\pi^R_{\text{DAM-ZK}}$ is simulated using ${\mathcal{S}_{\text{DAM-ZK}}}^R$. The individual commands are handled as follows:

initializing:

Simulate the value (N, L), by obtaining the system key N (along with its X-trapdoor t_N) from the ideal functionality $\mathcal{F}^{\mathtt{com}}_{\text{COM-PR}}$ and generating L as a random E-key with known E-trapdoor t_L. The keys $\overline{K}_i$ are generated correctly as E-keys, learning their trapdoors $\overline{t}_i$.

committing:

On input $(\mathtt{commit}, cid, P_i, P_j)$ from $\mathcal{F}_{\text{COM-PR}}$, where P_i is honest, we know that $\mathcal{Z}$ has given P_i input $(\mathtt{commit}, cid, P_i, P_j, m)$ on $\mathcal{F}_{\text{COM-PR}}$ for some $m \in \mathcal{M}_N$. We have to simulate P_i's behavior on the input $(\mathtt{commit}, cid, P_i, P_j, m)$ without knowing m.

To facilitate this, generate K as a uniformly random E-key with known E-trapdoor t_K; We call K the hitting key and we will make K the key used by P_i, which then allows to open the commitment to m, if m is later learned. We do this as follows:

C.1 Let $c_1 = \text{commit}_{\overline{K}_j}(K'_1; r'_1)$ for a random commitment key K'_1 and let $c_L = \text{commit}_L(m'; r'_L)$ for some dummy value m'. Then proceed as in the protocol.
If P_i is later corrupted, then let $K_1 = K'_1$ and let $r_1 = r'_1$. And, if $m' = m$, then let $r_L = r'_L$; Otherwise, use t_L (the E-trapdoor of L) to compute r_L for which $c_L = \text{commit}_L(m; r_L)$, where m is the correct value of m', learned from $\mathcal{F}_{\text{COM-PR}}$.

C.2 On a message $(\mathtt{com}_2, cid, K_2)$ from P_j, if $K_2 = K - K'_1$, where K is the hitting key, then let $K_1 = K'_1$ and let $r_1 = r'_1$. If not, let $K_1 = K - K_2$ and use $\overline{t}_j$ to compute r_1 such that $c_1 = \text{commit}_{\overline{K}_j}(K_1; r_1)$. Then use that $\mathcal{K}_N$ can be sampled in PPTIS to make it look as if K_1 was sampled uniformly at random from $\mathcal{K}_N$. In both cases we now have that $K = K_1 + K_2$, where K is the hitting key, and $c_1 = \text{commit}_{\overline{K}_j}(K_1; r_1)$. Finally, let $c_2 = \text{commit}_K(m''; r'_2)$ for some second dummy value[a] m'' and send $(\mathtt{com}_3, cid, K_1, r_1, c_2)$ to P_j.
If P_i is corrupted after this step and before Step C.3, then we learn m. If $m' = m$, let $r_2 = r'_2$, otherwise, use t_K (the E-trapdoor of the hitting key) to compute r_2 for which $c_2 = \text{commit}_K(m; r_2)$.

[a] The reason for introducing symbols for the dummy value and this second dummy value is only for easy reference in the following analysis, they could both have been 0.

Figure 9.3(a) (cont. in Fig. 9.3(b) on the facing page): The interface $\mathcal{S}_{\text{COM-PR}}$ used in the proof of Theorem 9.1 on the page before

and that

$$\text{HYB}^{\mathcal{F}^{\mathtt{com}}_{\text{CRS}}, \mathcal{F}^{\text{commit}}_{\text{PRS}}}_{\pi^{\mathtt{com}}_{\text{COMPR}}[\pi^R_{\text{DAM-ZK}}/\mathcal{F}^R_{\text{ZK-PM}}], \mathcal{Z}} \overset{\text{c}}{\approx} \text{IDEAL}^{\mathcal{F}^{\mathtt{com}}_{\text{CRS}}, \mathcal{F}^{\text{commit}}_{\text{PRS}}}_{\mathcal{F}^{\mathtt{com}}_{\text{COM-PR}}, \mathcal{S}_{\text{COM-PR}}, \mathcal{Z}}}$$

for all $\mathcal{Z}$ where

$$\Pr[\text{IDEAL}^{\mathcal{F}^{\mathtt{com}}_{\text{CRS}}, \mathcal{F}^{\text{commit}}_{\text{PRS}}}_{\mathcal{F}^{\mathtt{com}}_{\text{COM-PR}}, \mathcal{S}_{\text{COM-PR}}, \mathcal{Z}} = \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}^{\mathtt{com}}_{\text{COM-PR}}\text{''}] \overset{\text{c}}{\approx} 0 \ .$$

Interface $\mathcal{S}_{\text{COM-PR}}$

opening (C.3):

On input $(\texttt{open}, cid, P_i, P_j, m)$ from $\mathcal{F}_{\text{COM-PR}}$, where P_i is honest, we know that $\mathcal{Z}$ has given P_i input $(\texttt{open}, cid, P_i, P_j)$ on $\mathcal{F}_{\text{COM-PR}}$. To simulate this construct r_2 and r_L as specified in step C.2 and send $(\texttt{open}, cid, m, r_2, r_L)$ to P_j.

receiving a commitment:

This is how to simulate an honest receiver P_j receiving a commitment.

R.1 If P_i is corrupt, then generate K_2 as in the protocol. Otherwise, let $K_2 = K - K_1$ and us that $\mathcal{K}_N$ can be sampled in PPTIS to make it look as if K_1 was sampled uniformly at random from $\mathcal{K}_N$.

R.2 On receiving message $(\texttt{com}_3, cid, K_1, r_1, c_2, c_L)$ from P_i (and having received $(\texttt{receipt}, cid, P_i, P_j)$ from $\mathcal{F}_{\text{COM-PR}}$ two rounds ago), continue as follows: If $c_1 \neq \text{commit}_{\overline{K}_j}(K_1; r_1)$, then P_i is corrupted and we can input $(\texttt{fail}, cid)$ on the SIT, which makes $\mathcal{F}_{\text{COM-PR}}$ not act further on the commitment with id cid. If $c_1 = \text{commit}_{\overline{K}_j}(K_1; r_1)$, we have to make P_j output the value $(\texttt{receipt}, cid, P_i, P_j)$ on $\mathcal{F}_{\text{COM-PR}}$ and make $\mathcal{F}_{\text{COM-PR}}$ commit to an appropriate value under the id cid.

(a) If the $\texttt{com}_1$-message with id cid received by P_j was sent by $\mathcal{S}_{\text{COM-PR}}$, then it was sent because $\mathcal{S}_{\text{COM-PR}}$ received a $(\texttt{commit}, cid, P_i, P_j)$-message from $\mathcal{F}_{\text{COM-PR}}$, meaning that P_i, while honest, received input $(\texttt{receipt}, cid, P_i, P_j, m)$ on $\mathcal{F}_{\text{COM-PR}}$. Therefore, unless we instruct $\mathcal{F}_{\text{COM-PR}}$ otherwise, it will store m under cid and output $(\texttt{receipt}, cid)$ to P_j, which is what we desire. Observe that when P_i was corrupted $\mathcal{S}_{\text{COM-PR}}$ learned m, and $\mathcal{S}_{\text{COM-PR}}$ knows an opening of c_L to m. This will become essential.

(b) If the $\texttt{com}_1$-message was sent by $\mathcal{Z}$ and K is an X-key, then use t_N (the X-trapdoor of N) to decrypt c_2 and let m' denote the resulting value and input $(\texttt{change}, cid, m')$ on the SIT of $\mathcal{F}_{\text{COM-PR}}$.

(c) If the $\texttt{com}_1$-message was sent by $\mathcal{Z}$ and K is not an X-key, then intuitively the coin-flip protocol for generating K was broken. This should not happen, so let us give up. We let $\mathcal{S}_{\text{COM-PR}}$ terminate with output "coin-flip broken".

receiving an opening (R.3):

This is how to simulate an honest receiver P_j receiving an opening. On receiving a message $(\texttt{open}, cid, m', r'_2, r'_L)$ from P_i, check whether $c_2 = \text{commit}_K(m', r'_2)$ and $c_L = \text{commit}_L(m'; r'_L)$. If not, then P_i is corrupt, and we can input $(cid, \texttt{fail})$ on the SIT of $\mathcal{F}_{\text{COM-PR}}$ to prevent $\mathcal{F}_{\text{COM-PR}}$ from taking further actions on the commitment with id cid. If the check succeeds, we must make P_j output $(\texttt{open}, cid, P_i, P_j, m')$ on $\mathcal{F}_{\text{COM-PR}}$.

Figure 9.3(b) (cont. from Fig. 9.3(a) on the preceding page, cont. in Fig. 9.3(c) on the following page): The interface $\mathcal{S}_{\text{COM-PR}}$ used in the proof of Theorem 9.1 on page 291

That $\Pr[\text{IDEAL}^{\mathcal{F}_{\text{CRS}}^{\texttt{com}}, \mathcal{F}_{\text{PRS}}^{\text{commit}}}_{\mathcal{F}_{\text{COM-PR}}^{\texttt{com}}, \mathcal{S}_{\text{COM-PR}}, \mathcal{Z}} \in \text{``H} \rightsquigarrow \mathcal{F}_{\text{CRS}}\text{''} \cup \text{``H} \rightsquigarrow \mathcal{F}_{\text{PRS}}\text{''}] \overset{\text{c}}{\approx} 0$ is straight-forward to verify, so we proceed to prove the second claim.

Interface $\mathcal{S}_{\text{COM-PR}}$

receiving an opening (R.3) (cont.):

(a) If the open-message was sent by $\mathcal{S}_{\text{COM-PR}}$ (in opening (C.3)), then input cid on the SIT of $\mathcal{F}_{\text{COM-PR}}$ to make it output $(\texttt{open}, cid, P_i, P_j, m')$ to P_j.

(b) If the open-message was sent by $\mathcal{Z}$, then P_i is corrupt and we can input $(\texttt{open}, cid, P_i, P_j)$ on behalf of P_i to the ideal functionality and then input cid on the SIT to make it output $(\texttt{open}, cid, P_i, P_j, m'')$ to P_j.

As a result $\mathcal{S}_{\text{COM-PR}}$ receives a message $(\texttt{open}, cid, P_i, P_j, m)$ from $\mathcal{F}_{\text{COM-PR}}$. If $m \neq m'$, which only happens in case (b), then $\mathcal{S}_{\text{COM-PR}}$ sent the $\texttt{com}_1$-message in the commit protocol with id cid: If not, when the commitment was received by $\mathcal{S}_{\text{COM-PR}}$, the simulator would have executed case R.2(b) or case R.2(c). Clearly not case R.2(c), or the simulation would not have reached this point. So, case (b) must have been executed, meaning that $\mathcal{S}_{\text{COM-PR}}$ decrypted c_2 and put the resulting value m into $\mathcal{F}_{\text{COM-PR}}$. Since K is an X-key and $c_2 = \text{commit}_K(m'; r_2')$ it follows that $m = m'$, a contradiction. So, $\mathcal{S}_{\text{COM-PR}}$ *did* send the $\texttt{com}_1$-message in the commit protocol with id cid, so $\mathcal{S}_{\text{COM-PR}}$ have executed R.2(a) and therefore knows an opening $c_L = \text{commit}_K(m; r)$, see the remark in R.2(a). Since $\mathcal{Z}$ sent an opening $c_L = \text{commit}_K(m'; r_2')$ above and $m \neq m'$, we have a double-opening. This should not, so let give up. We let $\mathcal{S}_{\text{COM-PR}}$ terminate with output "here is a double-opening $(N, K, c_L, m, r, m', r')$".[a]

proving relation:

On input $(\texttt{prove}, cid, P_i, P_j, R, cid_1, \ldots, cid_a)$ from $\mathcal{F}_{\text{COM-PR}}$, input $(cid, P_i, P_j, (N, L, c_{L,1}, \ldots, c_{L,a}))$ to ${\mathcal{S}_{\text{DAM-ZK}}}^R$ to make it simulate a proof from P_i to P_j with the specified instance, where $c_{L,i}$ is the commitment under L from the commitment with id cid_i. Then proceed as follows:

1. If P_i is honest and if P_i is later corrupted we learn $m_1, \ldots, m_a$ and compute, as described in committing, values r_i such that $c_i = \text{commit}_L(m_i; r_i)$. Then input $(m_1, r_1, \ldots, m_a, r_a)$ to ${\mathcal{S}_{\text{DAM-ZK}}}^R$ to make it patch the internal state of P_i to these values.
2. If P_i is corrupted and P_j is honest, then ${\mathcal{S}_{\text{DAM-ZK}}}^R$ simulates P_j. The result is (cid, b). If $b = 0$, then input $(\texttt{fail}, cid)$ on the SIT of $\mathcal{F}_{\text{COM-PR}}$. This makes $\mathcal{F}_{\text{COM-PR}}$ output $(cid, 0)$ to P_j. If $\mathcal{Z}$ outputs $(cid, 1)$, then simulate P_j outputting $(cid, 1)$ in the simulation and input cid on the SIT of $\mathcal{F}_{\text{COM-PR}}$, which makes $\mathcal{F}_{\text{COM-PR}}$ output (cid, c), where $c = 1$ unless the values stored under $cid_1, \ldots, cid_a$ are not in R. If $\mathcal{F}_{\text{COM-PR}}$ outputs $(cid, 0)$, then the simulation failed, so let's just give up. We let $\mathcal{S}_{\text{COM-PR}}$ terminate with output "failure in proving relation".

[a] Notice how the presence of c_L in the *first* message was used essentially to compute a double opening when the simulation fails because of $m \neq m'$. Without c_L we would just have a failed simulation.

Figure 9.3(c) (cont. from Fig. 9.3(b) on the preceding page): The interface $\mathcal{S}_{\text{COM-PR}}$ used in the proof of Theorem 9.1 on page 291

The intuition is the following: First of all, that L in the simulation is a random E-key and not a random X-key, and that the keys K for honest parties are random E-keys and not random keys, should not matter by the key-indistinguishability assumption. Second, the commitment keys K for corrupted parties are generated via coin-flips, so corrupted parties will get random K, which by the assumption that the X-key constitute the entire key space except for a negligible fraction implies that K is an X-key, except with negligible probability. However, this intuition fails under more careful scrutiny of the simulator. Consider namely the case where P_i is honest until Step C.2, so that $\mathcal{S}_{\text{COM-PR}}$ generates K as a random E-key and sends the message as specified in the simulation. Then $\mathcal{Z}$ corrupts P_i, which allows $\mathcal{Z}$ to send a new message on behalf of the now corrupt P_i — we have a rushing environment. We say that $\mathcal{Z}$ captures an E-key. After the capture $\mathcal{Z}$ can commit under an E-key. So, what could happen? Well, if $\mathcal{Z}$ computes c_2 as a commitment to another value m', then we cannot learn the value m' by decrypting c_2. Therefore the simulator does not know which value to input to $\mathcal{F}_{\text{COM-PR}}$ to make it consistent with c_2. Potentially, this means that if the commitment is later opened, the output of P_j in the simulation, m', and the output of $\mathcal{F}_{\text{COM-PR}}$, are not the same, which makes the two trivial to distinguish. This is the role of the commitment c_L sent with the $\texttt{com}_1$-message. To later open the commitment $\mathcal{Z}$ must open both c_2 and c_L to the same value. This means that if $\mathcal{Z}$ captures an E-key, which it must do by corrupting in Step C.2, then the $\texttt{com}_1$-message was sent by $\mathcal{S}_{\text{COM-PR}}$ and opened as $c_L = \text{commit}_L(m; r_L)$, where m is the value input to $\mathcal{F}_{\text{COM-PR}}$ under the id *cid*. By inspection of the simulation this is exactly the value input to $\mathcal{F}_{\text{COM-PR}}$ by $\mathcal{S}_{\text{COM-PR}}$, so this leaves three possibilities: Either $\mathcal{Z}$ opens both c_2 and c_L to m, or $\mathcal{Z}$ opens c_2 and c_L to different values, or $\mathcal{Z}$ opens c_L to a value different from m. In the two first cases the simulation and $\mathcal{F}_{\text{COM-PR}}$ are consistent, resulting in output $(\texttt{open}, \mathit{cid}, m)$ respectively no output. In the last case the computational binding of commit_L was broken, which happens only with negligible probability.

Above we appealed to the computational binding of commit_L. There is one serious problem with that, namely that $\mathcal{S}_{\text{COM-PR}}$ uses the E-trapdoor of L. We can however still obtain a contradiction to the computational binding of commit_L, though some care must be taken, which there will be in the more formal proof following this intuitive sketch. Here we sketch the approach. We basically want to get rid of the use of t_N and t_L — i.e. we want to generate a distribution identical to the simulation given as input a random N and a random L with unknown trapdoor — after which we can appeal to the computational binding of commit_L. To get rid of the use of t_L we simply run the simulation, but cheating by letting the dummy value m' in C.1 be equal to the correct value m. By inspection of the simulator, now t_L is not used anymore. Then to get rid of the use of t_N, instead of decrypting c_2 we use $m' = 0$, and if $\mathcal{Z}$ ever opens the corresponding commitment to some value m'', instead of letting $\mathcal{F}_{\text{COM-PR}}$ output $m' = 0$, we simply patch the execution and let it output m'' to force the simulation and $\mathcal{F}_{\text{COM-PR}}$ to be consistent. This experiment does not use t_N nor t_L and is identical to the simulation, at least until the computational binding of commit_L is broken, which happens with negligible probability as we do not use any of the trapdoors t_N or t_K.

We now proceed with a more careful proof, via a hybrids argument. The hybrids used for pushing the rewinding to the analysis will be similar of those in the proof of Theorem 8.4 on page 275, and the formal details can be developed along the exact same lines. We consider

the following hybrids:

1. To produce $H^1_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ we execute the simulation $\text{IDEAL}^{\mathcal{F}_{\text{CRS}}^{\text{com}},\mathcal{F}_{\text{PRS}}^{\text{commit}}}_{\mathcal{F}_{\text{COM-PR}},\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$, but without using t_L. Instead we compute c_L as in the protocol. This is done as follows: We are running $\text{IDEAL}^{\mathcal{F}_{\text{CRS}}^{\text{com}},\mathcal{F}_{\text{PRS}}^{\text{commit}}}_{\mathcal{F}_{\text{COM-PR}},\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ and by inspection of $\mathcal{S}_{\text{COM-PR}}$ it can be seen that when $\mathcal{S}_{\text{COM-PR}}$ is about to compute $c_L = \text{commit}_L(m'; r'_L)$, the party P_i is honest and have received an input $(\texttt{commit}, cid, P_i, P_j, m)$ on $\mathcal{F}_{\text{COM-PR}}$. So, by inspecting the communication between $\mathcal{Z}$ and $\mathcal{F}_{\text{COM-PR}}$ in $\text{IDEAL}^{\mathcal{F}_{\text{CRS}}^{\text{com}},\mathcal{F}_{\text{PRS}}^{\text{commit}}}_{\mathcal{F}_{\text{COM-PR}},\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ we can easily determine m. We then simply let $m' = m$, where m' is the dummy value mentioned in C.1 in the simulation. Now, by inspection of the simulation it can be verified that the value t_L is no longer used.

   By the assumption that trapdoor openings are distributed statistically close to real openings, we have that $H^1_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z) \overset{\text{s}}{\approx} \text{IDEAL}^{\mathcal{F}_{\text{CRS}}^{\text{com}},\mathcal{F}_{\text{PRS}}^{\text{commit}}}_{\mathcal{F}_{\text{COM-PR}},\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$.

2. To produce $H^2_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ we execute $H^1_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$, but without using t_N. As we cannot do the decryption in R.2(b) and we cannot determine whether we are in case R.2(b) or case R.2(c), we always use $m = \perp$ to indicate that the value is undefined. Then in R.3(b) (in receiving an opening (R.3)), when $\mathcal{F}_{\text{COM-PR}}$ is about to output $(\texttt{open}, cid, P_i, P_j, \perp)$ to P_j, we patch the simulation and deliver $(\texttt{open}, cid, P_i, P_j, m')$ to $\mathcal{Z}$, where m' is the value to which $\mathcal{Z}$ opens c_2 and c_L in the open-message.

   Furthermore, in receiving an opening (R.3), when $\mathcal{S}_{\text{COM-PR}}$ is about to output "here is a double-opening $(N, K, c_L, m, r, m', r')$", instead deliver $(\texttt{open}, cid, P_i, P_j, m')$ to $\mathcal{Z}$.

   Notice that because some of the values in $\mathcal{F}_{\text{COM-PR}}$ are undefined ($\perp$), if supplied by a corrupted party, we cannot run $\mathcal{F}_{\text{COM-PR}}$ on inputs $(\texttt{prove}, cid, P_i, P_j, R, cid_1, \ldots, cid_a)$ for corrupted P_i. Instead we simulate it by returning (cid, b) to P_j iff P_j returns (cid, b) in the simulation. So, we simply force $\mathcal{F}_{\text{COM-PR}}$ to be consistent with the simulation on the inputs $(\texttt{prove}, cid, P_i, P_j, R, cid_1, \ldots, cid_a)$ for corrupted parties.

   That $H^1_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z) \overset{\text{c}}{\approx} H^2_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ is argued after the presentation of the remaining hybrids.

3. To produce $H^3_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ we execute $H^2_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$, but with the difference that we let L be a random key instead of a random E-key.

   That $H^3_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z) \overset{\text{c}}{\approx} H^2_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ follows from key indistinguishability as we no longer use t_N.

4. To produce $H^4_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ we execute $H^3_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$, but we start committing to the correct value m under the key K. I.e. we let $m'' = m$, where m'' is the second dummy value defined in C.2. Notice that we do not use t_K anymore.

   That $H^4_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z) = H^3_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ follows from fact that the keys K are E-keys and therefore perfect hiding.

5. To produce $H^5_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ we execute $H^4_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$, but we do not generate the keys K as random E-keys anymore. Instead we let the hitting keys in committing be

random keys. Notice that this is indeed possible as, from $H^4_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$, we commit to correct values under K.

That $H^5_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z) \stackrel{c}{\approx} H^4_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ follows from key indistinguishability, using the fact that we do not use t_N.

6. To produce $H^6_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ we execute $H^5_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$, but we start generating the keys K as in the protocol. I.e., instead of hitting a random key, we flip a key.

   That $H^6_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z) = H^5_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ follows from the fact that $+$, as in $K_1 + K_2$, is a group operation on the set of keys, and therefore $K_1 + K_2$ is a random key when P_i is honest: The commitment c_1 is perfect hiding, so K_2 is independent of K_1. Since K_1 is chosen uniformly at random when P_i is honest, the claim follows.

7. To produce $H^7_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ we execute $H^6_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$, but instead of the simulator $\mathcal{S}_{\text{DAM-ZK}}$ we run the protocol $\pi_{\text{DAM-ZK}}$ from Fig. 5.1 on page 180. This is possible as we are using correct inputs for all honest parties now, so all witnesses are known.

   That $H^7_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z) \stackrel{c}{\approx} H^6_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ follows from Theorem 5.1 on page 180.

8. That $\text{HYB}^{\mathcal{F}^N_{\text{CRS}},\mathcal{F}^{\text{commit}}_{\text{PRS}}}_{\pi^{\text{com}}_{\text{COMPR}}[\pi^R_{\text{DAM-ZK}}/\mathcal{F}^R_{\text{ZK-PM}}],\mathcal{Z}} \stackrel{s}{\approx} H^7_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$ follows by inspection. In the experiment $H^7_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$, the output of an opening by a corrupted party is always the value to which that corrupted party opened c_L, see the definition of H^2. As for an honest party (when the $\texttt{com}_1$ message was sent by $\mathcal{Z}$) the output is the value m input to $\mathcal{F}_{\text{COM-PR}}$. We argue that the same is the case in $\text{HYB}^{\mathcal{F}^N_{\text{CRS}},\mathcal{F}^{\text{commit}}_{\text{PRS}}}_{\pi^{\text{com}}_{\text{COMPR}}[\pi^R_{\text{DAM-ZK}}/\mathcal{F}^R_{\text{ZK-PM}}],\mathcal{Z}}$: Since L is a random key, and therefore a random X-key, except with a probability statistically close to 0, and the commitment c_L was sent by P_i (as part of the $\texttt{com}_1$-message) it follows that c_L is a perfect binding commitment to m. Since c_L must be opened to m' for the output to be m', it follows that the output of an opening is always m. Since we also force the output of $\mathcal{F}_{\text{COM-PR}}$ to be consistent with the simulation on proofs of relation the above argument extends to show the claim.

What remains is to argue that $H^2_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z) \stackrel{c}{\approx} H^1_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$.

There is only one case where the first modification (patching at the time of output instead of decrypting) gives a difference in the output from $\mathcal{F}_{\text{COM-PR}}$ in H^1 and H^2 when a commitment is opened, namely when the $\texttt{com}_1$-message was sent by $\mathcal{Z}$ and K is an E-key, i.e. when the simulation is given up with output "coin-flip broken". To see this observe that if the $\texttt{com}_1$-message was sent by $\mathcal{S}_{\text{COM-PR}}$, then we are in case R.2(a), which is identical in the two distributions, and if the $\texttt{com}_1$-message was sent by $\mathcal{Z}$ and K is an X-key, then in H^1 the output of $\mathcal{F}_{\text{COM-PR}}$ after an opening is always the value m' obtained by decrypting c_3, as this value is handed to $\mathcal{F}_{\text{COM-PR}}$ by $\mathcal{S}_{\text{COM-PR}}$ as the committed value in the `change`-command. In H^2 the output is also the value (which could have been) obtained by decryption as K is an X-key and so c_3 is perfect binding, to m', and must be opened to a value m'' for the output of the simulation to be m''. Therefore the output of the simulation is m', which proves the claim, as we force $\mathcal{F}_{\text{COM-PR}}$ to be consistent with the simulation in H_2.

The only way the second modification (of forcing the output from the open commands to be consistent with the simulation instead of outputting "here is a double-opening $(N, K, c_L, m, r, m', r')$") can make a difference is that in H^1 "here is a double-opening $(N, K, c_L, m, r, m', r')$" is output with a non-negligible probability. Since H^1 and H^2 are identical until this happens this would imply that a double openings under L is computed in H^2 with a non-negligible probability. Since $H^2 \stackrel{c}{\approx} H^3$ and L is a binding key in H^3, clearly double openings under L are computed with negligible probability in H^2. We can therefore ignore the possibility that "here is a double-opening $(N, K, c_L, m, r, m', r')$" is output.

The only way the third modification (of forcing the output from a proof of relation to be consistent with the simulation) can make a difference is that in H^1 the simulation and $\mathcal{F}_{\text{COM-PR}}$ have different outputs on an input $(\texttt{prove}, cid, P_i, P_j, R, cid_1, \ldots, cid_a)$, i.e. when "failure in proving relation is output" is output.

We now prove that the first and the third modifications do not matter either. Let p_1 be the probability, in H^1, that "coin-flip broken" is output and let p_2 be the probability that "failure in proving relation" is output. By the above analysis it is enough to prove that p_1 and p_2 are negligible.

Assume first that p_1 is not negligible. This means that with non-negligible probability, in H^1, the simulator outputs "coin-flip broken". Since H^1 and H^2 are identical until this happens, it happens in H^2 with probability at least p_1 that the simulator should have output "coin-flip broken", but did not — we consider H^2 here because it does not use t_N. Consider the following experiment: Run H^2 until it happens that c_1 was opened correctly and $K = K_1 + K_2 \in E$ for a key generated by a committer P_i which was corrupted when the $\texttt{com}_1$-message was sent, and a receiver P_j which was honest when the $\texttt{com}_3$-message was received. Here E denotes the set of E-keys, and the test $K \in E$ can be done e.g. by running through all possible choices of (m_1, r_1, m_2, r_2) and testing whether $\text{commit}_K(m_1; r_1) = \text{commit}_K(m_2; r_2)$ for some $m_1 \neq m_2$. It happens with non-negligible probability at least p_1 that $K \in E$. Then rewind until the point where K_2 was sent and rerun until it happens again that c_1 was opened correctly and $K = K_1' + K_2' \in E$ and that P_j was honest when the $\texttt{com}_3$-message was received. Exactly as in the proof of Theorem 5.1 on page 180 we can prove that the expected time used in rerunning the protocol is polynomial (of course the time used in the tests $K \in E$ is not). Since only expected polynomial time is used rerunning, the probability that any of the challenges K_2' generated in the reruns belong to $(E - K_1)$ is negligible. Therefore, except with negligible probability $K_2' + K_1 \notin E$. Together with $K_1 + K_2 \in E$ and $K_1' + K_2' \in E$ this implies that $K_1' \neq K_1$, so a double opening of c_1 is generated. This happens with a probability at most negligibly smaller than p_1. The problem is that the experiment is not PPT, but we can deal with that. Let p_3 be a polynomial such that running the above experiment for $p_3(k)$ steps, still not counting the test $K \in E$, gives a double opening with probability at least $1/p_3(k)$. Then the test $K \in E$ is not needed in the rerunning anymore, as we can simply rerun until either a double opening is produced or $p_3(k)$ steps were used. To get rid of the first test $K \in E$ for when to start the reruns, we do as follows: Since $H^2_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k, z)$ is PPT there is a polynomial upper bound b on the number of keys K generated in a run. Pick $i \in \mathbf{Z}_b$ uniformly at random and let it index a run of the commitment protocol. Run $H^2_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k, z)$ until the execution of this instance of the commitment protocol has terminated. If P_i was corrupted when the $\texttt{com}_1$-message was sent

and P_j was honest when the $\texttt{com}_3$-message was received, and P_i opened the commitment c_1 correctly, then save c_1, K_1, r_1 and start rerunning until a double opening is produced or $p_3(k)$ steps used. Since b is a polynomial and there is a non-negligible probability that $K \in E$ for some K, it follows that there is a non-negligible probability p_4 that we start rerunning for the first $K \in E$ as we would if we did the tests $K \in E$. So, with non-negligible probability $p_4 p_3$ we produce a double opening, but now in PPT. Since we do not use t_N nor $t_{\overline{K_j}}$, this contradicts the computational binding of $\text{commit}_{\overline{K_j}}$. We have proved that p_1 is negligible and we proceed under the assumption that all keys K generated in a run are X-keys when the $\texttt{com}_1$-message is sent by $\mathcal{Z}$.

Assume then that p_2 is not negligible. Observe that in $H^1_{\mathcal{S}_{\text{COM-PR}},\mathcal{Z}}(k,z)$, when $(pid, P_i, P_j, (N, L, c_1, \ldots, c_a))$ is input to the simulator $\mathcal{S}_{\text{DAM-ZK}}$ and P_i is corrupted, then for each of the commitments c_i under K_i corresponding to $c_{L,i}$ it holds that K_i is either an X-key, if the $\texttt{com}_1$-message was sent by $\mathcal{S}_{\text{COM-PR}}$, or K_i is an E-key and $\mathcal{S}_{\text{COM-PR}}$ knows m_i, r_i such that $c_{L,i} = \text{commit}_L(m_i; r_i)$, when the $\texttt{com}_1$-message was sent by $\mathcal{S}_{\text{COM-PR}}$.

Furthermore, since $\mathcal{S}_{\text{DAM-ZK}}$ accepted the proof, we can use rewinding to extract a witness $(m'_1, r'_1, \ldots, m'_a, r'_a)$ such that $c_{L,i} = \text{commit}_L(m'_i; r'_i)$ and $(m'_1, \ldots, m'_a) \in R$. And from the proof that c_i and $c_{L,i}$ commits to the same values we can extract a witness (m''_i, r''_i, r'''_i) such that $c_{L,i} = \text{commit}_L(m''_i; r''_i)$ and $c_i = \text{commit}_{K_i}(m''_i; r'''_i)$. To do the extraction we note that $\mathcal{S}_{\text{COM-PR}}$ is of the form $\mathcal{S}_{\text{COM-PR}} = \mathcal{S}_{\text{COM-PR}}'[\mathcal{S}_{\text{DAM-ZK}}]$ and only uses $\mathcal{S}_{\text{DAM-ZK}}$ to simulate instances that are in the relation. This follows from the fact that all involved keys are E-keys and can therefore be opened to arbitrary values, in particular values in the relation. Using the techniques used to prove Eq. 8.2 on page 277 and Eq. 8.3 on page 278 we can then argue that we can indeed extract witnesses except with negligible probability.

For the indices where K_i is an X-key, it holds that the value m''_i was found by decrypting and then input to $\mathcal{F}_{\text{COM-PR}}$ by $\mathcal{S}_{\text{COM-PR}}$ and is therefore the value stored under cid_i, see R.2(b). For the indices where K_i is an E-key, the value m_i for which $\mathcal{S}_{\text{COM-PR}}$ knows r_i such that $c_{L,i} = \text{commit}_L(m_i; r_i)$ is the value stored by $\mathcal{F}_{\text{COM-PR}}$, see R.2(a).

Assume for notational convenience that the E-keys are the first j keys. It follows from the fact that $\mathcal{F}_{\text{COM-PR}}$ did not accept the proof that $(m_1, \ldots, m_j, m''_{j+1}, \ldots, m''_a) \notin R$ and we have that $(m'_1, \ldots, m'_a) \in R$, so clearly $(m_1, \ldots, m_j, m''_{j+1}, \ldots, m''_a) \neq (m'_1, \ldots, m'_a)$. Assume that $m_i \neq m'_i$ for some $1 \leq i \leq j$. The simulator knows an opening $c_{L,i} = \text{commit}_L(m_i; r_i)$, as $\mathcal{S}_{\text{COM-PR}}$ sent the $\texttt{com}_1$-message (see R.2(a)). Furthermore, $\mathcal{S}_{\text{COM-PR}}$ knows $c_{L,i} = \text{commit}_L(m'_i; r'_i)$ from the witness extraction. In that case we have a double-opening of $c_{L,i}$. Assume that $m'_i \neq m''_i$ for some $j < i \leq a$. The simulator knows an opening $c_{L,i} = \text{commit}_L(m'_i; r'_i)$ and an opening $c_{L,i} = \text{commit}_L(m''_i; r''_i)$ from the witness extraction. Again we have a double opening of $c_{L,i}$.

This means that if we run H^1 and extract all witnesses using $\texttt{xtr}$, then if the probability of computing a double opening under L is negligible, then p_2 is negligible. Since H^1 and H^2 are identical until "failure in proving relation" is (respectively should have been) output, it follows that if we run H^2 and extract all witnesses using $\texttt{xtr}$, then the probability of computing a double opening under L is negligible. So, since we can generate H^2 given N and L as input it follows from the computational binding of commit_L that p_2 is negligible.

There is one apparent problem with this argument, *we have not required that* $\pi_{\text{DAM-ZK}}$ *is run as a proof of knowledge!* So, how can we extract all witnesses? We can not, and we

do not have to. Recall that in the proof of Theorem 5.1 on page 180 we used the condition that $\pi_{\text{DAM-ZK}}$ is run as a proof of knowledge to guarantee that during the extraction of one witness, we would not accept another proof which we would then have to extract a witness for before we could continue the ongoing extraction. In the above argument we need not run into this problem. We are not extracting the witnesses to *be able to continue the simulation*, but only to get our hands on the witnesses for the purpose of comparing them to the openings we already have. Therefore, during the reruns, we simply ignore all other accepted proofs. We only need to extract witnesses for the 'main copy'. In fact, if with a non-negligible probability there is a proof which would give us a double opening if extracted, then as we did in the proof that p_1 is negligible, we could just guess which proof it is, with a polynomial probability, and then extract that one proof, ignoring even all other accepted proofs in the main copy. □

9.5 Perfect Hiding or Perfect Binding

The scheme described above is not guaranteed to be perfect binding nor perfect hiding. Here we sketch how to construct a version of the commitment scheme with perfect hiding.

First notice that if we simply require L to be a random X-key, then the scheme is perfect binding, as the commitment under L is perfect binding. Letting L be a random X-key instead of a random key makes no difference because of the key indistinguishability.

To allow for perfect hiding commitments, include two random E-keys L_2 and P in the common reference string. To do a perfect hiding commitment the committer computes a uniformly random padding message $p \in \mathcal{M}_N$ and commits with $c_2 = \text{commit}_K(p + m; r_2)$, $c_{L_2} = \text{commit}_{L_2}(p + m; r_{L_2})$ and $c_P = \text{commit}_P(p; r_P)$, This is perfect hiding. The commitments c_{L_2} and c_P are perfect hiding, so given even $p+m$ the receiver would get no information about m. To open to m the committer must then send openings of the above form.

To do the simulation simply let the simulator make the excusable mistake of letting L be a random E-key, which is also the distribution of L in the current simulation, and letting P be a random X-key. Consider then a commitment of the form $c_2 = \text{commit}_K(p + m; r_2)$, $c_{L_2} = \text{commit}_{L_E}(p + m; r_{L_2})$ and $c_P = \text{commit}_P(p; r_P)$ in the simulation. When it is given by an honest party K is an E-key. Therefore, as also L_2 is an E-key, the simulator is not committed to $p+m$ and therefore not committed to m. When the environment commits, and sends the $\texttt{com}_1$-message, the key K is an X-key except with negligible probability. Since P is also an X-key the environment is committed. Assume then that the environment captures an E-key K by corrupting while the $\texttt{com}_3$-message is in transit. Even if the environment sends a new c_2' commitment it must under all circumstances open it to the same value as c_{L_2} which is sent in the $\texttt{com}_1$-message. So, L_2 servers exactly the same role L does in the current protocol, to protect against double openings under captured E-keys.

CHAPTER 10

General Multiparty Computation, Adaptive Security

The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore, all progress depends on the unreasonable man.
— George Bernard Shaw

10.1 Introduction

In this section we present an adaptive secure realization of $\mathcal{F}_{\text{BABB}}$. That a protocol is secure against environments which might corrupt parties during the computation does not follow from the static security of the protocol. In general, constructing adaptive secure protocols is considerably more involved than constructing static secure ones, as already exemplified in Chapter 4 by the difference between ordinary IND-CPA secure encryption and non-committing encryption. Another example is given by Cramer, Damgård, Dziembowski, Hirt and Rabin [CDD+99]. They give an example of a protocol which is secure against computationally unbounded adversaries, but is insecure against PPT adaptive adversaries! Yet another example is provided by Beaver [Bea96] who showed that the protocol in [GMW87], which was proved statically secure, is *not* adaptive secure, unless factoring is not intractable or the polynomial time hierarchy collapses. The issue of adaptive security vs. static security has been studied in detail by Canetti, Damgård, Dziembowski, Ishai and Malkin [CDD+01].

Our protocol will show that the threshold homomorphic encryption based approach to MPC can be used to realize adaptively secure general MPC for the cryptographic model without realizing non-committing encryption. Indeed, the protocol will be as efficient as the statically secure realization from Chapter 8. It is therefore the first general MPC solution for the cryptographic model where going to adaptive security does not cause a major loss in efficiency (or costs an extra assumption, such as secure erasures). In particular it seems

to be the first protocol which does not either realize non-committing encryption or assume erasures.[1]

We note that even though the protocol in [CLOS02] is not for the private channels model, it still uses non-committing encryption in an essential manner, namely for building adaptive secure oblivious transfer. That non-committing encryption is used for realizing adaptive secure OT is not a coincidence of course: If the sender inputs $b_0 = b_1 = b$, the OT functionality is reduced to an $\mathcal{F}_{\text{SMT}}$ functionality, so adaptive secure OT implies non-committing encryption with the same complexity. One consequence of this is that the protocol in [CLOS02] suffers the non-committing encryption overhead of a factor at least k, as all protocol realizing adaptive secure two-party computation will, unless of course one is willing to assume that parties can be trusted to securely erase certain critical data or one is able to realize non-committing encryption with a constant expansion factor.

10.1.1 Related Work

An adaptive secure MPC protocol is not a novelty though. Indeed, the very first general MPC protocols in [BGW88, CCD88] for the secure channels model are adaptive secure. And, as discussed in Chapter 4, this immediately implies adaptive secure protocols for the cryptographic model: We can start from an adaptive secure protocol for the secure channels model and realize the channels using non-committing encryption.

10.1.1.1 Reactive Security

One can consider an even more powerful adversary than the adaptive one. An adversary which is allowed to break into all parties, as long as it is only broken into a certain threshold of the parties during a given period of time — it can move around between the parties. This can e.g. be a realistic model of how viruses attack a network, at the same time spreading to new machines and being deleted on old ones. The studied of such adversaries was initiated by Ostrovsky and Yung [OY91] who called them mobile adversaries. Later Canetti and Herzberg [CH94] dubbed security against mobile adversaries proactive security. Proactive security has spawned a large amount of research in the context of threshold cryptosystems, in particular threshold signature schemes. We will not consider mobile adversaries in this chapter.

10.1.1.2 Separating Security and Termination

Independent of our work Canetti, Lindell, Ostrovsky and Sahai [CLOS02] published a universally composable adaptive secure general multiparty computation protocol. This protocol stands out in several respects compared to the adaptive secure MPC protocol mentioned

[1] The adaptively secure MPC protocol from [CG96] does not realize non-committing encryption either or use it to realize secure channels. The protocol, however, uses deniable encryption [CDNO97], which is harder to realize than non-committing encryption. It is not even known whether deniable encryption could exists. At the time of writing no realization of deniable encryption is known based on standard complexity theoretic assumptions.

above: It is not designed for the secure channels model and then augmented with non-committing encryption and second, it tolerates the corruption of *all* parties. The protocol basically follows the approach from [GMW87] but this time realizing all the primitives used in building the MPC protocol in [GMW87] in a *universally composable* way. Since the MPC problem contains the BA problem as a special case, it is clear of course that a protocol which allows more than half the parties to be corrupted cannot guarantee termination. However, as discussed in [GMW87], if termination is desired it can be added at the price that now only a minority of corrupted parties can be tolerated. Other work on protocols tolerating more than a minority of corrupted parties include the work by Beaver, Goldwasser and Levin [BG89, GL90].

We note that by a trivial simplification of our protocol we can obtain a protocol withstanding $n-1$ corrupted parties. Compared to the protocol from [CLOS02] this is however not particularly interesting. Even though tolerating $n-1$ and not n corruptions might not seem as a big difference, it is huge when $n = 2$. Opposed to the protocol in [CLOS02] we cannot do two-party computation secure against the corruption of both parties. This difference comes from the fact that our protocol will not realize non-committing encryption, which is exactly a secure two-party computation! However, what we lose in resilience, we win in efficiency.

10.1.2 Informal Description of the Main Ideas

We give an informal sketch of how we make the protocol from Chapter 8 adaptively secure.

10.1.2.1 Sender Non-Committing Encryption

We start by giving a sketch of how we avoid using non-committing encryption. As in Chapter 8 we assume that from the start the following scenario has been established: We have a IND-CPA secure threshold homomorphic encryption scheme given with public-key pk known by all parties and the matching private decryption key shared among the parties. We take as a starting point the protocol in Chapter 8 and consider the additional problems that needs to be dealt with to obtain adaptive security.

First of all, during the simulation the simulator has to show the environment encryptions that are claimed to contain the inputs of honest parties. At this time the simulator does not know these inputs, so it must encrypt some arbitrary values. Because of the semantic security, this is fine for the time being, but if one of the honest parties are later corrupted, the simulator learns the real inputs of this party and must reveal them to the adversary along with a simulation of all internal data of the party. The simulator is now stuck, since the real inputs most likely are not consistent with the arbitrary values encrypted earlier.

We handle this problem using a combination of two tricks: First, we include in the public key an encryption $K = E_{pk}(1)$. Then, we redefine the encryption method, and fix the rule that to encrypt a value a, one computes $\text{Blind}(a \boxdot K)$. Under normal circumstances, this will be an encryption of a. The point, however, is that in the simulation, the simulator can decide what K should be, and will set $K = E_{pk}(0)$. Then all encryptions used in the simulation will contain 0. We then require from the encryption scheme used that the simulator can compute C as an encryption of 0, store the random coins used for this and later make it

seem as if C was computed as $C = a \boxdot K$ for any a it desires — i.e. we require that $\text{commit}_K(m) = \text{Blind}(m \boxdot K)$ is a trapdoor commitment scheme when $K \in E_{pk}(0)$.

Notice that this encryption method has the flavor of non-committing encryption: In the simulation we can construct ciphertexts which can later be open arbitrarily. There is one big difference from non-committing encryption though: If one is given the decryption key, then it is trivial to distinguish simulated ciphertexts from real ones, as the simulated ciphertexts are all encryptions of 0. Therefore, the scheme is only non-committing to a party which does not know the decryption key — we say that the scheme is sender non-committing. Since no party knows the decryption key in our setting (it is shared between the parties) a sender non-committing encryption scheme will turn out to be sufficient.

10.1.2.2 Sender Non-Committing Encryption, Extraction

By letting K be an encryption of 0, we removed one problem and introduced another: The simulator must also be able to find the input values the adversary supplies, immediately as the encryptions are made public. This is not possible anymore as we now only see encryptions of 0. We therefore redefine the way inputs are supplied: For each input value x of party P_i, P_i uses the UC commitment scheme of Chapter 9 to make a commitment $\text{commit}_{K_i}(x)$ to x, with a freshly flipped key K_i, and proves in zero-knowledge that $\text{commit}_{K_i}(x)$ contains the same value as C. Since the commit phase in the protocol from Chapter 9 already contains for each commitment a commitment under the common key L, we can simply let L take the role of K in our protocol. Since K_i is an X-key except with negligible probability it follows that we can extract the plaintext of corrupted parties by decrypting the commitment under K_i, and fake the load of honest parties by letting K_i be an E-key.

Notice that for this approach to work we need $\text{commit}_K(m) = \text{Blind}(m \boxdot K)$ to be a mixed commitment scheme. Since we use E_{pk} both as an encryption scheme and a mixed commitment scheme, for the same key, we cannot do a modular proof in say the $\mathcal{F}_{\text{COM-PR}}$-hybrid model and then plug in the UC commitment scheme from Chapter 9. We have to do the proof over again to verify that this dual use of E_{pk} is actually sound. In doing this we will however use the intuition hopefully build in the proofs of Theorem 8.4 on page 275 and Theorem 9.1 on page 291.

10.1.2.3 Cheating in Decryption

Another problem that we face stems partially from the fact that our encryption scheme is only *sender* non-committing: The simulator will not be able to "cheat" in the threshold decryption protocol by decrypting a given ciphertext to any desired value. The key setup for the decryption protocol fixes the shares of the private key even in the simulation, so a ciphertext can only be decrypted to the value it actually contains. Of course, when decrypting the outputs, the correct results should be produced both in simulation and real life, and so we have a problem since we just said that all ciphertexts in the simulation really contain 0. We solve this by randomizing all ciphertexts before they are decrypted: We include another fixed encryption $R = E_{pk}(0)$ in the public key. Then, given ciphertext C, the parties cooperate to create an encryption $r \boxdot R$, where r is a (secret) randomly chosen value that depends on input from all parties. Then we compute $C \boxplus (r \boxdot R)$ and decrypt this ciphertext. Under normal

circumstances, it will of course contain the same plaintext as C. But in the simulation, the simulator will set $R = E_{pk}(1)$, and "cheat" in the process where r is chosen, s.t. $r = a$, where a is the value the simulator wants the decryption to return. This works, since in the simulation any C actually encrypts 0, so that $C \boxplus (r \boxdot R)$ turns out to be an encryption of a. Then we simply decrypt this value by honestly running the threshold decryption protocol, which ensures a perfect simulation of the decryption.

10.1.2.4 Simulating Threshold Decryption Only in the Analysis

A final problem that we face is that the threshold decryption protocol that we used in Chapter 8 is only statically secure. This was already discussed in Chapter 6. This is the second reason for setting up the output protocol as we did. We set it up so that we could decrypt honestly. This means that we only need to simulate the decryption protocol in the analysis of the simulator, at the point where we need to appeal to the IND-CPA security of the encryption scheme — at these points in the analysis we clearly cannot both use sk for running the decryption protocol correctly and at the same time assume that encryptions cannot be distinguished.[2] Since we only have to simulate the decryption protocol in the analysis, we can yet again use the weaker off-line notion of security — in doing the analysis we are free to rewind the environment as an imaginary experiment. This allow us to use an idea from the single inconsistent party framework.[3] What we will do is to share the secret key between the parties with corruption threshold $t = n - 1$, using the protocol from Section 6.3.3 instantiated with construction threshold $c = n$. Clearly this implies that if just one party refuses to send a decryption share, then the protocol hangs. We ignore this problem at first to obtain a protocol which do not guarantee termination, but which on the other hand can tolerate $n - 1$ corrupted parties.

By sharing the key with corruption threshold $n - 1$, we are able to simulate all public values and the secret key shares of $n-1$ parties and only have one party for which we cannot simulate the internal state if it is corrupted, thereby the name, single inconsistent party. Denote this party by P_s. Since all public values and $n - 1$ of the secret key shares can be simulated statistically close to those of the protocol, if we pick the single inconsistent party uniformly at random and assume that the environment corrupts at most $n - 1$ parties, the probability that P_s is *not* corrupted is at least $\frac{1}{n}$, which is polynomial. Furthermore, when P_s is *not* corrupted, then the simulation is statistically close to the simulation where we decrypted honestly. By repeating the simulation until P_s is not corrupted, we can therefore

[2] To see that we *do* indeed need to simulate the decryption protocol in the analysis, and to see where it is used, consider the following problem: Even though we cheat in such a way that we always decrypt to 'correct' values, this only deals with the simulation of the decryption *protocol*. We also have to simulate the *shares* of the decryption key. When we use the IND-CPA security of the encryption scheme in the analysis (and we *do* need to use the IND-CPA security at some point, as the encryptions of the honest parties contain 0) we can only simulate $c-1$ of the decryption key shares, if shared with construction threshold c. This gives problems in the presence of an adaptive adversary if $c < n$ for most simple schemes, see e.g. [CGJ+99]. Therefore we need to do an additive sharing and use Rabin's backup shares. Specifically the fact that we have an additive sharing will be used to apply the single inconsistent party framework at the first hybrid in the analysis to get rid of the use of sk in the simulation. Thereby IND-CPA security can be used in the remaining hybrids.

[3] The single inconsistent party framework was developed in the context of adaptive secure cryptosystems and signature schemes by Canetti, Gennaro, Jarecki, Krawczyk and Rabin [CGJ+99].

in expected PPT produce a distribution statistically close to the simulation, but without using the secret key. This is then the distribution that we prove indistinguishable from the real-life execution, now being able to use semantic security.

10.1.2.5 A Subtlety in the Single Inconsistent Party Framework

Interestingly the above line of reasoning is false. There can be an almost arbitrary bias between the two views obtained by using the secret key and the view obtained by rerunning. This has to do with the fact that we are looking at *distinguishers* and not adversaries which are e.g. breaking cryptosystems or signature schemes as in [CGJ+99]. In Section 10.2 we discuss why this is so and prove a technical lemma which allows us to cope with it. We soon return to how we add termination to the protocol.

10.2 An Off-Line Single Inconsistent Party

We discussed the concept of an off-line single inconsistent party in the introduction. The basic idea was the following: Create the setting where the simulator can learn enough trapdoor information to get through the simulation without rewinding. This could e.g. be the secret key of an encryption scheme, the security of which the protocol depends on. Then design an off-line experiment where this sensitive trapdoor information is not needed, by introducing a single inconsistent party whose internal state cannot be simulated. Make sure that this single inconsistent party is chosen uniformly at random by the simulator before the simulation and that the single inconsistent party is simulated such that its communication is perfectly, or statistically, indistinguishable from the communication of the consistent parties. In other words, make sure that s is stochastically independent of $\mathcal{Z}$. This means that if $\mathcal{Z}$ is required to corrupt at most $n-1$ parties, then with probability at least $\frac{1}{n}$ it will not corrupt P_s. On the runs where P_s is not corrupted $\mathcal{Z}$ will have seen a run perfectly identical to, or statistically close to, the experiment where the trapdoor information was used. By rerunning an expected n times we can therefore produce the view of $\mathcal{Z}$ on the experiment where the trapdoor information was used, but without using it! As mentioned, this way of arguing is actually false, as demonstrated by the below example.

Consider e.g. an environments $\mathcal{Z}$ trying to distinguish $\text{REAL}_{\pi,\mathcal{Z}}$ and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$. And assume that it is successful in doing so. I.e. $\text{REAL}_{\pi,\mathcal{Z}} \overset{\text{c}}{\not\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$. Assume that we introduce a single, perfectly hidden,[4] inconsistent party and let us define $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\curvearrowleft}}$ to be the distribution obtained by rerunning $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ until it does not corrupt P_s. We construct a situation where $\text{REAL}_{\pi,\mathcal{Z}} \overset{\text{c}}{\not\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ but $\text{REAL}_{\pi,\mathcal{Z}} = \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\curvearrowleft}}$. Assume that we have a protocol for two parties (P_1, P_2) and that $\mathcal{Z}$ can perfectly distinguish $\text{REAL}_{\pi,\mathcal{Z}}$ and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, without corrupting any parties at all. We let $\mathcal{Z}$ output as follows: If it detects that it is in $\text{REAL}_{\pi,\mathcal{Z}}$, then it outputs a uniformly random bit. If it detects that it is in $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, then it flips a bit b. If $b=0$, then it outputs 1 and corrupts a party. If $b=1$, then it outputs 1 with probability $\frac{1}{4}$ and it outputs 0 with probability $\frac{3}{4}$. This means that $\mathcal{Z}$ outputs 1 with probability $\frac{5}{8}$, so it clearly distinguishes. But, assume that P_s is the

[4] I.e. the simulated communication is independent of the identity of the single inconsistent party.

single inconsistent party for uniformly random s. It is not hard to see that the probability that $\mathcal{Z}$ outputs 1 in $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ conditioned on P_s not being corrupted is $\frac{1}{3}1 + \frac{2}{3}\frac{1}{4} = \frac{1}{2}$. Therefore $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}^\curvearrowleft} = \mathrm{REAL}_{\pi,\mathcal{Z}}$. This shows that when one uses the single inconsistent party framework in the context of distinguishers, one has to take care. We show that the above subtlety is not a problem. Basically, one can show that if $\mathrm{REAL}_{\pi,\mathcal{Z}} \overset{\mathrm{c}}{\not\approx} \mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\mathrm{REAL}_{\pi,\mathcal{Z}} = \mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}^\curvearrowleft}$, then the corruption pattern of $\mathcal{Z}$ must be significantly different in $\mathrm{REAL}_{\pi,\mathcal{Z}}$ and $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, which allows to massage $\mathcal{Z}$ into another environment $\mathcal{Z}'$ for which $\mathrm{REAL}_{\pi,\mathcal{Z}'} \overset{\mathrm{c}}{\not\approx} \mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}'^\curvearrowleft}$.

To be able to apply the idea of an off-line single inconsistent party in a concise manner in later analysis — in particular without rewinding — we prove a technical lemma. This lemma will at the same time deal with the issue mentioned above. The lemma basically says that to prove UC security against environments corrupting at most $n-1$ parties it is enough to prove security against all environments which at the beginning of the protocol pick the index of a random party P_s and commit to never corrupting P_s. Of course the interface trying to simulate a protocol execution to $\mathcal{Z}$ will not know s and cannot exploit that $\mathcal{Z}$ has restricted itself, which is the clue to proving that it is enough to consider this class of environments.

Definition 10.1 *Let $\mathcal{Z}$ be any environment. From $\mathcal{Z}$ we define a new environment $\mathcal{Z}^{\mathit{SIP}}$, as follows: On input (k,n,z) it picks uniformly random $s \in [n]$. Then it inputs (k,n,z) to $\mathcal{Z}$ and starts running exactly like $\mathcal{Z}$, except that when $\mathcal{Z}$ outputs $(\mathtt{guess}, b)$, if P_s was corrupted, then $\mathcal{Z}^{\mathit{SIP}}$ outputs $(\mathtt{guess}, 0)$. We call P_s the off-line single inconsistent party (OSIP).*

In a simulation $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\mathrm{SIP}}}$, if the simulator $\mathcal{S}$ actually knew the identity of P_s, then the simulator could safely do a simulation where the internal state of P_s could not be constructed — when it is asked to reveal this state $\mathcal{Z}^{\mathrm{SIP}}$ always outputs 0, so its value does not matter. However, since the behavior of $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\mathrm{SIP}}}$ is exactly identical to that of $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ from the view of $\mathcal{S}$, it has no information about s. But, since s can be read off $\mathcal{Z}^{\mathrm{SIP}}$, when one runs the simulation $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\mathrm{SIP}}}$ in an off-line experiment, e.g. when defining a hybrid, then one *can* exploit that P_s is never corrupted.[5] Therefore we call P_s the *off-line* single inconsistent party. Off-line it is easier to simulate a protocol to an environment of the form $\mathcal{Z}^{\mathrm{SIP}}$. It is therefore interesting that it is enough to consider these environments when proving security.

Lemma 10.1 *Assume that for all environments $\mathcal{Z}$, where $\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{F}\text{''}] \overset{\mathrm{c}}{\approx} 0$, it holds that $\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\mathit{SIP}}} \in \text{``}H \rightsquigarrow \mathcal{G}\text{''}] \overset{\mathrm{c}}{\approx} 0$ and*

$$\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\mathit{SIP}}} \overset{\mathrm{c}}{\approx} \mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}^{\mathit{SIP}}} \ .$$

Then for all environments $\mathcal{Z}$, where

$$\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{F}\text{''}] \overset{\mathrm{c}}{\approx} 0 \ ,$$

[5] Or technically, that the output of $\mathcal{Z}$ is fixed when P_s *is* corrupted. The difference is that the current formalization does not change the probability that IO restrictions are violated, as the protocol is allowed to run to the end.

it holds that

$$\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \in \text{``}H \rightsquigarrow \mathcal{G}\text{''}] \stackrel{c}{\approx} 0$$

and

$$\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \stackrel{c}{\approx} \mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}} \ .$$

Proof. Remember the subtlety discussed above that the distribution of the output of $\mathcal{Z}$ might be different from the distribution of the output of $\mathcal{Z}$ *conditioned* on P_s not being corrupted. This means that we might have that $\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \not\stackrel{c}{\approx} \mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}$ and $\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\mathtt{SIP}}} \stackrel{c}{\approx} \mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}^{\mathtt{SIP}}}$. The proof exploits that this can only be the case if there is a non-negligible difference in the corruption pattern of $\mathcal{Z}$ in $\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}$. In particular there will exist $j \in \{0,1,\ldots,n-1\}$ such that the probability that $\mathcal{Z}$ corrupts j parties is significantly different in the two executions. We exploit this difference to obtain a new environment, called $\mathcal{Z}^j$ below, such that $\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},(\mathcal{Z}^j)^{\mathtt{SIP}}} \not\stackrel{c}{\approx} \mathrm{HYB}^{\mathcal{G}}_{\pi,(\mathcal{Z}^j)^{\mathtt{SIP}}}$.

Let $\mathcal{Z}$ be any environment for which

$$\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \in \text{``}\mathtt{H} \rightsquigarrow \mathcal{F}\text{''}] \stackrel{c}{\approx} 0 \ . \tag{10.1}$$

By the assumption that $\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\mathtt{SIP}}} \stackrel{c}{\approx} \mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}^{\mathtt{SIP}}}$ we in particular have that

$$\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\mathtt{SIP}}} = \perp] \stackrel{c}{\approx} \Pr[\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}^{\mathtt{SIP}}} = \perp]$$

for all $\perp \notin \{0,1\}$. Since we defined $\mathcal{Z}^{\mathtt{SIP}}$ to run $\mathcal{Z}$ until it outputs its guess, it follows that $\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}} = \perp] \stackrel{c}{\approx} \Pr[\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}} = \perp]$ for all $\perp \notin \{0,1\}$. To prove the lemma, it is therefore enough to prove that $\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z) = 1] \stackrel{c}{\approx} \Pr[\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}} = 1]$ for all environments $\mathcal{Z}$.

Consider the following version $\mathcal{Z}^j$ of $\mathcal{Z}$. On input (k,z) it inputs (k,z) to $\mathcal{Z}$ and then runs exactly as $\mathcal{Z}$ until $\mathcal{Z}$ outputs $(\mathtt{guess},b)$. Then $\mathcal{Z}^j$ outputs $(\mathtt{guess},b)$ if exactly j parties were corrupted. Otherwise it outputs 0. For an execution $\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z;r)$, let $f(k,z,r)$ denote the number of parties corrupted by $\mathcal{Z}$ when it outputs its guess. If the execution is terminated earlier because a party violates an IO restriction, then we do not need a value of f. Let $\Pr[]$ denote a probability evaluation with r uniformly random. We have that

$$\begin{aligned} &\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},(\mathcal{Z}^j)^{\mathtt{SIP}}}(k,z;r) = 1] \\ =&\frac{n-j}{n} \Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z;r) = 1 \wedge f(k,z,r) = j] \ . \end{aligned}$$

Because at most $n-1$ parties are corrupted by $\mathcal{Z}$ we therefore get that

$$\begin{aligned} &\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z;r) = 1] \\ =&\sum_{j=0}^{n-1} \Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z;r) = 1 \wedge f(k,z,r) = j] \\ =&\sum_{j=0}^{n-1} \frac{n}{n-j} \Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},(\mathcal{Z}^j)^{\mathtt{SIP}}}(k,z;r) = 1] \ . \end{aligned}$$

In the same way we get that

$$\Pr[\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}(k,z;r)=1]=\sum_{j=0}^{n-1}\frac{n}{n-j}\Pr[\mathrm{HYB}^{\mathcal{G}}_{\pi,(\mathcal{Z}^j)^{\mathtt{SIP}}}(k,z;r)=1]\ .$$

From Eq. 10.1 on the facing page we trivially have that $\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}^j}\in\text{``}\mathtt{H}\rightsquigarrow\mathcal{F}\text{''}]\overset{\mathrm{c}}{\approx}0$. So, by the premise of the theorem we have that

$$\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},(\mathcal{Z}^j)^{\mathtt{SIP}}}\overset{\mathrm{c}}{\approx}\mathrm{HYB}^{\mathcal{G}}_{\pi,(\mathcal{Z}^j)^{\mathtt{SIP}}}\ .$$

Using this and the fact that n is polynomial we get that

$$\begin{aligned}\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z;r)=1]&=\sum_{j=0}^{n-1}\frac{n}{n-j}\Pr[\mathrm{IDEAL}^{\mathcal{G}}_{\mathcal{F},\mathcal{S},(\mathcal{Z}^j)^{\mathtt{SIP}}}(k,z;r)=1]\\&\overset{\mathrm{c}}{\approx}\sum_{j=0}^{n-1}\frac{n}{n-j}\Pr[\mathrm{HYB}^{\mathcal{G}}_{\pi,(\mathcal{Z}^j)^{\mathtt{SIP}}}(k,z;r)=1]\\&=\Pr[\mathrm{HYB}^{\mathcal{G}}_{\pi,\mathcal{Z}}(k,z;r)=1]\ ,\end{aligned}$$

as desired. □

10.3 ABB Threshold Homomorphic Encryption Schemes

In this section we formalize the requirements we need on the homomorphic encryption scheme to be able to realize $\mathcal{F}_{\mathrm{BABB}}$ in the presence of adaptive adversaries. We basically need that it is both a static ABB $(n-1)$-threshold homomorphic encryption scheme (gen,E,D) and a special mixed commitment scheme with the commitment function $\mathrm{commit}_{pk,K}(m;r)=\mathrm{Blind}_{pk}(m\boxdot K;r)$. To be more precise:

Definition 10.2 *Let $\mathcal{THE}=(gen,E,D)$ be a static ABB threshold homomorphic encryption scheme. The commitment scheme derived from $\mathcal{THE}$ is defined as follows: Let $(pk,sk)\leftarrow gen$ be the key generator. The system key is $N=pk$, the X-trapdoor is $t_N=sk$, and the message space is the plaintext ring of E_{pk}, $\mathcal{M}_N=R_{pk}$. The key space is the ciphertext space, $\mathcal{K}_N=\mathcal{C}_{pk}$. The commitment function is given by $\mathrm{commit}_K(m;r)=\mathrm{Blind}_{pk}(m\boxdot_{pk}K;r)$. The X-keys are the keys $K\in\mathcal{K}_N$, where $D_{sk}(K)$ is invertible in R_{pk} and the E-keys are the keys of the form $K=E_{pk}(0;r)$.*

This setup will basically allow us to realize $\mathcal{F}_{\mathrm{BABB}}$ along the lines of the protocol in Chapter 8, but replacing the proofs of knowledge there with the proof of relation between committed values from Chapter 9. Actually, what we will do is simply to run the commitment protocol in Fig. 9.2 on page 291 with L being an encryption of 1. Thereby the commitment $c_L=\mathrm{commit}_L(m;r)=\mathrm{Blind}_{pk}(m\boxdot L;r)\in E_{pk}(m)$ will end up being an encryption of m. We then run a proof as in Fig. 9.2 on page 291 that c_L and the commitment c_2 under the flipped

key K commit to identical values. For this reason we need a Σ-protocol for proving that two commitments commit to identical values. This will be our proof of plaintext knowledge. We then run the protocol from Chapter 8 on the c_L values. We also need that given a commitment $c \in E_{pk}(0)$, one can compute r such that $c = E_{pk}(0;r)$ given sk. Denote this property by E-trapdoor extraction. We also need that given $a \in R_{pk}$ and random bits r_1 and r_2 one can compute $r_{\boxdot}$ and $r_{\boxplus}$ such that $E_{pk}(0;r_{\boxplus}) = E_{pk}(0;r_1) \boxplus E_{pk}(0;r_2)$ and $E_{pk}(0;r_{\boxdot}) = a \boxdot E_{pk}(0;r_1)$. Let us denote this property by E-trapdoor preservation. For randomizing ciphertext we need to do interpolation through the homomorphism, by which we mean that we are able to generate in PPT n elements $e_1, \ldots, e_n \in R_{pk}$ such that they are all invertible and their pairwise differences are invertible, see Section 6.3.1. In the following we assume that these elements are $1, \ldots, n$ for notational convenience. Finally, we also need that the realization of $\mathcal{F}_{\text{THE}}^{\mathcal{THE}}$ is of a particular form lending itself to SIPification.

Definition 10.3 *We say that $\mathcal{THE} = (gen, E, D)$ is an adaptive ABB threshold homomorphic encryption scheme if $\mathcal{THE}$ is a static ABB $(n-1)$-threshold homomorphic encryption scheme where all Σ-protocols are non-erasure Σ-protocols and where the commitment scheme derived from $\mathcal{THE}$ is a special mixed commitment scheme with a Σ-protocol for the relation `equality of committed values` and the realization of $\mathcal{F}_{\text{THE}}^{\mathcal{THE}}$ is given by a statistically secure simple function sharing scheme with gen being the key generator of $\mathcal{THE}$ and the function being $C \mapsto D_{sk}(C)$. We also require that we can do interpolation through the homomorphism and that the scheme has E-trapdoor extraction and is E-trapdoor preserving.*

The Paillier threshold homomorphic encryption scheme meets this definition. This is a simple corollary to the results in Theorem 8.2 on page 268 and Section 8.3.1. That the scheme has E-trapdoor extraction and is E-trapdoor preserving follows from Lemma 2.4 on page 22. The author is not aware of other examples.

Corollary 10.1 *The threshold homomorphic encryption scheme in Section 8.3.1 is an adaptive ABB threshold homomorphic encryption scheme.*

10.4 Rabin's Share Backups

Before presenting the protocol we show how we can realize $\mathcal{F}_{\text{THE}}^{\mathcal{THE}}$ so that it can be simulated with a single inconsistent party P_s. As hinted by Definition 10.3 we are going to distribute the secret key with construction threshold n. This actually means that the simulators $\mathcal{S}_{\text{keydist}}$ and $\mathcal{S}_{\text{eval}}$ from Definition 6.2 on page 194 can simulate the internal values of all parties but one. The only problem is of course, that with a construction threshold of n, if just one honest party refuses to participate, then we cannot decrypt. We solve this problem by using the so-called share backups of Rabin [Rab98]. Basically we distribute a sharing of each of the secret values sv_i between the parties, but with construction threshold c, where $t = c - 1$ is the corruption threshold we want for the protocol. If then a party P_j refuses to give away his evaluation share, the honest parties simply reconstruct his secret value sv_j by exchanging shares, and then from then on compute his evaluation share on their own. We say that the party is reconstructed. If $c \geq n - t$, i.e $2t + 1 \leq n$, then the honest parties can always

Functionality $\mathcal{F}_{\text{key-dist}}^{gen,\text{SingleToThresh},c,\texttt{SIP}}$

The functionality runs with parties $P_1, \ldots, P_n$ and is parametrized by a key generator gen, an algorithm SingleToThresh for sharing a secret key with construction threshold n and a perfect hiding commitment scheme $(gencom, \text{commit})$.

generate:

If all honest parties inputs `init` in the same rounds, then run $(pk, sk) \leftarrow gen(k)$ and $(pv, sv_1, \ldots, sv_n) \leftarrow \text{SingleToThresh}(pk, sk)$ and output pv and sv_i, for all corrupted parties P_i, on the SOT. If P_i is corrupted later, then output sv_i on the SOT.

Generate a key-pair for the commitment scheme $(K, t) \leftarrow gencom(k)$.

Then for $i \in [n]$ do:

1. Compute a sharing of sv_i with construction threshold c. A sharing over a large enough prime field e.g., see Section 6.3.1. Denote the shares by $(sv_{i,1}, \ldots, sv_{i,n})$. We call $sv_{i,j}$ the backup share of sv_i given to server j.
2. For $j = 1, \ldots, n$, compute $c_{i,j} \leftarrow \text{commit}_K(sv_{i,j}; r_{i,j})$.
3. In a round specified by the adversary, output $((pv, K, \{c_{i,j}\}_{i,j\in[n]}), (sv_i, \{(sv_{j,i}, r_{j,i})\}_{j\in[n]}))$ to P_i.
   For corrupted parties these value are output on the SOT as soon as they are generated.

incorrect inputs:

If in the first round were an honest party inputs a non-trivial value some honest party do not input `init`, then break down. Also break down if an honest party inputs `init` twice or any other value than `init`.

Figure 10.1: The functionality $\mathcal{F}_{\text{key-dist}}^{\texttt{SIP}}$ for distributing a threshold key for gen, with construction threshold n and with backup shares.

reconstruct the needed secret values. Since we do not know the secret values sv_s of the single inconsistent party P_s, we simply share some dummy value, $sv_s = 0$ say, instead. Since an environment corrupting at most t parties only sees t of these shares and the construction threshold is $c = t + 1$, this will go unnoticed as long as P_s is not corrupted. To prevent corrupted parties from contributing wrong backup shares when reconstructing the secret value of a cheater we make a verifiable secret sharing. Since we are not going to address proactive security here, we do not have to concern ourself with what particular verifiable secret sharing scheme is used. We therefore simply commit the parties to their shares with any perfect hiding commitment. This gives us the key distribution ideal functionality in Fig. 10.1. Let $\text{SingleToThresh}^{\texttt{SIP}}$ denote the algorithm which given (pk, sk) generates values as in Fig. 10.1, with $pv' = (pv, K, \{c_{i,j}\}_{i,j\in[n]})$ and $sv_i' = (sv_i, \{(sv_{j,i}, r_{j,i})\}_{j\in[n]})$.

We have to settle for a mechanism for reconstructing parties that are caught cheating. What we will do is to let the parties broadcast their evaluation shares and then give individual proofs to each party that the share is correct. All parties which do not accept the proof from P_j then sends their backup share of sv_j. There are two possibilities then: Either some honest party accepts the proof it got from P_j or all honest parties reject the proof. In the last case

Protocol $\pi_{\text{THE}}^{\mathcal{THE},\texttt{SIP},c}$

The protocol runs in the $(\mathcal{F}_{\text{ZK-PM}}^{R}, \mathcal{F}_{\text{key-dist}}^{gen,\text{SingleToThresh},c,\texttt{SIP}}, \mathcal{F}_{\text{broadcast}})$-hybrid model with parties $P_1, \ldots, P_n$ and is parametrized by a generator gen and a construction threshold $0 < c \leq n$. We assume that $\mathcal{F}_{\text{ZK-PM}}^{R}$ has a round complexity of $r-1$ per proof. Party P_i proceeds as follows:

key generation:

On input `init` the party P_i inputs `init` to $\mathcal{F}_{\text{ZK-PM}}^{R}$ and $\mathcal{F}_{\text{key-dist}}^{gen,\text{SingleToThresh},c,\texttt{SIP}}$ and waits until $\mathcal{F}_{\text{ZK-PM}}^{R}$ has output `ready` and $\mathcal{F}_{\text{key-dist}}^{gen,\text{SingleToThresh},c,\texttt{SIP}}$ has output $((pv, K, \{c_{i,j}\}_{i,j\in[n]}), (sv_i, \{(sv_{j,i}, r_{j,i})\}_{j\in[n]}))$. Then output pk.

decryption:

1. On input (did, x) the server P_i computes the evaluation share $y_i \leftarrow sf(pv, sv_i, x)$ and broadcasts (did, y_i).
2. Let y_j be the value broadcast by P_j.
   Then P_i inputs $((did, i), P_j, (pv, x, i, y_i), sv_i)$ to $\mathcal{F}_{\text{ZK-PM}}^{R}$ for $j \in [n]$ and inputs $((did, j), P_j, (pv, x, j, y_j))$ to $\mathcal{F}_{\text{ZK-PM}}^{R}$ for $j \in [n]$.
3. Then wait for an output of the form $((did, j), b_j)$.
4. For each P_j where $b_j = 0$, if not done previously, send $(\texttt{backup}, j, sv_{j,i}, r_{j,i})$ to all parties.
5. If a value $(\texttt{backup}, j, sv_{j,l}, r_{j,l})$ is received from P_l and $c_{j,l} = \text{commit}_K(sv_{j,l}; r_{j,l})$, then add $(l, sv_{j,l})$ to a set BACKUP(j), if there was not already a value of the form (l, sv') in BACKUP(j). These sets are initially empty. If during this it happens that $|\text{BACKUP}(j)| = c$, then let $\{(l, sv_{l,j})\}_{l\in L}$ be the shares in BACKUP(j) and use these to reconstruct sv_j.
6. Then for all $j \in [n]$, if sv_j is known, compute $y_j \leftarrow sf(pv, sv_j, x)$; Otherwise use the y_j value received from P_j in Step 2. This gives values $\{y_j\}_{j\in[n]}$. Then output $(did, \text{Combine}(\{y_j\}_{j\in[n]}))$.

Figure 10.2: The SIP threshold protocol for $\mathcal{THE}$ with construction threshold c.

at least c honest parties send they backup share of sv_j to all parties, meaning that the parties can reconstruct sv_j and compute a correct evaluation share. In the first case, the broadcast evaluation share y_j is correct, so the parties can use y_j. In other words, if a party cannot reconstruct sv_j then it knows that the broadcast evaluation share y_j is correct. This gives us the realization in Fig. 10.2. We denote the protocol in decryption by $\pi_{\text{dec}}^{\texttt{SIP}}$.

We sketch how to simulate this protocol with a single inconsistent party P_s and corruption threshold $t = c - 1$. I.e. given all public values and the internal state of any $C \subseteq [n] \setminus \{s\}$ with $|C| = t$, these values are distributed statistically close to the real protocol. We show this by giving the necessary extensions to $\mathcal{S}_{\text{keydist}}$ and $\mathcal{S}_{\text{eval}}$. We first describe an extension $\mathcal{S}_{\text{keydist}}{}^{\texttt{SIP}}$ to $\mathcal{S}_{\text{keydist}}$: Given pk we first run $\mathcal{S}_{\text{keydist}}$ with $C = [n] \setminus \{s\}$. Notice that except for the values $(\{(sv_{j,i}, r_{j,i})\}_{j\in[n]}, \{c_{i,j}\}_{i,j\in[n]})$, $\mathcal{S}_{\text{keydist}}$ already simulates all the necessary values with a single inconsistent party, P_s. We show how we can simulate the values $(\{(sv_{j,i}, r_{j,i})\}_{j\in[n]}, \{c_{i,j}\}_{i,j\in[n]})$: Simply let $sv_s = 0$ and then generate all the values $(\{(sv_{j,i}, r_{j,i})\}_{j\in[n]}, \{c_{i,j}\}_{i,j\in[n]})$ for $i \in [n]$ as done in Fig. 10.1 on the preceding page. These

Protocol π_{ABABB}

The protocol runs in the $(\mathcal{F}^{R}_{\text{ZK-PM}}, \mathcal{F}^{\texttt{SIP}}_{\text{key-dist}}, \mathcal{F}_{\text{broadcast}}, \mathcal{F}^{\text{commit}}_{\text{PRS}})$-hybrid model with parties $P_1, \ldots, P_n$. It proceeds as follows:

initialize:

On input `init`, input `init` to $\mathcal{F}^{R}_{\text{ZK-PM}}$, $\mathcal{F}^{\texttt{SIP}}_{\text{key-dist}}$, and $\mathcal{F}_{\text{broadcast}}$ and request the key of all parties from $\mathcal{F}^{\text{commit}}_{\text{PRS}}$. Wait for a value (pv', sv'_i) from $\mathcal{F}_{\text{key-dist}}$ and the value `ready` from $\mathcal{F}^{R}_{\text{ZK-PM}}$ and $\mathcal{F}_{\text{broadcast}}$. The value pv contains a public-key N for the encryption scheme. We assume that $\mathcal{F}_{\text{key-dist}}$ at the same time outputs two random encryptions $K = E_{pk}(1)$ and $R = E_{pk}(0)$, also on the SOT. The key K will be used as the key L in Fig. 9.2 on page 291 and the key R will also be used as L, but for committing to a special value in the output command. We sometimes denote K by L when discussing the commit protocol in Fig. 9.1 on page 289. Furthermore, wait for a key $\overline{K}_j$ from $\mathcal{F}^{\text{commit}}_{\text{PRS}}$ for $j \in [n]$; The key $\overline{K}_j$ which is used for committing to the K_1 value in the commit protocol. When all values are received, output `ready`. Until outputting `ready`, ignore all inputs but the first `init` input.

load:

1. For $j \neq i$ party P_i commits to s to P_j (using the commit protocol in Fig. 9.1 on page 289), with the modification that it uses the same value $c_L = \text{commit}_L(s; r_L)$ in all proofs, and instead of sending c_L individually to all parties, it broadcasts c_L. Let $(K_j, c_{2,j}, r_{2,j})$ denote the commitment under the flipped key K_j in the commitment to P_j. I.e. $c_{2,j} = \text{commit}_{\overline{K}_j}(s; r_{2,j})$.
2. Then P_i proves to P_j that c_L and $c_{2,j}$ commit to the same value. The instance is $x = (N, L, c_L, K_j, c_{2,j})$ and the witness is $(s, r_L, s, r_{2,j})$.
3. Then the parties run a BA to determine whether the proof should be accepted, each inputting its output value from the proof.
4. If the BA outputs 1, then let $\text{enc}(x) = c_L$ and then output $(\texttt{defined}, x)$.

linear combination:

Set $\text{enc}(x) = a_0 \boxplus (\boxplus_{j=1}^{l} a_j \boxdot \text{enc}(x_j))$ and output $(\texttt{defined}, x)$.

Figure 10.3(a) (cont. in Fig. 10.3(b) on the next page): The Adaptive secure realization of the Basic Arithmetic Black-Box

values are clearly perfectly simulated except for those where sv_s enter in the computation. Of those $\{c_{s,j}\}_{j\in[n]}$ are public, and furthermore the adversary learns $\{(sv_{j,i}, r_{j,i})\}_{j\in C}$, where $|C| = t$ and $s \notin C$. Since the values $\{sv_{j,i}\}_{i\in[n]}$ are a sharing of $sv_s = 0$ with construction threshold $c = t + 1$ it follows that when $|C| = t$, then the values $\{sv_{j,i}\}_{j\in C}$ are distributed exactly as if sv_s had had any other value, especially the right one. Furthermore, the values $\{c_{s,j}\}_{i\in[n]}$ are all perfectly hiding commitments and therefore contain no information about sv_s. As for $\mathcal{S}_{\text{eval}}$, it already simulates the necessary values, perfectly. Therefore, what remain is to make a trivial extension to $\mathcal{S}_{\text{eval}}$ which simulates the step where evaluation shares are broadcast and commitments to backup shares are opened. This is done following the protocol. We call the result ${\mathcal{S}_{\text{dec}}}^{\texttt{SIP}}$.

Protocol π_{ABABB}

private multiplication:

We assume that P_i knows (s, r_s) such that $\text{enc}(x_1) = \text{commit}_K(s; r_s)$.

1. P_i computes $X = \text{commit}_{\text{enc}(x_2)}(s; r_X) = \text{Blind}_{pk}(s \boxdot \text{enc}(x_2); r_X)$ and broadcasts X.
2. For $j \neq i$ party P_i proves to P_j that this was done correctly, with instance $x = (N, K, \text{enc}(x_1), \text{enc}(x_2), X)$ and witness (s, r_S, s, r_X).
3. Then the parties run a BA to determine whether the proof should be accepted.
4. If the BA outputs 1, then set $\text{enc}(x) = X$ and output $(\texttt{defined}, x)$.

output:

1. Party P_i generates random $r_i \in R_N$ and loads it into variable x_i. Let $(K_j, c_{2,i}, r_{2,i})$ denote the commitment used in the load.
2. Party P_i computes $\text{enc}(y_i) \leftarrow \text{commit}_R(r_i; \text{rnd}(y_i))$ and broadcasts $\text{enc}(y_i)$.
3. Then P_i proves to all P_j that $\text{enc}(y_i)$ commits to the same value as $c_{2,i}$. The instance is $x = (N, K, \text{enc}(x_i), K_i, c_{2,i})$ and the witness is $(r_i, \text{rnd}(x_i), r_{2,i})$.
4. Then the parties run a BA to determine whether the proof should be accepted.
5. Let J_2 be the indices of P_j having the proof accepted. In some fixed way, pick a subset $I \subseteq J_2$ where $|I| = t+1$. E.g. the $t+1$ lowest indices. Since all honest parties are participating we have that we can indeed pick such a subset. Then compute $S = \boxplus_{i \in I}(\lambda_{i,I}(0) \boxdot \text{enc}(y_i))$ and $T = \text{enc}(x) \boxplus S$, where $\lambda_{i,I}(0)$ is the Lagrange coefficient defined in Eq. 6.4 on page 199, with all computations done in $\mathcal{M}$.[a]
6. The parties run the $\pi^{\texttt{SIP}}_{\text{dec}}$ protocol to decrypt T and take as their output the value v returned by $\pi^{\texttt{SIP}}_{\text{dec}}$.

[a] This is well-defined as we required that we can interpolation through the homomorphism in Definition 10.3 on page 310.

Figure 10.3(b) (cont. from Fig. 10.3(a) on the page before): The Adaptive secure realization of the Basic Arithmetic Black-Box

10.5 The Protocol

In this section we present a realization of $\mathcal{F}_{\text{BABB}}$ from any adaptive ABB threshold homomorphic encryption scheme. The realization is adaptive secure against an adversary corrupting at most $(n-1)/2$ parties.

We present the protocol in the $(\mathcal{F}^R_{\text{ZK-PM}}, \mathcal{F}^{\texttt{SIP}}_{\text{key-dist}}, \mathcal{F}_{\text{broadcast}}, \mathcal{F}^{\text{commit}}_{\text{PRS}})$-hybrid model, where R is the relation `equality of committed value` from Definition 10.3 on page 310. We then plug in the realization of $\mathcal{F}^R_{\text{ZK-PM}}$ from Fig. 5.1 on page 180. Needless to say that the trapdoor commitment scheme needed by the realization in Fig. 5.1 on page 180 can be constructed based on the assumptions that we have already made.

Theorem 10.1 *Let $\mathcal{THE} = (gen, E, D)$ be an adaptive ABB threshold homomorphic encryption scheme. Let $\pi^{\Sigma}_{\text{DAM-ZK}}$ be the realization of $\mathcal{F}^R_{\text{ZK-PM}}$ from Fig. 5.1 on page 180. Let*

$c = \lceil (n-1)/2 \rceil$. *The protocol* $\pi^{\mathcal{THE}}_{\text{ABABB}}[\pi^{\Sigma}_{\text{DAM-ZK}}/\mathcal{F}^{R}_{\text{ZK-PM}}, \pi^{SIP,c}_{\text{THE}}/\mathcal{F}^{SIP,c}_{\text{THE}}]$ $(c-1)$*-realizes* $\mathcal{F}^{gen^{pk}}_{\text{BABB}}$ *in the* $(\mathcal{F}^{SIP,c}_{\text{key-dist}}, \mathcal{F}_{\text{broadcast}}, \mathcal{F}^{\text{commit}}_{\text{PRS}})$*-hybrid model.*

Proof. In the following we use $\mathcal{F}_{\text{BABB}}$ to denote $\mathcal{F}^{gen^{pk}}_{\text{BABB}}$ and we use π_{ABABB} to denote $\pi^{\mathcal{THE}}_{\text{ABABB}}[\pi^{\Sigma}_{\text{DAM-ZK}}/\mathcal{F}^{R}_{\text{ZK-PM}}, \pi^{\texttt{SIP},c}_{\text{THE}}/\mathcal{F}^{\texttt{SIP},c}_{\text{THE}}]$.

We prove the theorem by giving an interface $\mathcal{S}_{\text{ABABB}}$ simulating π_{ABABB} in the ideal process $\text{IDEAL}^{\mathcal{F}^{\texttt{SIP},c}_{\text{key-dist}}, \mathcal{F}_{\text{broadcast}}, \mathcal{F}^{\text{commit}}_{\text{PRS}}}_{\mathcal{F}_{\text{BABB}}, \mathcal{S}_{\text{ABABB}}, \mathcal{Z}}$. The simulator will run a copy of the protocol π_{ABABB} and carry out the environment's commands on this copy. It keeps it consistent with the inputs and outputs that $\mathcal{F}_{\text{BABB}}$ has in $\text{IDEAL}^{\mathcal{F}^{\texttt{SIP},c}_{\text{key-dist}}, \mathcal{F}_{\text{broadcast}}, \mathcal{F}^{\text{commit}}_{\text{PRS}}}_{\mathcal{F}_{\text{BABB}}, \mathcal{S}_{\text{ABABB}}, \mathcal{Z}}$. In simulating π_{ABABB}, $\mathcal{S}_{\text{ABABB}}$ must provide inputs to $\mathcal{F}_{\text{BABB}}$ on behalf of the corrupted parties. By specification of $\mathcal{F}_{\text{BABB}}$ the inputs from $\mathcal{S}_{\text{ABABB}}$ do not matter in most cases. For all commands except load and private multiplication the input on the SIT is not considered by $\mathcal{F}_{\text{BABB}}$. However, for a load $(P_i, x \leftarrow S)$ and a private multiplication $(P_i, x \leftarrow x_1 \cdot x_2)$ the input from $\mathcal{S}_{\text{ABABB}}$ matters if P_i is corrupted. In the first case $\mathcal{S}_{\text{ABABB}}$ must provide an input $(P_i, x \leftarrow s)$ for x to be defined and in the second case $\mathcal{S}_{\text{ABABB}}$ must provide the input $(P_i, x \leftarrow x_1 \cdot x_2)$ for x to be defined (to $\text{val}(x_1) \cdot \text{val}(x_2)$). Whether the simulator supplies these inputs, and if it does, the value of s, will depend on the messages requested delivered by $\mathcal{Z}$ on behalf of corrupted parties — if $\mathcal{Z}$ lets P_i participate in π_{ABABB} in a way which results in the honest parties defining $\text{enc}(x)$ to hold some encryption, then $\mathcal{S}_{\text{ABABB}}$ will provide the input to $\mathcal{F}_{\text{BABB}}$ and in the case of a load it will determine an appropriate value of s.

The simulator will mostly be build out of parts we already constructed. It will be using the simulator $\mathcal{S}_{\text{COM-PR}}$ from the proof of Theorem 9.1 on page 291 for simulating the commitment protocol, and will also be using the simulator $\mathcal{S}_{\text{DAM-ZK}}$ for the protocol $\pi_{\text{DAM-ZK}}$ for simulating the proofs of knowledge. It uses SingleToThresh$^{\texttt{SIP}}$ for simulating a key distribution given a secret key to hit. Later, when specifying hybrids we will also be using the simulator $\mathcal{S}_{\text{keydist}}{}^{\texttt{SIP}}$, for simulating a key distribution given just the public key, and the simulator $\mathcal{S}_{\text{dec}}{}^{\texttt{SIP}}$, for simulating the decryption protocol given a ciphertext and its plaintext value. When using $\mathcal{S}_{\text{dec}}{}^{\texttt{SIP}}$ we will not make all the input values explicit and we will not input a set $C \subset [n]$ of size $n-1$ as required in the definition in Definition 6.2 on page 194. The input values will be given by the context, and we represent C by $s \in [n]$, where $\{s\} = [n] \setminus C$. In the description of the simulator marks of the form **(sk***i***)** occur. These mark places where the private key is used by the simulator and is for easy reference in the analysis. The simulator is given in Fig. 10.4 on the following page.

It is straight-forward to verify that the protocol uses the ideal functionalities of the hybrid model correctly. Using the OSIP lemma, Lemma 10.1 on page 307, it is therefore enough to prove that

$$\text{IDEAL}^{\mathcal{F}^{\texttt{SIP},c}_{\text{key-dist}}, \mathcal{F}_{\text{broadcast}}, \mathcal{F}^{\text{commit}}_{\text{PRS}}}_{\mathcal{F}_{\text{BABB}}, \mathcal{S}_{\text{ABABB}}, \mathcal{Z}^{\texttt{SIP}}} \overset{\text{c}}{\approx} \text{HYB}^{\mathcal{F}^{\texttt{SIP},c}_{\text{key-dist}}, \mathcal{F}_{\text{broadcast}}, \mathcal{F}^{\text{commit}}_{\text{PRS}}}_{\pi_{\text{ABABB}}, \mathcal{Z}^{\texttt{SIP}}}$$

for all environments $\mathcal{Z}$ where $\Pr[\text{IDEAL}^{\mathcal{F}^{\texttt{SIP},c}_{\text{key-dist}}, \mathcal{F}_{\text{broadcast}}, \mathcal{F}^{\text{commit}}_{\text{PRS}}}_{\mathcal{F}_{\text{BABB}}, \mathcal{S}_{\text{ABABB}}, \mathcal{Z}^{\texttt{SIP}}} \in \text{``H} \rightsquigarrow \mathcal{F}_{\text{BABB}}\text{''}] \overset{\text{c}}{\approx} 0$. In

Interface $\mathcal{S}$

The interface proceeds as follows:

initialize:

In the initialization the simulator simulates the ideal functionalities as follows: It receives $(pk = N, sk = t_N)$ from $\mathcal{F}_{\text{BABB}}$. To simulate $\mathcal{F}_{\text{key-dist}}^{\text{SIP},c}$ it computes $(pv', sv'_1, \ldots, sv'_n) = \text{SingleToThresh}^{\text{SIP}}(pk, sk)$.**sk1** Then it computes $K \leftarrow E_{pk}(0; t_K)$ and $R \leftarrow E_{pk}(1; r_R)$. Then it simulates the output (sk'_i, pv', K, R) to party P_i and simulates output (pv', K, R) on the SOT of $\mathcal{F}_{\text{key-dist}}^{\text{SIP},c}$. It simulate $\mathcal{F}_{\text{PRS}}^{\text{commit}}$ as in initializing in $\mathcal{S}_{\text{COM-PR}}$.

load:

On the value $(P_i, x \leftarrow R_{pk})$ from $\mathcal{F}_{\text{BABB}}$, if P_i is honest we proceed as follows:

1. The commitments are simulated as in commit in $\mathcal{S}_{\text{COM-PR}}$. If P_i is later corrupted, then we learn s from $\mathcal{F}_{\text{BABB}}$ and open the commitments to this value.
2. The proof that c_L commits to the same value as $c_{2,j}$ is simulated as in proof of relation in $\mathcal{S}_{\text{COM-PR}}$.

For corrupted P_i the simulator inputs $(P_i, x \leftarrow 0)$ to $\mathcal{F}_{\text{BABB}}$ on behalf of P_i and then simulates the honest parties by following the protocol. If any of the simulated honest parties output $(\texttt{defined}, x)$, then the simulator must at the end of the load input $(\texttt{change}, s)$ on the SIT of $\mathcal{F}_{\text{BABB}}$ to define x. The value of s is determined as follows: Since the honest party output $(\texttt{defined}, x)$, the BA must have output 1. This means that some honest party P_j input 1 to the BA meaning that P_j accepted the proof of plaintext knowledge. Let $(K_{i,j}, c_{2,i,j})$ be the commitment received by P_j from P_i. Decrypt$^{\textbf{sk2}}$ this commitment as in receiving a commitment in $\mathcal{S}_{\text{COM-PR}}$. If we end up in case R.2(c) in $\mathcal{S}_{\text{COM-PR}}$, then we terminate and output "coin-flip failed".

In all cases, if $\text{enc}(x)$ is accepted, the simulator uses E-trapdoor extraction$^{\textbf{sk3}}$ to learn random bits $\text{rnd}(x)$ such that $\text{enc}(x) = \text{commit}_K(0; \text{rnd}(x))$.

linear combination:

Compute $\text{enc}(x)$ as in the protocol, and using that the scheme is E-trapdoor preserving, compute $\text{rnd}(x)$ such that $\text{enc}(x) = E_{pk}(0; \text{rnd}(x))$.

private multiplication:

All honest parties P_j for $j \neq i$ just follow the protocol. If P_i is honest it is simulated as follows:

1. P_i computes $X = \text{commit}_K(0; r_0)$ and broadcasts X.
   If P_i is corrupted after this step the simulator learns s. Using $\text{rnd}(x_2)$ and r_0 it uses equivocability to compute random bits r_X such that $X = \text{commit}_{\text{enc}(x_2)}(s; r_X) = \text{Blind}_{pk}(s \boxdot \text{enc}(x_2); r_X)$ and uses r_X as the internal state of P_i.

Figure 10.4(a) (cont. in Fig. 10.4(b) on the next page): The interface $\mathcal{S}$ used in the proof of Theorem 10.1 on page 314

Interface $\mathcal{S}$

private multiplication (cont.):

2. For each party P_j where $j \neq i$ party P_i runs $\mathcal{S}_{\text{DAM-ZK}}$ with instance $x = (N, K, \text{enc}(x_1), \text{enc}(x_2), X)$.

   If P_i is corrupted after this step, then as x_1 was loaded by P_i we learn from the patching of the load a value r_S such that $\text{enc}(x_1) = \text{commit}_K(s; r_S)$ and in Step 1 we computed r_X such that $X = \text{commit}_{\text{enc}(x_2)}(s; r_X) = \text{Blind}_{pk}(s \boxdot \text{enc}(x_2); r_X)$. Then $w = (s, r_S, r_X)$ is a witness to the instance x. Give this witness to $\mathcal{S}_{\text{DAM-ZK}}$ to patch the state of the proof of correct multiplication.

3. Then follow the protocol, and if $\text{enc}(x)$ is defined, then let $\text{rnd}(x) = r_0$.

output:

On input $(\texttt{output}, x = v)$ from $\mathcal{F}_{\text{BABB}}$ we know that all honest parties got the input $(\texttt{output}, x)$. Notice that the commitment $\text{enc}(x) \in E_{pk}(0)$, so we cannot just decrypt $\text{enc}(x)$ using $\pi^{\texttt{SIP}}_{\text{dec}}$. This is where the randomization of $\text{enc}(x)$ will come into play. It will allow us to make T an encryption of v, so that we can 'simulate' the decryption protocol by running it honestly:

1. Simulate the load commands as specified in load in $\mathcal{S}_{\text{COM-PR}}$, using for P_i as the dummy value m' and the second dummy value m'' a uniformly random value $m' = m'' = r'_i$. If a party P_i is corrupted, then let $r_i = r'_i$, so that the internal state of P_i need no patching.[a] As for the load command, learn the values r_i loaded by the corrupted parties.

2. Let J_1 be the indices j of corrupted parties for which the load into x_j succeeded.

   If $|J_1| < t$, then add the indices of some uniformly random honest parties until $|J_1| = t$ and for those honest parties now fix $r_i = r'_i$. Then pick a random degree $t+1$ polynomial $f(X) \in \mathcal{M}_N[X]$ such that $f(j) = r_j$ for $j \in J_1$ and $f(0) = v$. Then for the remaining honest parties set $r_i = f(i)$ and patch the load into x_i done in the previous step to be consistent with r_i.

   Then proceed according to the protocol — this is possible as the previous load is now consistent with r_i and so all witnesses are known.

3. This step is simulated by following the protocol. Notice that R is an encryption of 1, so unless a corrupted party P_j was able to load different value into x_j and y_j, including giving a proof of a false instance, we will have that $\text{enc}(y_i)$ is an encryption of r_i for all parties P_i. Therefore $S = \boxplus_{j \in I}(\lambda_{j,I}(0) \boxdot \text{enc}(y_j)) \in E_{pk}(\sum_{j \in I} \lambda_{j,I}(0) f(j)) = E_{pk}(f(0)) = E_{pk}(v)$. Therefore $T \in E_{pk}(v)$ as $\text{enc}(x)$ is an encryption of 0.**sk4**

4. Run $\pi^{\texttt{SIP}}_{\text{dec}}$ as in the protocol.

[a] This is done as not to use the E-trapdoor of the key K_i so that we can reduce to the computational binding of commit_{K_i}.

Figure 10.4(b) (cont. from Fig. 10.4(a) on the preceding page): The interface $\mathcal{S}$ used in the proof of Theorem 10.1 on page 314

the proof, seven distribution ensembles $H^1, \ldots, H^7$ are specified and it is then proved that

$$\mathrm{IDEAL}^{\mathcal{F}_{\text{key-dist}}^{\mathtt{SIP},c},\mathcal{F}_{\text{broadcast}},\mathcal{F}_{\text{PRS}}^{\text{commit}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{ABABB}},\mathcal{Z}} \overset{\text{s}}{\approx} H^1 \overset{\text{c}}{\approx} H^2 \overset{\text{c}}{\approx} H^3 = H^4 \overset{\text{c}}{\approx} H^5 \overset{\text{c}}{\approx} H^6 \overset{\text{c}}{\approx} H^7 \overset{\text{s}}{\approx} \mathrm{HYB}^{\mathcal{F}_{\text{key-dist}}^{\mathtt{SIP},c},\mathcal{F}_{\text{broadcast}},\mathcal{F}_{\text{PRS}}^{\text{commit}}}_{\pi_{\text{ABABB}},\mathcal{Z}^{\mathtt{SIP}}}} .$$

As opposed to the sequence of hybrids in the proof of Theorem 9.1 on page 291 we cannot take as the first hybrid the one where we start supplying correct inputs to the commitment scheme from all honest parties. This is because we also use the commitment scheme in the first step of the `output` protocol to load r_i under R, which is an encryption of 1, so we cannot just start following the protocol and supplying uniformly random r_i values on behalf of corrupted parties before we get to the hybrid where sk is not used anymore — it requires semantic security to argue that this is not noticed by the environment. Therefore, what we will do is to get rid of the use of sk, then let R be an encryption of 0 so that both K and R are encryptions of 0, and then do the hybrids from the proof of Theorem 9.1 on page 291.

Define the first hybrid H^1 by running $\mathrm{IDEAL}^{\mathcal{F}_{\text{key-dist}}^{\mathtt{SIP},c},\mathcal{F}_{\text{broadcast}},\mathcal{F}_{\text{PRS}}^{\text{commit}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{ABABB}},\mathcal{Z}^{\mathtt{SIP}}}$ with the modification that the simulators ${\mathcal{S}_{\text{keydist}}}^{\mathtt{SIP}}$ and ${\mathcal{S}_{\text{dec}}}^{\mathtt{SIP}}$ are used instead of $\text{SingleToThresh}^{\mathtt{SIP}}$ and $\pi_{\text{dec}}^{\mathtt{SIP}}$. This is done by running ${\mathcal{S}_{\text{keydist}}}^{\mathtt{SIP}}(pk, s)$ at **(sk1)** to produce $(pv', sv'_1, \ldots, sv'_n)$ and running ${\mathcal{S}_{\text{dec}}}^{\mathtt{SIP}}(T, D_{sk}(T), s)$ at **(sk4)** instead of $\pi_{\text{dec}}(T)$. Here s is the off-line single inconsistent party defined by $\mathcal{Z}^{\mathtt{SIP}}$. Doing this we got rid of the use of $sk = t_N$ at **(sk1)**, but introduced a new use of sk at **(sk4)**. By construction of ${\mathcal{S}_{\text{keydist}}}^{\mathtt{SIP}}(pk, s)$ and ${\mathcal{S}_{\text{dec}}}^{\mathtt{SIP}}(T, D_{sk}(T), s)$ we have that $\mathrm{IDEAL}^{\mathcal{F}_{\text{key-dist}}^{\mathtt{SIP},c},\mathcal{F}_{\text{broadcast}},\mathcal{F}_{\text{PRS}}^{\text{commit}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{ABABB}},\mathcal{Z}^{\mathtt{SIP}}} \overset{\text{s}}{\approx} H^1$.

We define a second hybrid H^2 by running H^1 with the modification that t_N is not used anymore. Instead, each time a proof has been given, we use $\mathcal{S}_{\text{DAM-ZK}}$ to extract a witness for the proof. This rids of the use of the secret key $sk = t_N$ at **(sk2)** as P_i gave a proof to P_j that $\mathrm{enc}(x)$ and $c_{2,j}$ commit to identical values, so indeed the witness contains openings of $\mathrm{enc}(x) = c_L$ and $c_{2,j}$, as needed in the simulation, including the random bits. If this is not an opening to 0, as required at the end of `open`, then equivocability can be used to compute an opening to 0 as needed. To do the extraction we note that H^1 is of the form $\mathrm{IDEAL}^{\mathcal{F}_{\text{key-dist}}^{\mathtt{SIP},c},\mathcal{F}_{\text{broadcast}},\mathcal{F}_{\text{PRS}}^{\text{commit}}}_{\mathcal{F}_{\text{BABB}},\mathcal{S}_{\text{ABABB}}'[\mathcal{S}_{\text{DAM-ZK}}],\mathcal{Z}^{\mathtt{SIP}}}$ and that $\mathcal{S}_{\text{ABABB}}{}'$ only uses $\mathcal{S}_{\text{DAM-ZK}}$ to simulate instances that are in the relation. This follows from the fact that L and all flipped K_j keys for honest parties are E-keys and that the load under the X-key R in Step 2 in `output` is done with the correct values. Using the techniques used to prove Eq. 8.2 on page 277 and Eq. 8.3 on page 278 we can then argue that we can indeed extract witnesses except with negligible probability. In addition, we start running ${\mathcal{S}_{\text{dec}}}^{\mathtt{SIP}}(T, v, s)$ instead of ${\mathcal{S}_{\text{dec}}}^{\mathtt{SIP}}(T, D_{sk}(T), s)$ at **(sk4)**. To see that $H^2 \overset{\text{c}}{\approx} H^1$, observe that we can prove that the probability that "coin-flip broken" is output (respectively should have been output) in H^1 (respectively H^2) is negligible and that the probability that some witness is not extracted is negligible, following the proof of Theorem 9.1 on page 291. If therefore $H^2 \overset{\text{c}}{\not\approx} H^1$, then $D_{sk}(T) \neq v$ with a non-negligible probability. We argue that this contradicts the fact that we extract witnesses except with negligible probability. To see this, consider a simulation of `output`. Assume that from the loads into x_j we have witnesses $(r_j, \mathrm{rnd}(x_j), r_{2,j})$ for which $\mathrm{enc}(x_j) = \mathrm{commit}_K(r_j; \mathrm{rnd}(x_j))$

and $c_{2,j} = \text{commit}_{K_j}(r_j; r_{2,j})$ for all parties P_j. By construction of Step 2 in output we have that $v = \sum_{j \in I} \lambda_{j,I}(0) r_j$. Assume furthermore that from the loads into y_j we know witnesses $(r'_j, \text{rnd}(x'_j), r'_{2,j})$ for which $\text{enc}(y_j) = \text{commit}_R(r'_j; \text{rnd}(x_j)')$ and $c_{2,j} = \text{commit}_{K_j}(r'_j; r'_{2,j})$ for all P_j. Since $R \in E(1)$ it follows that $T \in E_{pk}(\sum_{j \in I} \lambda_{j,I}(0) r'_j)$. So, except with negligible probability we have that $\sum_{j \in I} \lambda_{j,I}(0) r'_j \neq \sum_{j \in I} \lambda_{j,I}(0) r_j$, which implies that $r_j \neq r'_j$ for some $j \in I$. Clearly $r_j = r'_j$ for all $j \in I$ where P_j is still honest when S is computed. Second, for the parties which were corrupted before the load into x_j we have that K_j is an X-key except with negligible probability. Therefore, $r_j \neq r'_j$ for a party which was honest when the $\mathtt{com}_1$-message of the protocol for generating K_j was sent. So, we did not use the trapdoor of K_j in output.[6] Therefore we computed a double-opening of $c_{2,j}$ under a random E-key K_j for which we did not use the trapdoor. We can easily show that this is in contradiction with the computational binding of the E-keys.[7]

We define a third hybrid H^3 by running H^2 with the modification that we use $R = E_{pk}(0)$ instead of $R = E_{pk}(1)$. This is possible because the simulator does not use R as anything but a key. Since we do not use the secret key in generating H^2 or H^3 it follows via IND-CPA security that $H^3 \overset{c}{\approx} H^2$.

The fourth hybrid H^4 is defined by running H^3 except that we use correct inputs to all honest parties and pick the r_i values uniformly at random for all honest parties, i.e. without patching r_i to $f(i)$. Notice that this means that we provide known correct inputs to all runs of the commitment protocol. It can be seen that $H^4 = H^3$. This is so because $K, R \in E_{pk}(0)$, so no information is leaked about the inputs and because the environment sees at most t of the r_i values and so cannot distinguish uniformly random values from values consistent with a uniformly random polynomial of degree at most t.

Define a fifth hybrid H^5 by running H^4, but with $K = E_{pk}(1)$. This is possible as we are now providing correct inputs for all loads under K, including the loads into x_i in output. By the IND-CPA security of E_{pk} we have that $H^5 \overset{c}{\approx} H^4$.

Then define H^6 by running H^5 and starting to use sk at **(sk2)** again. As for $H^1 \overset{c}{\approx} H^2$ we get that $H^6 \overset{c}{\approx} H^5$.

Define H^7 as H^6 but at **(sk4)** run $\mathcal{S}_{\text{dec}}{}^{\mathtt{SIP}}(T, D_{sk}(T))$ instead of $\mathcal{S}_{\text{dec}}{}^{\mathtt{SIP}}(T, v)$. To prove that $H^7 \overset{c}{\approx} H^6$ it is sufficient to prove that $v = D_{sk}(T)$ except with negligible probability, which basically comes down to proving the protocol correct. This can be done using that except with negligible probability, witnesses can be extracted for all proofs and all keys are X-keys. This means that for all commitments entering several proofs, the extracted witnesses are the same. It is straightforward to verify that when all keys are X-keys and all witnesses are extracted, then $v = D_{sk}(T)$, which proves the claim. Notice that the argument does not need the computational binding of the commitments, as they are perfectly binding under the X-keys. This is essential as we are again using sk, so commitments under E-keys would not be guaranteed to be binding. Since we are again using sk we are also using essentially that the membership soundness of all the applied Σ-protocols is unconditional, so that in particular the soundness still holds when sk is known.

The only difference between H^7 and $\text{HYB}_{\pi_{\text{ABABB}}, \mathcal{Z}^{\mathtt{SIP}}}^{\mathcal{F}_{\text{key-dist}}^{\mathtt{SIP},c}, \mathcal{F}_{\text{broadcast}}, \mathcal{F}_{\text{PRS}}^{\text{commit}}}$ is between the use of

[6] See Footnote *a* in Fig. 10.4(b) on page 317.

[7] As we do not use $sk = t_N$ anymore.

$\mathcal{S}_{\text{keydist}}{}^{\mathtt{SIP}}(pk)$ instead of SingleToThresh$^{\mathtt{SIP}}$ and the use of $\mathcal{S}_{\text{dec}}{}^{\mathtt{SIP}}(T, D_{sk}(T))$ instead of $\pi_{\text{dec}}(T)$. So, $H^7 \overset{\text{s}}{\approx} \text{HYB}^{\mathcal{F}_{\text{PRS}}^{\text{commit}}, \mathcal{F}_{\text{key-dist}}^{\mathtt{SIP},c}, \mathcal{F}_{\text{broadcast}}}_{\pi_{\text{ABABB}}, \mathcal{Z}^{\mathtt{SIP}}}$ follows as $\text{IDEAL}^{\mathcal{F}_{\text{key-dist}}^{\mathtt{SIP},c}, \mathcal{F}_{\text{broadcast}}, \mathcal{F}_{\text{PRS}}^{\text{commit}}}_{\mathcal{F}_{\text{BABB}}, \mathcal{S}_{\text{ABABB}}, \mathcal{Z}^{\mathtt{SIP}}} \overset{\text{s}}{\approx} H^1$. □

Go on, get out. Last words are for fools who haven't said enough.
— Karl Marx

Bibliography

[And97] Ross Anderson. Two remarks on public key cryptology. `http://www.cl.cam.ac.uk/ftp/users/rja14/forwardsecure.pdf`, 1997. Invited Lecture, ACM-CCS.

[Awe85] Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, 1985.

[Bac88] Eric Bach. How to generate factored random numbers. *SIAM Journal on Computing*, 17(2):179–193, April 1988.

[BB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proc. ACM PODC'89*, pages 201–209, 1989.

[BBP03] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An un-instantiable random-oracle-model scheme for a hybrid-encryption problem. Obtainable from the Cryptology ePrint Archive, record 2003/077, `http://eprint.iacr.org/`, April 2003.

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation (extended abstract). In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 52–61, San Diego, California, 16–18 May 1993.

[BCK98] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 419–428, Dallas, TX, USA, 24–26 May 1998.

[BDGK95] Amotz Bar-Noy, Xiaotie Deng, Juan A. Garay, and Tiko Kameda. Optimal amortized distributed consensus. *Information and Computation*, 120(1):93–100, 1995.

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science*, Miami Beach, FL, 19–22 October 1997. IEEE.

[BE03] Michael Ben-Or and Ran El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16(4):249–262, 2003.

[Bea91] Donald Beaver. Secure multi-party protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.

[Bea96] Donald Beaver. Adaptive zero knowledge and computational equivocation. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 629–638, Philadelphia, Pennsylvania, 22–24 May 1996.

[Bea97] Donald Beaver. Plug and play encryption. In Burt Kaliski, editor, *Advances in Cryptology - Crypto '97*, pages 75–89, Berlin, 1997. Springer-Verlag. Lecture Notes in Computer Science Volume 1294.

[Ben83] Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, 1983.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 103–112, Chicago, Illinois, 2–4 May 1988.

[BG85] Manuel Blum and Shafi Goldwasser. An efficient probabilistic public key encryption scheme which hides all partial information. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology: Proceedings of Crypto '84*, pages 289–302, Berlin, 1985. Springer-Verlag. Lecture Notes in Computer Science Volume 196.

[BG89] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority (extended announcement). In *30th Annual Symposium on Foundations of Computer Science*, pages 468–473, Research Triangle Park, North Carolina, 30 October–1 November 1989. IEEE.

[BGP89] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consensus. In *30th Annual Symposium on Foundations of Computer Science*, pages 410–415, Research Triangle Park, North Carolina, 30 October–1 November 1989. IEEE.

[BGP92] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Optimal early stopping in distributed consensus. In *Proceedings of the sixth International Workshop on Distributed Algorithms*, pages 221–237, 1992.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, Illinois, 2–4 May 1988.

[BH92] Donald Beaver and Stuart Haber. Cryptographic protocols provably secure against dynamic adversaries. In Rainer A. Rueppel, editor, *Advances in Cryptology - EuroCrypt '92*, pages 307–323, Berlin, 1992. Springer-Verlag. Lecture Notes in Computer Science Volume 658.

[BKR94] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computation with optimal resilience. In *Proc. ACM PODC'94*, pages 183–192, 1994.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computing and Communications Security*, pages 62–73. ACM, 1993.

[BR95] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *Advances in Cryptology - EuroCrypt '94*, pages 92–111, Berlin, 1995. Springer-Verlag. Lecture Notes in Computer Science Volume 950.

[Bra84] Gabriel Bracha. An asynchronous $\lceil (n-1)/3 \rceil$-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162, 1984.

[Bra93] Stefan Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, CWI, Computer Science, Department of Algorithmics and Architecture, CWI, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands, 1993.

[BS96] Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory, Volume I: Efficient Algorithms*. MIT Press, August 1996.

[Can97] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Burt Kaliski, editor, *Advances in Cryptology - Crypto '97*, pages 455–469, Berlin, 1997. Springer-Verlag. Lecture Notes in Computer Science Volume 1294.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, winter 2000.

[Can01a] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science*. IEEE, 2001.

[Can01b] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Obtainable from the Cryptology ePrint Archive, record 2000/067, `http://eprint.iacr.org/`. Preliminary version appeared as [Can01a], October 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 11–19, Chicago, Illinois, 2–4 May 1988.

[CD98] Ronald Cramer and Ivan Damgaard. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free. In Hugo Krawczyk, editor, *Advances in Cryptology - Crypto '98*, pages 424–441, Berlin, 1998. Springer-Verlag. Lecture Notes in Computer Science Volume 1462.

[CDD+99] Ronald Cramer, Ivan Damgård, Stefand Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In Jacques Stern, editor, *Advances in Cryptology - EuroCrypt '99*, pages 311–326, Berlin, 1999. Springer-Verlag. Lecture Notes in Computer Science Volume 1592.

[CDD00] Ronald Cramer, Ivan Damgård, and Stefan Dziembowski. On the complexity of verifiable secret sharing and multiparty computation. In *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing*, pages 325–334, Portland, OR, USA, 21–23 May 2000.

[CDD+01] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. On adaptive vs. non-adaptive security of multiparty protocols. In *Advances in Cryptology - EuroCrypt 2001*, pages 262–279, Berlin, 2001. Springer-Verlag. Lecture Notes in Computer Science Volume 2045.

[CDG87] David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In Carl Pomerance, editor, *Advances in Cryptology - Crypto '87*, pages 87–119, Berlin, 1987. Springer-Verlag. Lecture Notes in Computer Science Volume 293.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *Advances in Cryptology - EuroCrypt 2000*, pages 316–334, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1807.

[CDN01] Ronald Cramer, Ivan Damgaard, and Jesper B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology - EuroCrypt 2001*, pages 280–300, Berlin, 2001. Springer-Verlag. Lecture Notes in Computer Science Volume 2045.

[CDNO97] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In Burt Kaliski, editor, *Advances in Cryptology - Crypto '97*, pages 90–104, Berlin, 1997. Springer-Verlag. Lecture Notes in Computer Science Volume 1294.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *Advances in Cryptology - Crypto '94*, pages 174–187, Berlin, 1994. Springer-Verlag. Lecture Notes in Computer Science Volume 839.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In J. Kilian, editor, *Advances in Cryptology - Crypto 2001*, pages 19–40, Berlin, 2001. Springer-Verlag. Lecture Notes in Computer Science Volume 2139.

[CF02] Ronald Cramer and Serge Fehr. Optimal black-box secret sharing over arbitrary abelian groups. In M. Yung, editor, *Advances in Cryptology - Crypto 2002*, pages 272–287, Berlin, 2002. Springer-Verlag. Lecture Notes in Computer Science Volume 2442.

[CFGN96] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 639–648, Philadelphia, Pennsylvania, 22–24 May 1996.

[CFS+03] S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu. RFC 3647 - internet X.509 public key infrastructure certificate policy and certification practices framework, November 2003.

[CG96] Ran Canetti and Rosario Gennaro. Incoercible multiparty computation (extended abstract). In *37th Annual Symposium on Foundations of Computer Science*, pages 504–513, Burlington, Vermont, 14–16 October 1996. IEEE.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 209–218, Dallas, TX, USA, 24–26 May 1998.

[CGJ+99] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael Wiener, editor, *Advances in Cryptology - Crypto '99*, pages 98–115, Berlin, 1999. Springer-Verlag. Lecture Notes in Computer Science Volume 1666.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science*, pages 383–395, Portland, Oregon, 21–23 October 1985. IEEE.

[CH94] Ran Canetti and Amir Herzberg. Maintaining security in the presence of transient faults. In Yvo Desmedt, editor, *Advances in Cryptology - Crypto '94*, pages 425–438, Berlin, 1994. Springer-Verlag. Lecture Notes in Computer Science Volume 839.

[CHH00] Ran Canetti, Shai Halevi, and Amir Herzberg. Maintaining authenticated communication in the presence of break-ins. *Journal of Cryptology*, 13(1):61–106, winter 2000.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *Advances in Cryptology - EuroCrypt*

2003, pages 255–271, Berlin, 2003. Springer-Verlag. Lecture Notes in Computer Science Volume 2656.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In L. Knudsen, editor, *Advances in Cryptology - EuroCrypt 2002*, pages 337–351, Berlin, 2002. Springer-Verlag. Lecture Notes in Computer Science Volume 2045.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper B. Nielsen. Relaxing chosen-ciphertext security. In D. Boneh, editor, *Advances in Cryptology - Crypto 2003*, Berlin, 2003. Springer-Verlag. Lecture Notes in Computer Science.

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous Byzantine agreement using cryptography. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC 2000)*, pages 123–132. ACM, July 2000.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on the Theory of Computing*, pages 494–503, Montreal, Quebec, Canada, 2002.

[CP92] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology - Crypto '92*, pages 89–105, Berlin, 1992. Springer-Verlag. Lecture Notes in Computer Science Volume 740.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous Byzantine agreement with optimal resilience (extended abstract). In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 42–51, San Diego, California, 16–18 May 1993.

[CR03] Ran Canneti and Tal Rabin. Universal composition with joint state. In D. Boneh, editor, *Advances in Cryptology - Crypto 2003*, Berlin, 2003. Springer-Verlag. Lecture Notes in Computer Science.

[Cra96] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI and University of Amsterdam, 1996.

[CS01] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. Research Series RS-01-37, BRICS, Department of Computer Science, University of Aarhus, October 2001.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *Advances in Cryptology - EuroCrypt 2000*, pages 418–430, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1807.

[DCIO98] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Non-interactive and non-malleable commitment. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 141–150, Dallas, TX, USA, 24–26 May 1998.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-maleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[Des87] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Advances in Cryptology - Crypto '87*, pages 120–127, Berlin, 1987. Springer-Verlag. Lecture Notes in Computer Science Volume 293.

[Des97] Yvo Desmedt. Some recent research aspects of threshold cryptography. In M. Mambo, E. Okamoto, and E. Davida, editors, *Information Security, First International Workshop ISW '97*, volume 1196 of *Lecture Notes in Computer Science*, pages 158–173. Springer-Verlag, 1997.

[DF89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology - Crypto '89*, pages 307–315, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science Volume 435.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Y. Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002*, pages 125–142, Berlin, 2002. Springer. Lecture Notes in Computer Science Volume 2501.

[DG02] Ivan Dangaard and Jens Groth. Non-interactive and reusable non-maleable commitment schemes. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on the Theory of Computing*, Montreal, Quebec, Canada, 2002.

[DH76] Whitefield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 110–136, Berlin, 2001. Springer. Lecture Notes in Computer Science Volume 1992.

[DJN01] Ivan Damgård, Mads J. Jurik, and Jesper B. Nielsen. A generalization of Paillier's public-key system with applications to electronic voring. Accepted for publication in the special issue on Financial Cryptography, International Journal on Information Security (IJIS). Obtainable from the authors, 2001.

[DK01] Ivan Damgård and Maciej Koprowski. Practical threshold RSA signatures without a trusted dealer. In *Advances in Cryptology - EuroCrypt 2001*, pages 152–165, Berlin, 2001. Springer-Verlag. Lecture Notes in Computer Science Volume 2045.

[DLM82] Richard A. DeMillo, Nancy A. Lynch, and Michael J. Merritt. Cryptographic protocols. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 383–400, San Francisco, CA, 5–7 May 1982.

[DN00] Ivan Damgård and Jesper B. Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In Mihir Bellare, editor, *Advances in Cryptology - Crypto 2000*, pages 432–450, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1880.

[DN02a] Ivan Damgård and Jesper B. Nielsen. Expanding pseudorandom functions; or: From known-plaintext security to chosen-plaintext security. In M. Yung, editor, *Advances in Cryptology - Crypto 2002*, pages 449–464, Berlin, 2002. Springer-Verlag. Lecture Notes in Computer Science Volume 2442.

[DN02b] Ivan Damgård and Jesper B. Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In M. Yung, editor, *Advances in Cryptology - Crypto 2002*, pages 581–596, Berlin, 2002. Springer-Verlag. Lecture Notes in Computer Science Volume 2442.

[DN03] Ivan Damgård and Jesper B. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In D. Boneh, editor, *Advances in Cryptology - Crypto 2003*, Berlin, 2003. Springer-Verlag. Lecture Notes in Computer Science.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 409–418, Dallas, TX, USA, 24–26 May 1998.

[Dod03] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In Yvo Desmedt, editor, *Public Key Cryptography, 6th International Workshop on Practice and Theory in Public Key Cryptography*, pages 1–18, Berlin, 2003. Springer. Lecture Notes in Computer Science Volume 2567.

[DRS90] Danny Dolev, Ruediger Reischuk, and H. Raymond Strong. Early stopping in Byzantine agreement. *ACM Transactions on Programming Languages and Systems*, 37(4):720–741, October 1990.

[DS82] Danny Dolev and Raymond H. Strong. Polynomial algorithms for multiple processor agreement. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 401–407, San Francisco, CA, 5–7 May 1982.

[FH96] Matthew Franklin and Stuart Haber. Joint encryption and message-efficient secure computation. *Journal of Cryptology*, 9(4):217–232, Autumn 1996.

[Fis01] Marc Fischlin. *Trapdoor Commmitment Schemes and Their Applications.* PhD thesis, Fachbereich Mathematik, Johann Wolfgang Goethe-Universität, Frankfurt am Main, 2001.

[FL82] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, June 1982.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[FM97] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, August 1997.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero-knowledge protocols to prove modular polynomial relations. In Burt Kaliski, editor, *Advances in Cryptology - Crypto '97*, pages 16–30, Berlin, 1997. Springer-Verlag. Lecture Notes in Computer Science Volume 1294.

[FPS00] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *Proceedings of Financial Crypto 2000*, 2000.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology - Crypto '86*, pages 186–194, Berlin, 1986. Springer-Verlag. Lecture Notes in Computer Science Volume 263.

[FW00] Matthew Franklin and Rebecca N. Wright. Secure communication in minimal connectivity models. *Journal of Cryptology*, 13(1):9–30, winter 2000.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.

[GIL+90] Oded Goldreich, Russell Impagliazzo, Leonid Levin, Ramarathnam Venkatesan, and David Zuckerman. Security preserving amplification of hardness. In *31st Annual Symposium on Foundations of Computer Science*, volume I, pages 318–326, St. Louis, Missouri, 22–24 October 1990. IEEE.

[GJKR96a] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. On fast and provably secure message authentication based on universal hashing. In Neal Koblitz, editor, *Advances in Cryptology - Crypto '96*, pages 157–172, Berlin, 1996. Springer-Verlag. Lecture Notes in Computer Science Volume 1109.

[GJKR96b] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS. In Ueli Maurer, editor, *Advances in Cryptology - EuroCrypt '96*, pages 354–371, Berlin, 1996. Springer-Verlag. Lecture Notes in Computer Science Volume 1070.

[GJKR00] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust and efficient sharing of RSA functions. *Journal of Cryptology*, 13(2):273–300, 2000.

[GK90] Oded Goldreich and Hugo Krawczyk. On the composition of zero knowledge proof systems. In *Proceedings of ICALP 90*, Berlin, 1990. Springer-Verlag. Lecture Notes in Computer Science Volume 443.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, 15–17 May 1989.

[GL90] Shafi Goldwasser and Leonid Levin. Fair computation of general functions in presence of immoral majority. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - Crypto '90*, pages 77–93, Berlin, 1990. Springer-Verlag. Lecture Notes in Computer Science Volume 537.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[GM93] Juan A. Garay and Yoram Moses. Fully polynomial Byzantine agreement in $t + 1$ rounds. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 31–41, San Diego, California, 16–18 May 1993.

[GM95] Rosario Gennaro and Silvio Micali. Verifiable secret sharing as secure computation. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology - EuroCrypt '95*, pages 168–182, Berlin, 1995. Springer-Verlag. Lecture Notes in Computer Science Volume 921.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 291–304, Providence, Rhode Island, 6–8 May 1985.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ron Rivest. A digital signature scheme secure against chosen message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 174–187, Toronto, Ontario, Canada, 27–29 October 1986. IEEE.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of*

the Nineteenth Annual ACM Symposium on Theory of Computing, pages 218–229, New York City, 25–27 May 1987.

[Gol01a] Oded Goldreich. *The Foundations of Cryptography*, volume 1. Cambridge University Press, 2001.

[Gol01b] Oded Goldreich. *The Foundations of Cryptography*, volume 2. Cambridge University Press, 2001.

[GP90] Oded Goldreich and Erez Petrank. The best of both worlds: Guaranteeing termination in fast randomized Byzantine agreement protocols. *Information Processing Letters*, 36(1):45–49, 1990.

[Gro02] Jens Groth. personal communication, 2002.

[GRR98] Rosario Gennaro, Michael Rabin, and Tal Rabin. Simplified VSS and fast-track multi-party computations with applications to threshold cryptography. In *Proc. ACM PODC'98*, 1998.

[GT03] Shafi Goldwasser and Yael Taumann. On the (in)security of the Fiat-Shamir paradigm. In *44th Annual Symposium on Foundations of Computer Science*, pages 102–115, Cambridge, MA, 11–14 October 2003. IEEE.

[HM00] Martin Hirt and Ueli Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, winter 2000.

[HM01] Martin Hirt and Ueli Maurer. Robustness for free in unconditional multi-party computation. In J. Kilian, editor, *Advances in Cryptology - Crypto 2001*, pages 101–118, Berlin, 2001. Springer-Verlag. Lecture Notes in Computer Science Volume 2139.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In D. Boneh, editor, *Advances in Cryptology - Crypto 2003*, Berlin, 2003. Springer-Verlag. Lecture Notes in Computer Science.

[JJ00] Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000*, pages 162–177, Berlin, 2000. Springer. Lecture Notes in Computer Science Volume 1976.

[KMO89] Joe Kilian, Silvio Micali, and Rafail Ostrovsky. Minimum resource zero-knowledge proofs (extended abstract). In *30th Annual Symposium on Foundations of Computer Science*, pages 474–479, Research Triangle Park, North Carolina, 30 October–1 November 1989. IEEE.

[KT76] Donald E. Knuth and Pardo L. Trabb. Analysis of a simple factorization algorithm. *Theoretical Computer Science*, 3(3):321–348, 1976.

[KY02] Jonathan Katz and Moti Yung. Threshold cryptosystems based on factoring. In Y. Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002*, pages 192–205, Berlin, 2002. Springer. Lecture Notes in Computer Science Volume 2501.

[LLR02a] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated Byzantine agreement. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on the Theory of Computing*, pages 514–523, Montreal, Quebec, Canada, 2002.

[LLR02b] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. Sequential composition of protocols without simultaneous termination. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 203–212. ACM Press, 2002.

[LP01] Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001*, pages 331–350, Berlin, 2001. Springer. Lecture Notes in Computer Science Volume 2248.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):381–401, July 1982.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[MR91] Silvio Micali and Phillip Rogaway. Secure computation. In Joan Feigenbaum, editor, *Advances in Cryptology - Crypto '91*, pages 392–404, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science Volume 576.

[MRH04] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *The First Theory of Cryptography Conference, TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39, Cambridge, MA, USA, February 2004. Springer-Verlag.

[MRV99] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science*, New York City, NY, 17–19 October 1999. IEEE.

[MS95] Silvio Micali and Ray Sidney. A simple method for generating and sharing pseudo-random functions, with applications to clipper-like escrow systems. In Don Coppersmith, editor, *Advances in Cryptology - Crypto '95*, pages 185–196, Berlin, 1995. Springer-Verlag. Lecture Notes in Computer Science Volume 963.

[Nie02a] Jesper B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In M. Yung, editor, *Advances in Cryptology - Crypto 2002*, pages 111–126, Berlin, 2002. Springer-Verlag. Lecture Notes in Computer Science Volume 2442.

[Nie02b] Jesper B. Nielsen. A threshold pseudorandom function construction and its applications. In M. Yung, editor, *Advances in Cryptology - Crypto 2002*, pages 401–416, Berlin, 2002. Springer-Verlag. Lecture Notes in Computer Science Volume 2442.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions (extended abstract). In *38th Annual Symposium on Foundations of Computer Science*, pages 458–467, Miami Beach, FL, 19–22 October 1997. IEEE.

[OU98] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In K. Nyberg, editor, *Advances in Cryptology - EuroCrypt '98*, pages 308–318, Berlin, 1998. Springer-Verlag. Lecture Notes in Computer Science Volume 1403.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In *Proc. ACM PODC'91*, pages 51–59, 1991.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residue classes. In Jacques Stern, editor, *Advances in Cryptology - EuroCrypt '99*, pages 223–238, Berlin, 1999. Springer-Verlag. Lecture Notes in Computer Science Volume 1592.

[Pas03] Rafael Pass. On deniability in the common reference string and random oracle model. In D. Boneh, editor, *Advances in Cryptology - Crypto 2003*, Berlin, 2003. Springer-Verlag. Lecture Notes in Computer Science.

[Rab83] Michael O. Rabin. Randomized Byzantine generals. In *23th Annual Symposium on Foundations of Computer Science*, pages 403–409, Los Alamitos, CA, 1983. IEEE.

[Rab98] Tal Rabin. A simplified approach to threshold and proactive RSA. In Hugo Krawczyk, editor, *Advances in Cryptology - Crypto '98*, pages 89–104, Berlin, 1998. Springer-Verlag. Lecture Notes in Computer Science Volume 1462.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 73–85, Seattle, Washington, 15–17 May 1989.

[RSA78] Ronald Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology - EuroCrypt 2000*, pages 207–220, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1807.

[Sho01] Victor Shoup. OAEP reconsidered. In J. Kilian, editor, *Advances in Cryptology - Crypto 2001*, pages 239–259, Berlin, 2001. Springer-Verlag. Lecture Notes in Computer Science Volume 2139.

[SYY99] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC^1. In *40th Annual Symposium on Foundations of Computer Science*, New York City, NY, 17–19 October 1999. IEEE.

[TC84] Russell Turpin and Brian A. Coan. Extending binary Byzantine agreement to multivalued Byzantine agreement. *Information Processing Letters*, 18(2):73–76, February 1984.

[Tou84] Sam Toueg. Randomized Byzantine agreements. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 163–178, 1984.

[Yao82a] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, 3–5 November 1982. IEEE.

[Yao82b] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 November 1982. IEEE.

Index

Recent BRICS Dissertation Series Publications

DS-03-8 **Jesper Buus Nielsen.** ***On Protocol Security in the Cryptographic Model*****. August 2003. PhD thesis. xiv+341 pp.**

DS-03-7 **Mario José Cáccamo.** ***A Formal Calculus for Categories*****. June 2003. PhD thesis. xiv+151.**

DS-03-6 **Rasmus K. Ursem.** ***Models for Evolutionary Algorithms and Their Applications in System Identification and Control Optimization*****. June 2003. PhD thesis. xiv+183 pp.**

DS-03-5 **Giuseppe Milicia.** ***Applying Formal Methods to Programming Language Design and Implementation*****. June 2003. PhD thesis. xvi+211.**

DS-03-4 **Federico Crazzolara.** ***Language, Semantics, and Methods for Security Protocols*****. May 2003. PhD thesis. xii+159.**

DS-03-3 **Jiří Srba.** ***Decidability and Complexity Issues for Infinite-State Processes*****. 2003. PhD thesis. xii+172 pp.**

DS-03-2 **Frank D. Valencia.** ***Temporal Concurrent Constraint Programming*****. February 2003. PhD thesis. xvii+174.**

DS-03-1 **Claus Brabrand.** ***Domain Specific Languages for Interactive Web Services*****. January 2003. PhD thesis. xiv+214 pp.**

DS-02-5 **Rasmus Pagh.** ***Hashing, Randomness and Dictionaries*****. October 2002. PhD thesis. x+167 pp.**

DS-02-4 **Anders Møller.** ***Program Verification with Monadic Second-Order Logic & Languages for Web Service Development*****. September 2002. PhD thesis. xvi+337 pp.**

DS-02-3 **Riko Jacob.** ***Dynamic Planar Convex hull*****. May 2002. PhD thesis. xiv+110 pp.**

DS-02-2 **Stefan Dantchev.** ***On Resolution Complexity of Matching Principles*****. May 2002. PhD thesis. xii+70 pp.**

DS-02-1 **M. Oliver Möller.** ***Structure and Hierarchy in Real-Time Systems*****. April 2002. PhD thesis. xvi+228 pp.**

DS-01-10 **Mikkel T. Jensen.** ***Robust and Flexible Scheduling with Evolutionary Computation*****. November 2001. PhD thesis. xii+299 pp.**

DS-01-9 **Flemming Friche Rodler.** ***Compression with Fast Random Access*****. November 2001. PhD thesis. xiv+124 pp.**