New Notions of Security

Manoj Prabhakaran

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE

September 2005

Abstract

Secure multi-party computation (MPC, for short) is a powerful cryptographic concept which lets mutually distrusting parties collaborate without compromising their private information (beyond what is required by the functionality for which they collaborate). The functionality allowed in such a collaboration is so general that MPC subsumes virtually all other cryptographic tasks. Much of the two and a half decades of cryptographic research can be seen as striving towards the Holy Grail of realizing secure MPC in the most challenging scenario in which the parties carry out multiple tasks concurrently, the entire communication is adversarially controlled and there are no universally trusted entities.

In this thesis, for the first time we show how to realize secure MPC in such a general setting. Our contribution can be considered three-fold:

- **Definition of Security.** We present a new framework — called *Los Angeles Network-aware security* — for defining security of MPC protocols. This builds on Canetti's Universally Composable security framework, under which it was known that very few distributed tasks can be carried out securely (unless globally trusted entities were used by the protocol).

- **Protocols and Proofs.** We build a new protocol for multi-party computation which uses no globally trusted entities, and prove that it is secure in the Los Angeles Network-aware security framework, under certain complexity theoretic assumptions. The high-level structure of the protocol and proof of security resembles that in previous works on multi-party computation, but we employ novel approaches in designing the lower level elements of our protocol.

- **Complexity Theoretic Assumptions.** We introduce new complexity theoretic assumptions, and show their use in proving the security of our protocols. The assumptions we make are somewhat different from those used in previous works. We informally argue why our assumptions are reasonable.

Also, we introduce an extension to the Los Angeles Network-aware security framework — called monitored security — to obtain a greater security-efficiency trade off.

We show how Network-aware security guarantees, albeit weak, can be given under this framework for much more efficient and simpler protocols.

Acknowledgments

First of all, I would like to acknowledge my advisor Amit Sahai for all the help and guidance during my five years in graduate school. Amit was a co-author in most of the work I did during this time. But beyond that he has been a wonderful friend and a kind and generous advisor. Amit put more faith in me than I would myself, made sure I had an uninterrupted supply of exciting problems to work on and also let me work on my pet problems which led to the results in this thesis.

I thank my PhD committee — Sanjeev Arora, Boaz Barak, Moses Charikar, Ed Felten and Amit Sahai — for all the encouragement and feedback. The presentation of this thesis has benefited from suggestions by Sanjeev.

I would like to thank everyone at the computer science department at Princeton: not only for the great education, but also for the friendly environment. I will forever be indebted to the remarkable faculty as well as the wonderful fellow students in the theory group for helping me make the transition from an aspiring researcher to a researcher. I fondly recall the very first publication I worked on: a paper co-authored with my office-mate, neighbor and friend Ding Liu, prompted by Andy Yao, the ever encouraging teacher that he is.

Perhaps I attended more lectures at the neighboring Institute of Advanced Study than at the University. My thanks to Avi Wigderson and others at the Institute for organizing those lectures, as well as to all the visitors who came there to present their work.

I have been fortunate to work with an increasing number of researchers, and I am grateful to them for their patience, for sharing their knowledge and experience with me and for making research fun. Right at the top of this list are the fabulous researchers at the cryptography group at IBM T J Watson Research Center — Masayuki Abe, Ran Canetti, Rosario Gennaro, Shai Halevi, Yael Kalai, Hugo Krawczyk, Yehuda Lindell and Tal Rabin — with whom I spent a fruitful summer. My collaborators in cryptography include Yevgeniy Dodis, Juan Garay, Shai Halevi, Yuval Ishai, Yael Kalai, Yehuda Lindell, Ben

Lynn, Phil McKenzie, Tatsuaki Okamoto, Shien Jin Ong, Alon Rosen, Amit Sahai, David Wagner and Ke Yang. Thanks to Rafi Ostrovsky and others at the Computer Science department at UCLA for making my brief stay there a good experience. This list remains incomplete as I have not listed all those who shared their insights with me at various occasions (many of whom are acknowledged in my publications).

Beyond the department and the research, Princeton — my first international experience — has been a remarkable phase in my life. It also turned out to be my first opportunity to explore the possibility of making contributions (however small they may be) beyond my professional career, as I discovered the spirit of volunteerism. I wish to thank all my room-mates, other friends and those I interacted with in various organizations for making Princeton an unforgettable experience. Also thanks to my friends and everyone at the UC Housing Association in Los Angeles for a thoroughly enjoyable, if not educative, "co-operative" experience.

Thanks are also due to the faculty and friends at the Indian Institute of Technology, Bombay, where I decided I wanted to be a theoretical computer scientist. Thanks also to everyone back in my hometown — all my teachers in school, old friends and well-wishers — for a (mostly!) pleasant childhood.

And last, but by no means the least, thanks to my parents, my brother and the extended family back home, for everything. It is to them that I dedicate this thesis.

My graduate studies were partly supported by an IBM PhD Fellowship in 2004-05, grants from the NSF ITR and Cybertrust programs and Prof. Sahai's Alfred P. Sloan Foundation Fellowship.

To *acchan*, *amma* and *ceṭṭan*.

Contents

Preface

The contents of this thesis are primarily derived from the two papers [PS04] and [PS05]. In addition, it includes many unpublished extensions and a new presentation of the model. We briefly point out some of the highlights of this thesis, which the readers who are familiar with the prior works in this area (in particular [Can01, Can05]) may find useful.

- **A Fresh Presentation of the Model.** We provide a fresh presentation of the model for computation and communication, with the aim of making it fit the real-life scenario easier. In particular, we use high-level descriptions of programs (using their input-output behavior) eschewing the formulation of a Network as a system of Interactive Turing Machines; we avoid explicit sequentialization of concurrent programs, instead using an abstraction which facilitates working directly with concurrent executions; we provide an alternate way to impose polynomial time restrictions on computations; we use a more general notion of program scheduling, to take into account the possibility that scheduling of programs may provide *side-channels* leaking secret information. This is especially useful in extending the model to allow the notion of clock-time and timing-attacks.

- **Extended Universal Composition Theorem.** We present an extended version of the Universal Composition theorem, which addresses the following enhancements to the regular Universal Composition theorem: it allows nested subroutines, joint-state subroutines, security given with respect to different angels and multiple simultaneous protocol sessions invoked by the "environment." While some of these enhancements have appeared in earlier literature, it is for the first time that they are all presented together to obtain the big picture of Universal Composition.

- **Proofs Regarding the use of "Locked States" with Monitors.** Chapter 6 consists mostly of results from [PS05]. However it also contains a detailed exposition to the notion of "locked states" which was only outlined in [PS05].

- **Detailed Proofs.** The proofs of security in Chapters 5 and 6 are given in substantially more detail and rigor compared to previous presentations of these results. Since the proofs typically involve many constructions interspersed with reasoning about their properties, for ease of following, we have marked out the larger constructions in the text.

- **New Devices.** For clarity we have introduced some new devices into the description. Firstly, we use a shorthand ($\mathtt{comb}(\cdots)$) for expressing combination of multiple elements in a Network. This replaces the typical convention used in previous works of explaining the mechanics of constructions by enumerating how various

messages are handled. Our new convention prevents the essence of the constructions from being obfuscated. Secondly, for clarity we introduce a new device called a transvisor for describing simulators (Section 3.3.4).

- **New Terminology.** The terms *Network-aware security* and *Los Angeles Network-aware security* are new. In [PS04, PS05] they were termed *Environmental Security* and *Generalized Environmental Security,* respectively.

- **Notes for the nuanced reader.** Several comments and elaborations are included as endnotes in each chapter. Many of these are meant for the expert reader, and may be skipped by the first time reader.

As is inevitable in the first published version of any work of this length and technical detail, we expect that there may be some minor errors and inaccuracies. Our precaution against introducing any serious flaws has been the level of detail at which arguments are presented. However, theoretical cryptography lacks a comprehensive language to formally capture all arguments that any non-trivial work like this requires. Thus the proofs employ a mix of plain English and some abuse of notation along side rigorous mathematical derivations. By generously introducing mathematical notation we have tried to avoid the pitfalls of intuitive reasoning.

We hope that this exposition to the state of the art in what is a fast evolving frontier of cryptography will be useful for new students as well as the experts in the field.

Chapter 1

Introduction

With the revolutions in Information and Communication Technology, and the availability of enormous computational resources, the nature of "information" changed for ever. Among other things, it has reshaped the concept of information security. This thesis fits into – and indeed, contributes to defining – these evolving notions of security.

There are two aspects in the modern understanding of information security which which gets highlighted in this thesis.

- **Nature of tasks for which security is required.** Traditionally information security was confined to security of information transmission (encryption and authentication). However, increasingly it has come to include security of information that is the input for, or the output from, a variety of distributed computational tasks. Example scenarios would include electronic commerce, online voting, private information retrieval or privacy preserving datamining. All such computational tasks are collectively referred to as *multi-party computation* (MPC).

- **Nature of the environment in which we must operate.** Internet has defined the standard computational and communication environment where the distributed tasks are carried out. Each agent typically takes part in multiple tasks simultaneously, and shares information among its different tasks. There is little or no trust among the different agents; the communication channels are unreliable, and open to sniffing and spoofing by adversarial agents. We refer to this environment involving all the agents, their computations and communications, as well as all the adversarial agents, collectively as the *Network*. Any guarantee of information security has to be cognizant that the activities take place in such a Network.

Recent advances in cryptography take into account both of the above aspects of information security. The most advanced notion of security developed in theoretical cryptography, which in this thesis we call *Network-aware security*, is specifically designed to address these issues, something which the earlier notions of security did not address.

The central contribution of this thesis is in showing that protocols for multi-party computation can be designed which are secure in the sense of Network-aware security. This answers perhaps the most pressing question at the center of more than two decades of modern cryptography which had set secure multi-party computation as its primary goal. Our results build on top of the rich collection of tools, techniques and concepts that this thread of research has forged. And in doing so, we improve on previous attempts which either did not address the issue of Network-awareness, used trusted setups or imposed non-standard restrictions.

1.1 Three Elements of Modern Cryptography

Before we proceed further, it will be useful to present an overall outlook on the central elements in this thesis. Instead of considering this as a monolithic result for securely realizing multi-party computation, we consider three important aspects of cryptography and how this thesis makes important contributions to all of them.

Modern cryptography can be viewed as built on the two pillars of *security definitions* and *complexity theoretic primitives*. Resting on these pillars (and indeed bridging them, in a technical sense) is what forms the expansive edifice of cryptography: the *cryptographic protocols and their proofs of security*. On our way to secure MPC we shall make contributions to all these three segments.

1.1.1 Definition of Security

The first hurdle in the challenging route to secure multi-party computation is *defining security*. The single most important contribution of this thesis would be the new definitional framework that we develop. For the first time, we believe we have achieved the delicate balance between a rigorous mathematical definition and a pragmatic definition which allows us to construct protocols (without trusted setups or non-standard restrictions) for secure MPC.

The first satisfactory definition of Network-aware security was introduced by Canetti in 2001 [Can01, Can05] under the name of UC Security (for Universally Composable Security), and independently by Pfitzmann and Waidner [PW01]. However it was shown that very few distributed tasks can be securely carried out under these definitions (except with trusted setups or non-standard restrictions). In this work we provide an alternate notion, called *Los Angeles Network-aware security.* (We shall explain the colorful name shortly.) Our definition builds on the original Network-aware security definition. Like them, our definition also satisfactorily captures security notions for general multi-party computation in the context of a Network. Further, it also satisfies the so called "Universal Composition" property: it refers to the guarantee that secure protocols remain *collectively secure* when deployed together; it also allows one to build complex protocols in a modular way by *composing* simpler protocols.

The name *Los Angeles security* derives from the new conceptual tool we add to the definition, namely an *angel*.[1] The angel is a critical tool which enables us to *define and prove* the security of the protocols in this thesis. However, the protocols themselves make no direct use of the angel.

Security using Monitors. We also introduce an interesting variant to Los Angeles security called *monitorable security*. It uses yet another new concept, called *monitors*. The typical the use of this definition would be to provide *weaker* security guarantees compared to the normal Los Angeles Network-aware security definition. The advantage of this definition is that it can be used to provide Network-aware security guarantees (albeit weak) to even some extremely simple (and relatively efficient) protocols.

1.1.2 Complexity Theoretic Primitives

Modern cryptography crucially relies on Computational Complexity Theory to prove the security of protocols. We introduce new complexity theoretic primitives, and show their use in proving the security of our protocols. It is well known that even for the simplest tasks, proving security of any protocol, under any reasonable definition of security, requires complexity theoretic results, which are way beyond what is currently provable. Hence the proofs of security always make a few well-stated *basic assumptions* about hardness of certain computational problems. We too make such assumptions regarding our primitives.

Our assumptions are somewhat different from those used in previous works. This is a drawback, because the new assumptions are not as well studied as the more conventional ones. On the other hand, using new assumptions comes with its own advantages, both for the theoretical studies and practical implementations. The practical advantage is that new assumptions can often provide a trade-off between security and efficiency. Today, most of the protocols from theoretical cryptography are unimplemented, thanks to their high complexity: computational complexity, communication complexity and conceptual complexity. The protocols in this work are still too complex and inefficient to be considered acceptable by the practitioner today. However our protocols are significantly more efficient and simpler than previous comparable protocols which sought to provide Network-aware security. This simplicity is facilitated in part by our complexity theoretic assumptions.

As we shall see, our definitional framework is very general and holds promise for introducing even further trade-offs between complexity assumptions and efficiency. This might make it easier for theoretical cryptography to make its way more and more into practice.

A more fundamental benefit to theory, however, is in the form of the challenge to understand the new assumptions: how they relate to other complexity theoretic assumptions, and whether there is anything fundamentally different in the nature of these as-

sumptions that separates it from previous ones. The application to cryptography strengthens this pursuit by providing fundamental connections and directions. It also makes it more than of academic interest.

Like in other works in foundational cryptography, our assumption is a "general assumption," as opposed to assumptions about any specific number-theoretic or algebraic problem. This allows the possibility that even if a specific assumption required in an implementation of our protocol turns out to be incorrect, an alternate implementation based on another assumption may be available. We shall informally argue why our new assumptions are reasonable, and point to recent progress by other researchers in instantiating our general assumptions using number theoretic primitives.

1.1.3 Protocols and Proofs

The foundational contributions apart, the central concrete result of this thesis, is a new protocol for multi-party computation, which is provably secure under our definition of Network-aware security, under the new complexity theoretic assumptions. The high-level structure of the protocol and proof of security resembles that in previous works on multi-party computation. Especially, it draws from the work of Canetti, Lindell, Ostrovsky and Sahai [CLOS02] (which provided secure MPC under Canetti's definition of Network-aware security, but depended on a non-standard setup called common random string), which in turn was based on the work of Goldreich, Micali and Wigderson [GMW87] (which provided secure MPC under a "stand-alone" security definition). However, we will need to employ some novel approaches in designing the lower level elements of our protocol.

This protocol could be considered as an illustration of the viability of our new security definition. When seeking improved efficiency, the protocol, and the complexity theoretic assumptions used to prove its security, may be changed.

The protocols in the monitorable security framework (Chapter 6) are fairly simple and resemble the conventional protocols. However the proofs are quite non-trivial, especially when the security guarantee is enhanced by using "locked states" (Section 6.5.2).

1.2 A Brief History

Over the last two decades, there has been tremendous success in placing cryptography on a sound theoretical foundation, and building an amazingly successful theory out of it. The single most important driving force behind the development of this theory was the goal of achieving secure multi-party computation. This thread of research led to robust security definitions, tools and techniques for building and analyzing cryptographic schemes, and numerous results in trying to understand the connections between various complexity theoretic assumptions used through out cryptography.

1.2.1 Security Definitions

Original notions of cryptographic security were centered around encryption. Mathematical notions of security of encryption was pioneered by Shannon [Sha49], who defined security in information theoretic terms. Decades later, the utility of encryption was revolutionized by the concept of public-key encryption [DH76], initiating "modern cryptography." However adequate definitions were introduced only much later, in the seminal work of Goldwasser and Micali [GM84]. After Shannon's remarkable feat of mathematically formulating the concept of information, this marked the next major advance by capturing the notion of "knowledge" (in a manner relevant to cryptography). This work laid down the basic pattern for future definitions of security in modern cryptography going beyond encryption, and has been epitomized in the definition of the "zero-knowledge" property of protocols [GMR85].

These definitions in the earlier part of 1980's formed the basis for defining security of general multi-party computations. Yao [Yao82a] introduced the concept of security for multi-party computation in intuitive terms through the famous "millionaire's problem." But, even after protocols for general multi-party computation were developed [Yao86, GMW87], it was over a decade that a mature definition of security for multi-party computation slowly evolved [GL90, MR91, Bea91a, Bea91b, Can95, Can00].

The evolution of definitions not only addressed more applications (going from encryption to general multi-party computation), but also started addressing applicability to more and more realistic situations. Initial definitions were framed in a "stand-alone" setting and provided no security guarantees for "composition," when instances of one protocol were to be used as subroutines within a larger protocol. Neither were these definitions applicable to a Network setting. The composition issues were identified first with respect to basic primitives like commitments and zero-knowledge proofs [GK96, DDN00, DNS98]. These works also presented security definitions which addressed the specific composability issues that they examined. In the larger context of multi-party computation, security definitions addressing composability were first developed with several restrictions [Can00, DM00, PSW00, PW00], before general frameworks for Network-aware security, appeared [Can01, PW01, BPW04]. These last works present definitions which are essentially equivalent, and constitute the notion of Network-aware security, as we intend in this work. We refer the reader to [Can05] for a brief survey on this thread of development.

UC Security. As mentioned above [Can01] (with refinements in [Can05]) introduced one of the first satisfactory definitions of Network-aware security, under the name UC security (which stands for Universally Composable security). We shall briefly mention the salient features of this framework, as they are relevant to our definitions.

The basic underlying notion of Network-aware security in the UC model and its predecessors is based on *simulation*. An "ideal" world is described, where all requisite tasks

get accomplished securely, as if by magic. The goal of the protocol designer is to find a way to accomplish these tasks in the "real" world (where magic is hard to come by) so that no malicious adversary can take advantage of this substitution of ideal magic by real protocols. To formalize this, we say that for every malicious adversary $\mathcal{A}$ that tries to take advantage of the real world, there is an adversary $\mathcal{S}$ that can achieve *essentially the same results* in the ideal world. The "results" are reflected in the behavior of an *environment*. In Chapter 3 we further elaborate on the definition and its implications.

Universal Composition. The advantage of UC secure protocols, as shown in [Can01, Can05], is that they are "Universally Composable." Roughly this means the following: one can build a protocol using idealized components and prove its security; then, if the idealized components are replaced by protocols which *securely realize* these components (according to the UC security definition), then the security of the larger protocol continues to hold. One key aspect here is that multiple instances of the smaller components may be concurrently invoked by the larger protocol. Security should continue to hold even in this case. Canetti [Can01, Can05] shows that this is indeed the case if the UC security definition is employed. Interestingly, in [Can01, Can05] the security definition itself is named after this key property, though the UC property does not immediately follow from the definition.

Impossibility Results. Unfortunately, UC security as introduced in [Can01, Can05] turns out to be too strong to be achievable in standard settings. It has been shown that much of the interesting cryptographic tasks (including *e.g.* commitment, non-trivial zero knowledge proofs and secure multi-party computation) *cannot* be secure by this definition when the adversary can control at least half the parties in a protocol [Can01, CF01, CKL03, Lin03b]. On the other hand, under a trusted setup assumption (of questionable applicability in many situations) — that there is a public reference string chosen by a completely trusted party — it is known how to build protocols for secure multi-party computation (even with dishonest majority) satisfying the UC security definition. Also it was known how to achieve this when the majority of the parties are honest [Can01, Can05, BGW88, CCD88].

New Notions of Security. Prior to the results described here, even for simple multi-party computation tasks cryptography could offer only an unsatisfactory choice of compromises: either choose to use protocols which have *no provable guarantee in a Network setting* or use trusted setups like common reference string to implement provably secure protocols.

In [PS04], we introduced the notion of Los Angeles Network-aware security[2] to which the impossibility results above no more applied. Further, this enabled us for the first time to give protocols for general multi-party computation, which are provable secure (against

"static adversaries") under a satisfactory definition of Network-aware security. This definition is presented in Chapters 2 and 3. (The protocols and proofs of security are given in Chapter 5). In [PS05] we modified the security definition of [PS04] to capture *weaker* notions of Network-aware security, in an attempt to allow greater security-efficiency tradeoffs. This definition, called "monitored security" is presented in Chapter 6.

1.2.2 Constructions and Techniques

It is remarkable that the promise of theoretical investigation — that an in-depth study of specific and simplified problems in idealized settings can eventually lead to sophisticated solutions for the more general and complex problem in realistic settings — is indeed borne out by modern cryptography. Starting with fundamental techniques developed very early on [Yao82b, GM84], a wealth of constructions and methods of analysis that were devised for very specific problems, directly feeds into the development of protocols for secure multi-party computation. While it is beyond the scope of our work to enumerate all such contributions, a few important tools that need to be mentioned are *oblivious transfer schemes* [Rab81, EGL85, GMW87], *commitment schemes*, *secure coin-flipping* [Blu82], *zero-knowledge proofs of membership* (for languages in NP) [GMR85, GMW91], and the first *two-party and multi-party computation protocols* [Yao86, GMW87]. A subsequent line of work on multi-party computation, initiated by [BGW88, CCD88, RBO89], restricted itself to the case of "honest majority" (wherein a majority, or sometimes two-third, of the participants are guaranteed to be honest), in order to avoid complexity theoretic assumptions, using verifiable secret sharing [CGMA85] as a central tool.

All the above mentioned works were developed in the simplified "stand-alone" setting. Network-aware secure protocols for multi-party computation were known only with the trusted setup of common reference string (CRS) [CF01, CLOS02], or for the "honest majority" restriction [BGW88, CCD88, RBO89, GRR98, Can01]. We point out that the ones based on the CRS setup in turn followed the protocol framework developed in [GMW87]. Protocols for multi-party computation in [PS04, PS05] (presented in Chapters 5 and 6) also heavily rely on this framework. One of our protocols in Chapter 5 also derives from construction techniques used in [BL02].

Recently, in an attempt to achieve highly composable multi-party computation using the conventional notion of PPT simulation (i.e., not using angels), and without setup assumptions, we gave a protocol in which the parties in the Network have access to local clocks [KLP05] (with reasonable assumptions on bounds in clock-drifts). The complexity theoretic assumptions there are more standard than the ones here. However, though formulated in a Network environment, the security guarantees there fall short of unrestricted Network-aware security, and the composition property is weaker as well.[3] Indeed there we argue that some such weakening is inevitable, even when making use of the clocks of the parties. More recently Barak and Sahai [BS05] gave a secure multi-party computation protocol in the Los Angeles Network-aware security framework (see

Endnote 7 of Chapter 5), which while much more complicated, uses a different set of complexity assumptions which are arguably more standard. Their protocol also falls into the framework initiated by [GMW87].

Weaker composability of protocols were investigated mainly by considering restricted Networks with limitations on how the multiple sessions of a protocol could be executed. This includes the research on concurrent zero-knowledge [DNS98, RK99, KPR98, KP01, CKPR01, PRS02], (with a heavy focus on simulation by "rewinding" making it probably less relevant in the context of Network-aware security), and on *bounded concurrent* zero-knowledge proofs and multi-party computation [Bar01, Lin03a, PR03, Pas04, PR05] (techniques from which have found use in [BS05]). In contrast, the results presented in Chapter 6 offer strong Network-awareness and composability properties, but offer weaker security guarantees to begin with. The results are limited in applicability and do not by themselves yield sufficiently secure protocols for general multi-party computation tasks. Instead this must be looked up on as a promising direction for carrying out the trade-off between efficiency and security. These results appeared in [PS05] but the proofs of Section 6.5.2 are mostly new.

We remark that security definitions based on super-PPT simulators have been considered before, in the context of zero-knowledge proofs, but in much more restricted settings [Oka91, Pas03]. Also, alternate weaker definitions were explored within the Network-aware security framework in the context of simpler tasks: the use of "non-information oracles" in [CK02] is somewhat similar to the use of semi-functionalities in this work.

1.3 Overcoming the Impossibility

An interesting aspect of the results presented here is that they achieve something which was widely considered to have been ruled out by the strong impossibility results proven earlier. In fact, virtually all research on Network-aware security was being focused on using the trusted setup of common reference string. We briefly point out what is involved in avoiding the impossibility results. A more illuminating discussion is deferred to Section 5.10.2.

Our starting point for overcoming the impossibility of secure MPC in the plain-model (i.e., without trusted setups) is the observation that in the UC model, even if the adversary has unlimited computational powers, the "ideal world" model — where the functionality is achieved through idealized components — still captures the notion of a secure process in practically all cases of interest. (See Chapter 3 for a detailed discussion.) Accordingly, we generalize the UC model, by providing the ideal adversary (or "simulator") with super-PPT computational resources.

However, if composability needs to be retained, we should provide the environment also with similar computational powers, which will lead us back to the strong impossibility results of [Can01, CF01, CKL03, Lin03b]. If composability were to be abandoned,

then it is not clear how to build and prove security of protocols for complex tasks, nor is it clear if multiple secure protocols remain *simultaneously secure* in any sense when deployed together. Thus, on the face of it, we still cannot attain (in the plain model) a useful Network-aware security notion.

We break out of this conundrum by allowing the adversary *and* the environment *carefully regulated access* to super-PPT computational resources. This regulated access is provided by the (imaginary) entity that we call the *angel*. The angel has super-PPT computational resources, *and* it bases its behavior on the state of the Network. [4] Roughly, the way we use the angel is as follows: it acts as an oracle providing collisions in a collision resistant hash function used in the protocol. However, it provides *only such collisions as cannot be used to break the security of the honest parties* (but can be used by the simulator to "cheat" an internal copy of the adversary). In Section 3.2 we define what an angel is. In Section 5.3.2 we present the specific angel that we use. In Section 5.10.2 we shall describe in further detail how the introduction of the angel helped us avoid the impossibility results.

1.4 On Trusted Setups

Our primary motivation can be seen as removing the need to have globally trusted elements in the Network for realizing secure MPC (as was needed before our work). In fact, the entire task of designing secure MPC protocols can be regarded as removing the need for global trust: if a globally trusted *party* were available in the Network, secure MPC can be almost trivially realized, by using this party to act as the ideal functionality.[5] Using secure MPC protocols in the common reference string model, this trust requirement is significantly reduced: the trusted entity has to only pick a reference string and distribute it to all parties at the beginning of the computation. It need not be available as the computation proceeds, and indeed if the state of this trusted entity is destroyed then it need not remain trusted after that. Nevertheless, it is crucial for the security that initially when the common reference string is being chosen, it is done honestly. Otherwise the adversary could build a trapdoor into this string and in all schemes which make use of the CRS this leads to a *complete* break-down of security.

The ultimate goal of reducing the trust setup is to have each party trust nothing else but its local execution.[6] The contribution of this thesis is to reach this goal, without sacrificing the generality of the security definition, or of the model of the Network and adversary.

1.5 Organization of the Thesis

In Chapter 2 we describe the model of computation and communication under which our results are stated. Chapter 3 gives the main security definition we develop, and Chapter 4 states and proves the Universal Composition theorem and its extension. The

construction and proof of security of our protocol for secure MPC, along with complexity assumptions and the specification of the angel used in the security statement, appear in Chapter 5. Chapter 6 presents the framework of monitored security, along with simple protocols for limited tasks in that model. We conclude with possible directions of further investigation, in Chapter 6. For ease of reference, an index of notation is provided at the end. Endnotes appear at the end of each chapter.

Notes

[1]We assure the reader that it is only incidental that this thesis was written in the city of Los Angeles.

[2]In [PS04], Network-aware security is called "environmental security." [Gol04] also uses this term. Los Angeles Network-aware security is termed "generalized environmental security" in [PS04]. The term "generalized" refers to the fact that using different angels, one gets different (incomparable) notions of security, and in particular, using a "null-angel" gives the security definition from [Can01].

[3]Using the vocabulary of Network-aware security, protocols in [KLP05] are secure only when the adversary reads all communication from the environment *with a delay*. (This ensures that the messages from the environment are available to a transvisor a little in advance, giving it a limited ability to "rewind.") The composition property is also slightly weaker: it requires modifying a protocol while substituting one of its subroutines with a subroutine as secure as the first. The modification involves delaying the messages sent by the protocol.

[4]If Γ behaves as an oracle to a fixed function (which does not depend on the state of the Network at all), results of [Can01, CF01, CKL03, Lin03b] will still imply impossibility of securely realizing the functionalities even with respect to classes of environment, adversary and simulator which have access to Γ.

[5]Even if a trusted party is used to directly implement the ideal functionality, one still requires that messages be transmitted securely to the trusted party. Network-aware secure encryption schemes (see Section 5.8) can be used for this.

[6]The local execution that can be trusted includes all the programs started by the party. Also the local "operating system" can be trusted to protect the internals of the programs from direct access by other entities (this will be required by our model of the Network). If these trust assumptions are violated, we consider the party to be corrupted.

Chapter 2

Modeling the Network

2.1 Introduction

The security guarantees are only as good as the models in which they are proven. Though it is impossible to precisely model all scenarios of communication and computation, and take into account all possible avenues of attack (the so-called *side-channel* attacks), it is important to have a general model which addresses all major concerns in a *typical* scenario. In this chapter we detail the models of computation, communication and adversarial behavior underlying the results of the thesis. We also point out a few possible variations, where relevant.

2.2 The Concrete Model

Our attempt is to use an abstract model that can accommodate a typical internet-like scenario, while not being tied too closely to any particular technology in use today. We start by highlighting the features of a concrete model that we shall accommodate in our abstraction.

2.2.1 The Network

In the typical internet setting, the overall system consists of multiple computers (or other computing devices), connected through communication channels. The communication medium itself is effected through many computing devices (routers, switches etc.), but as we shall see, we will be able to ignore the details of this. Each computer has multiple programs running in it, which communicate with other programs in the same computer or across the network. All the computers and all the programs running in them, the communication channels, the user inputs and outputs are all together termed as the Network.

2.2.2 The Adversarial Elements

The security guarantees will be available only to parties which follow the prescribed protocols correctly, and whose internal states are not accessible to other parties. Parties which deviate from the protocol, or whose internals become accessible to others (any more than allowed by the protocol) are considered *corrupt*. Corrupt parties may collude with each other and share their information. The messages sent out on the communication channels are prone to eavesdropping (by the corrupt parties). Messages may be altered (by corrupt routers, for instance), delivered out of order, or never delivered. Further *a priori* there is no mechanism to identify the point of origin of a message that is delivered.

2.2.3 The Tasks

The tasks that we want to securely realize are beyond what is typically found in an internet setting today.[1] Traditionally information security was confined to security of information transmission (encryption and authentication). However, in theoretical cryptography it is standard to consider security of information that is the input for, or the output from, a variety of distributed computational tasks. Applications of such tasks would include electronic commerce, online voting, private information retrieval or privacy preserving datamining. All such computational tasks are collectively referred to as *multi-party computation* (MPC). We shall see below how to define these tasks and their security requirements.

2.3 The Abstract Model

Now we present the abstraction used to model the situation described in the previous section. As it will be clear, this abstraction is minimal and independent of much of the specifics of the concrete model.

Our model. while building on the models in [Can01, Can05], differs from them significantly in details. The main points of difference are

- we avoid the use of Interactive Turing Machines (ITMs) or any specific model of computation to describe the Network,[2]
- we avoid explicit sequentialization of the parallel computations in the system,
- we allow arbitrary scheduling of the computations by the environment (instead of restricting it to be computable by a "controller function" which does not have access to the internals of the environment), and
- we differ in the way polynomial time bounds are imposed on the Network.

The current presentation is more general and, we believe, simpler.

2.3.1 The Network: System of Networked Computers

Recall that by the Network, we refer to the entire system of computation and communication.

The computation itself can be modeled in standard ways: say as a Turing Machine, or at a higher level as just a computable function which is specified by its observable behavior (without specifying the implementation; however, the computational complexity of this function will be relevant to us).

The behavior of the communication medium is non-trivial and may involve computational effects: message delivery is unreliable, there may be long delays, and the order and even the contents of the delivered messages may be adversarially controlled. To allow for such generality, we model the communication channels by a computing entity, with which all parties can directly communicate. This entity receives messages to be delivered, and delivers (possibly entirely different) messages to the other parties. We expect the system to function only when there are some minimum guarantees on message delivery from the this channel. But, *the security properties must be preserved no matter how this channel behaves*.

Now we proceed to describe the system more formally. First we shall present it with no reference to the complexity of the computations involved, and later specify the complexity restrictions. The model consists of the following entities, which either closely model easily recognizable real-life entities, or are "virtual entities" which either abstract out the behavior of various components in a real-life setting (as is the case with the adversary), or are useful in conceptualizing the security obtained by a protocol (as with ideal functionalities and the angel).

- The *parties* (loosely) correspond to the various computers in the Network. They participate in protocol sessions, by running *program instances*.
- The adversary, a virtual entity which controls all the corrupted programs as well as the communication medium, will be denoted by $\mathcal{A}$.
- Other virtual entities – functionalities and the angel – used to define security will be described later.
- The rest of the Network is abstracted out as the environment $\mathcal{Z}$.

Protocols, Programs, Parties.

A *protocol* is simply a program specification (instructions or code) to be followed by a set of participants to achieve some functionality. Formally we define a protocol as a computable function which operates on its inputs (incoming messages) and internal state, and produces outputs (outgoing messages) and a new modified state. We shall denote

protocols by symbols like π, ρ, σ, ϕ etc. (Later we shall have named protocols like COM, ZK etc.)

An instantiation of a protocol is termed a *session*. A particular session of a protocol will have an associated *session identifier* (session ID or SID, for short). A session of a protocol π with session ID sid will be denoted by $\Sigma_{\pi,sid}$ (possibly omitting either or both of the symbols π and sid, when not required). We shall insist that if an environment starts multiple sessions, then they should all have different session IDs.[3]

A protocol session Σ_π consists of the *program instances* (or simply programs) of the protocol π. (These correspond to the processes that run on different computers in the concrete model of Network.) Since a single session consists multiple program instances, we refer to each program instance by a program instance ID (denoted typically as pid;)[4] We denote a program instance – with ID pid, and belonging to a session Σ – as $\wp_{\Sigma,pid}$. When the session and ID are clear or irrelevant, we simply write $\wp$. For program instances which are playing specific roles in explicit protocols (as those in Chapter 5), we refer to them using symbols like $\wp_C$ and $\wp_R$ (for committer and receiver in a commitment protocol), or $\wp_P$ and $\wp_V$ (for prover and verifier in a proof protocol).[5]

We clarify the differentiation between a program instance in a protocol session and a party in the Network. The parties are entities which "own" the program instances. A party has an identity (or more than one identity) and can own multiple program instances. Loosely, a party corresponds to a computer (or a user in a computer) in the Network, running multiple programs. A party is the "unit of corruption" in the Network: if any one program belonging to a party is corrupted, all programs of the party are automatically corrupted.[6] A party can be any set of programs in the Network, with the restriction that all *subroutine programs* of a program belong to the same party as the parent program.[7]

Computations.

Formally a program $\wp$ is a collection of variables. This includes $\mathsf{internal}_\wp$, which is the "internal state" of the program (hidden from $\mathcal{A}$, $\mathcal{Z}$ and other programs in the Network). It communicates with $\mathcal{Z}$, $\mathcal{A}$ and the other programs through input/output messages, which are denoted by variables $\mathsf{mesg}_{\mathcal{Z}\to\wp}$, $\mathsf{mesg}_{\wp\to\mathcal{A}}$, $\mathsf{mesg}_{\wp\to\wp'}$ etc. (See below for more details on modeling communication.)

Behavior of a program $\wp_\pi$ is determined by the protocol π and its inputs and internal state. The execution of the program is modeled as applying the (computable) function π to its input variables $\mathsf{input}_\wp$ (which includes messages it receives, like $\mathsf{mesg}_{\mathcal{Z}\to\wp}$ and $\mathsf{mesg}_{\mathcal{A}\to\wp}$) and internal state variables $\mathsf{internal}_\wp$ (including the internal randomness). It produces new values for the internal state variables $\mathsf{internal}_\wp$ and for the output variables $\mathsf{output}_\wp$ (which includes messages it sends, like $\mathsf{mesg}_{\wp\to\mathcal{Z}}$ and $\mathsf{mesg}_{\wp\to\mathcal{A}}$). We write it as follows.

$$(\mathsf{internal}_\wp, \mathsf{output}_\wp) \leftarrow \pi(\mathsf{input}_\wp, \mathsf{internal}_\wp).$$

Since a program would typically read its inputs multiple times and produce output (or communications) more than once, we allow the the above updating to be invoked multiple times. As explained below, each of these invocations will be scheduled by the environment as a two-step process.

Virtual Entities.

We already introduced the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ as virtual entities in our model. Another set of virtual entities correspond to "ideal functionalities."

Ideal functionalities, or functionalities for short, are modeled — like the program instances — with internal, input and output variables, and a PPT function $\mathcal{F}$. However, the variables associated with an ideal functionality instance $\wp_{\mathcal{F}}$ are not updated according to the startrun and finisrun commands. Instead (being a virtual entity) on being sent a message, $\wp_{\mathcal{F}}$ will have its variables immediately updated by applying the associated function. Ideal functionalities also differ from usual programs in that they are not associated with any parties. Since a functionality models an ideal trusted party, we do not allow the adversary to corrupt it.

Our final virtual entity is called the angel. We shall denote the angel typically by the symbol Γ. The angel does not correspond to any physical entity in the real-life scenario, but is a conceptual device to formulate security guarantees. The angel is similar to an ideal functionality, but can be accessed by the environment as well.[8] Further, we would impose different computational restrictions on functionalities and the angel. In Chapter 3 we motivate the use of this entity and describe it in more detail in Section 3.2.

2.3.2 System Execution

The system is described by various variables corresponding to the internal states of the various entities (programs, $\mathcal{Z}$, $\mathcal{A}$, functionalities and Γ), as well as by the "communication variables" (like $\mathsf{mesg}_{\mathcal{Z}\to\wp}$). The execution of the system is specified by how these variables get modified (and get created; new programs can be created, leading to new variables).

In a real-life scenario, the timing of the input and output actions of a program depend on a variety of factors[9] (speed of the processor, cache hits and misses, load on the system, computations in the protocol, the values being computed etc.). We model all this as being *determined* by the environment (recall that the environment has access to everything except the internal randomness and internal state of the program).[10]

The system execution is best described in terms of the environment's actions: $\mathcal{Z}$ repeatedly reads its variables (the ones it is allowed to read) and updates its internal variables and communication variables (the ones it is allowed to update), until it terminates with an output.[11] In addition, at each step it can invoke the computation of a program $\wp$ (existing or new), by outputting one of the two "commands": $\mathsf{startrun}_{\wp}$, when the

($\mathsf{input}_{\wp}, \mathsf{internal}_{\wp}$) are read in, and $\mathsf{finisrun}_{\wp}$ when ($\mathsf{internal}_{\wp}, \mathsf{output}_{\wp}$) are updated (according to the output of the function $\wp$ applied to the inputs read in the last time $\mathsf{startrun}_{\wp}$ was issued). The duration between the two corresponds to the time the program *runs*. For simplicity we restrict that at each step $\mathcal{Z}$ is allowed to issue at most one startrun or finisrun command. However, to have the effect of parallel execution, $\mathcal{Z}$ can issue multiple startrun commands (for different programs) before invoking the corresponding finisrun commands.[12]

Unlike the protocol programs, the virtual entities are not explicitly scheduled by $\mathcal{Z}$, but independently invoked as follows: whenever an entity outputs a message for a virtual entity, the function corresponding to latter is applied immediately, and its outputs and internal state are immediately updated with the results. (If there are messages to more than one virtual entity in a single output, all of them are invoked simultaneously and after that all their outputs and internal states are updated simultaneously.) [13]

2.3.3 Communication

The model so far does not impose any restriction in the communication between different programs. However, we need to work with a pessimistic model which assumes that all communication between different programs (but not between the programs and the ideal functionalities) is through unreliable medium, and in fact may be adversarially controlled. The virtual entity $\mathcal{A}$ is used to account for what faulty communication links or the malicious players can do, namely, not only delay or block messages, but also actively inject spurious messages into the network. This can be modeled as follows: for $\wp$ to send a message msg to a program $\wp'$ (say, running in another computer), it has to send it via $\mathcal{A}$: i.e., it can set $\mathsf{mesg}_{\wp\to\mathcal{A}}$ to (To:$\wp'$, msg), and hope that $\mathcal{A}$ will later set $\mathsf{mesg}'_{\mathcal{A}\to\wp}$ to (From:$\wp$, msg). It is up to $\mathcal{A}$ what it does with this message. Similarly, when a program $\wp$ receives a message of the form (From:$\wp'$, msg) from $\mathcal{A}$ (i.e., when the variable $\mathsf{mesg}_{\mathcal{A}\to\wp}$ is set to (From:$\wp'$, msg)), it interprets it as a message sent by $\wp'$.

Note that the adversary can not only read all the messages, but also deliver arbitrary messages. In particular it can cut off all communication between the honest players. This is not a problem, since the protocols need to be functional only when the adversary is "well-behaved." Security guarantees, however, need to be provided for this strongly pessimistic model of communication. For this, on top of this model, first we shall build authentication mechanisms (discussed below), which we shall then use in all our protocols.

Programs run by the same party can communicate with each other privately, without using the communication medium. In particular, the communication between a program and a subroutine of its internal to this program, and is not visible to $\mathcal{A}$ (or $\mathcal{Z}$).[14]

Communication with Ideal Functionalities. As we remarked above, a program $\wp$ can communicate directly with an ideal functionality $\wp_{\mathcal{F}}$ (using message variables $\mathsf{mesg}_{\wp\to\wp_{\mathcal{F}}}$

and $\mathsf{mesg}_{\wp_{\mathcal{F}}\to\wp}$). *However, the adversary schedules when these variables actually get updated.* More formally, the adversary $\mathcal{A}$ can not only update its own variables ($\mathsf{internal}_{\mathcal{A}}$ and $\mathsf{output}_{\mathcal{A}}$), but also variables of the form $\mathsf{mesg}_{\wp_{\mathcal{F}}\to\wp}$. However, this updating is done blindly, using the following provision in the system execution: the functionality $\wp_{\mathcal{F}}$ cannot directly set the variable $\mathsf{mesg}_{\wp_{\mathcal{F}}\to\wp}$; instead when it wants to send a message to $\wp$, it sets a variable $\mathsf{mesg}'_{\wp_{\mathcal{F}}\to\wp}$, and informs the adversary of this (through $\mathsf{mesg}_{\wp_{\mathcal{F}}\to\mathcal{A}}$). Whenever the adversary sets a boolean variable $\mathsf{msgrls}_{\wp_{\mathcal{F}}\to\wp}$ to 1, then the current value of $\mathsf{mesg}'_{\wp_{\mathcal{F}}\to\wp}$ gets copied to $\mathsf{mesg}_{\wp_{\mathcal{F}}\to\wp}$.

The rationale behind modeling the communication with the ideal functionality in this way is that with a pessimistic adversarial model as ours, we must settle for the possibility that the adversary can prevent some or all (honest) parties from obtaining the outputs from a functionality, or delay it indefinitely.[15] However, we point out that it is possible, and sometimes desirable,[16] to have a simpler model where the messages from functionalities are directly delivered to the programs. However since this complicates the definition of the specific functionalities we are interested in, we avoid this convention.[17]

2.3.4 Corruption of Programs

Any program instance may at any point of the execution become *corrupt*. This may be because the program deviates from the prescribed protocol (say because of a bug in the program code, in which case we could consider it corrupt from the very beginning), or because the adversary has gained access into the program's internal state. When a program $\wp$ is corrupted, all the programs belonging to the party owning $\wp$ are also corrupted. We use this convention because when a subroutine is corrupted the adversary gets access to the inputs the subroutine passed by the calling program (thereby gaining access to its internal state), and also gets to change the outputs from the subroutine which will be used by the calling program (thereby making it deviate from the protocol).[18] In our model, corruption is determined explicitly by the adversary: to corrupt a party, the adversary adds the identity of that party to a variable $\mathsf{corruptlist}$ (to which it can only add identities; the environment, all the functionalities and the angel will have read access to this variable.[19]). Then on all the variables associated with that program (internal state, as well as communication variables) are read and written by the adversary.

Static Corruption. The protocols in this work are secure with respective to a class of adversaries denoted by $\mathcal{A}^{\Gamma}_{\mathsf{static}}$. The restriction on $\mathcal{A} \in \mathcal{A}^{\Gamma}_{\mathsf{static}}$ is that a party can be corrupted only at the beginning of the Network execution (or at the beginning of the first session in which the party participates). This is called *static corruption*, as opposed to *dynamic* or *adaptive* corruption, wherein the adversary can make a decision to corrupt a party at any time of the execution.

2.3.5 Identities and Authenticated Communication.

The protocols we present later on in this work depend on being able to differentiate genuine messages from an honest party from spurious messages injected by the adversary. The solution involves using signatures. However, establishing "meaningful" identities of parties, is impossible, with no prior setups. Nevertheless, identities can be assigned to parties by using their signature verification keys as their identities. In Section 5.9 we shall explain how such a scheme can be abstracted and incorporated into our model.

For now, we shall assume the following: at the beginning of any protocol the parties involved establish a session with an authenticated message delivery functionality $\mathcal{F}_{\text{AMD}}$. In an initialization phase, $\mathcal{F}_{\text{AMD}}$ sends all players a consistent numbering for themselves and for the other players (numbered 1, 2, etc.). To send a message msg to player i, a party with numbering j sends a message $(\text{To:}i, \mathsf{msg})$ to $\mathcal{F}_{\text{AMD}}$ (all interactions with $\mathcal{F}_{\text{AMD}}$ being in the same session as in the initialization phase), which in turn sends this message to $\mathcal{A}$; later if $\mathcal{A}$ asks $\mathcal{F}_{\text{AMD}}$ to deliver this message, then $\mathcal{F}_{\text{AMD}}$ sends a message $(\text{From:}j, \mathsf{msg})$ to the party with identity i.[20] The full functionality that we will discuss later is more general, and allows each party to use "local views" of the identities of other parties consistently across multiple unrelated sessions.

2.4 Computational Considerations and Classes of Environment and Adversary

To be able to construct secure protocols for any non-trivial multi-party computation, we will need to rely on the fact the adversary (and other elements of the Network) are computationally bounded. We shall insist, loosely speaking, that all computations in the Network are Probabilistic Polynomial Time (PPT), except for the angel.[21] We proceed to specify these restrictions in detail.

The computing entities in a Network can be grouped as follows:

- The environment and program instances created by it. This also includes subroutines started by these programs.
- The adversary.
- The functionalities.
- The angel.

We shall require that the first three of these be representable as functions which can be implemented using non-uniform computations which are PPT in the security parameter k. However, there are many subtleties to be taken care of. We proceed by first reviewing the modeling of computation as functions.

Recall that execution of the Network was defined as a series of invocations of the functions on various variables standing for the internal state and the communication to and from an entity. The domain and range of these functions are merely strings (over a binary alphabet, say). To add computational restriction to such a function we will require the following. Firstly, when a program is invoked directly by the environment, or as a subroutine, it receives the security parameter k as input. A program which does not receive the correct security parameter is considered corrupt. Further we require that each program is PPT in the following sense.

Definition 2.1. *A program $\wp_\pi$ is said to be a* Probabilistic Polynomial Time *(or PPT) program if there are polynomials p and q (determined by π) such that,*

- *when invoked with security parameter k, it updates its variables as follows:*

$$(\mathsf{internal}_\wp, \mathsf{output}_\wp) \leftarrow \pi(\mathsf{input}_\wp|_{p(k)}, \mathsf{internal}_\wp|_{p(k)}),$$

  where the restriction to $p(k)$ indicates that the variable is truncated *to a string of length at most $p(k)$, and*

- *when the input variables are of length at most $p(k)$, the function π can be implemented by a (probabilistic) Turing Machine with a time bound $q(k)$.*[22]

With this convention, to ensure that the environment and all the programs run in polynomial time, it is enough to require the following:

- We shall require that all the programs are invoked by the environment are PPT. Here we use a convention regarding subroutines which needs to be highlighted. A subroutine *is not* invoked by the environment, but is considered *a part of* the program invoking it. If a protocol π involves invoking subroutines, they are also part of the Turing Machine referred to above. (Correspondingly, the variables $\mathsf{internal}_{\wp_\pi}$, $\mathsf{input}_{\wp_\pi}$ and $\mathsf{output}_{\wp_\pi}$ as used above also include the corresponding variables for all the subroutines invoked by $\wp_\pi$.) Thus the time bound $q(k)$ includes the time taken by all subroutine programs that $\wp_\pi$ may invoke directly or indirectly.

- Recall that $\mathcal{Z}$ repeatedly updates its variables, until it terminates with an output. We shall require that $\mathcal{Z}$ is PPT (in the above sense, with truncated inputs): that is the update function can be implemented by a PPT Turing Machine.[23] Further, we shall require a polynomial bound (in k) on *the number of applications* of the update function before $\mathcal{Z}$ terminates.

- We shall require the adversary and each functionality program also to be PPT in the above sense. In particular, the number of invocations is bounded by a polynomial in k.

- We do not impose any computational restrictions on the angel.

Truncating the inputs is an important technical convention we use.[24] Note that since the adversary communicates with the environment and the programs, it may choose to set the variables to strings longer than the running time of the environment and the programs. The above convention makes it clear that the extra amount of input will simply be ignored. But there is another subtle issue which is settled by this convention. This relates to the security definition and will be discussed in Section 3.3.4 (see proof of Lemma 3.2).

Classes of Environment and Adversary. The above restrictions on the environment and adversary are incorporated into the specification of the environment and adversary classes with respect to which security statements will be made. We shall use the following symbols to indicate the various classes we shall consider:

- Class $\mathcal{Z}^\Gamma$ of PPT environments which access the angel Γ. These environments will invoke a single session of a given protocol.

- Class $\mathcal{Z}_*^\Gamma$. These environments are like environments in $\mathcal{Z}^\Gamma$, but may invoke multiple sessions of a given protocol (or of protocols from a given collection).

- Class $\mathcal{A}^\Gamma$ of PPT adversaries which access the angel Γ. We do not specify any restrictions on the corruption pattern. When we want to specify that the adversary is restricted to static corruption (i.e., corrupting the parties only at the beginning of the execution of the Network), we denote the class by $\mathcal{A}^\Gamma_{\text{static}}$.

- Class $\mathring{\mathcal{A}}$. These are "nice" adversaries which do not corrupt any party, and do not communicate with the environment at all. Further they deliver all messages immediately. (This class of adversaries is used to specify when a protocol should be *functional*. Note that *security* will be required against $\mathcal{A}^\Gamma$ or $\mathcal{A}^\Gamma_{\text{static}}$.)

- Class $\mathcal{A}^\Gamma_{\text{SH}}$ and $\mathcal{A}^\Gamma_{\text{SH-static}}$. These are "semi-honest" adversaries which do not alter the behavior of the parties they corrupt. They are also called "honest-but-curious" adversaries. They can try and obtain information by observing the internals of a corrupted party. $\mathcal{A}^\Gamma_{\text{SH-static}}$ is restricted to static corruption.

- Adversary classes $\mathcal{S}^\Gamma$, $\mathcal{S}^\Gamma_{\text{static}}$, $\mathring{\mathcal{S}}$, $\mathcal{S}^\Gamma_{\text{SH}}$ and $\mathcal{S}^\Gamma_{\text{SH-static}}$ are the same as $\mathcal{A}^\Gamma$, $\mathcal{A}^\Gamma_{\text{static}}$, $\mathring{\mathcal{A}}$, $\mathcal{A}^\Gamma_{\text{SH}}$ and $\mathcal{A}^\Gamma_{\text{SH-static}}$ respectively. We use these extra symbols for notational clarity when the adversary in question is a "simulator" used in a security definition (see Definition 3.1.)

- Classes $\mathcal{Z}$, $\mathcal{A}_{\text{static}}$, $\mathcal{A}_{\text{SH-static}}$, $\mathcal{S}_{\text{static}}$, $\mathcal{S}_{\text{SH-static}}$ are classes as above, but without access to any angel.

2.5 Preliminaries

Here we collect some of the standard definitions and conventions regarding PPT computation that we will need in defining and proving security in the following chapters. We assume the reader is familiar with most of these.

An *ensemble* $\{\text{DIST}_k\}_{k=1}^{\infty}$ is a sequence of probability distributions DIST_k parameterized by k (which for us will invariably be the security parameter). Each distribution DIST_k in this ensemble is over $\{0,1\}^{\ell k}$ for some function ℓ. Often we shall refer to the ensemble $\{\text{DIST}_k\}_{k=1}^{\infty}$ by just referring to the distribution DIST_k (for a general value of k). We say two distributions DIST_k and DIST'_k are *indistinguishable* for a distinguisher M (either uniform, i.e., a Turing Machine, or more often, non-uniform, i.e., a circuit family), if the difference

$$\delta_k = |\,\mathbf{Pr}_{x\leftarrow \text{DIST}_k}[M(x)=1] - \mathbf{Pr}_{x\leftarrow \text{DIST}'_k}[M(x)=1]|$$

is *negligible*. That is, for any polynomial p, there exists some value k_0 such that for $k > k_0$, $\delta_k \leq 1/p(k)$. We denote this as

$$\text{DIST}_k \overset{M}{\approx} \text{DIST}'_k.$$

If $\ell(k) = 1$ (which is a special case of interest to us), indistinguishability by PPT distinguishers is the same as *statistical indistinguishability*. Then we write

$$\text{DIST}_k \approx \text{DIST}'_k$$

to mean that $|\,\mathbf{Pr}_{x\leftarrow \text{DIST}_k}[x=0] - \mathbf{Pr}_{x\leftarrow \text{DIST}'_k}[x=0]|$ is negligible.

2.5.1 Some Notation

In our analysis of the Network, often we will consider multiple entities together as a single entity. For instance, one could consider two parties as a combined single party (which uses multiple identities), or the environment and a party as a larger environment. Formally, this amounts to defining a Network with the two entities replaced by a new entity whose internal state is a combination of the internal states of those two parties and the variables used for (direct) communication between the two parties. This new Network behaves in exactly the same way as the old one: the combination of entities is just a syntactic modification. We refer to the new entity obtained this way by combining entities $\mathcal{E}_1, \ldots, \mathcal{E}_\ell$ by $\mathtt{comb}(\mathcal{E}_1, \ldots, \mathcal{E}_\ell)$.

2.5.2 Protocol Conventions

Most of the conventions we use regarding protocol specifications are standard in related literature (though often implicit). We highlight a few of them below.

- Actions specified for each party is taken only if the previous step in the protocol is successfully completed.
- If a party receives messages which cannot be interpreted as required by the protocol, then the party aborts the execution. Also, all the messages in a protocol are numbered serially. If a party receives a message out of order, then that party aborts the protocol.
- When we write that a party P_1 *sends* a message to P_2, we imply that the message is sent to the instance of authenticated message delivery functionality, $\wp_{\mathcal{F}_{\text{AMD}}}$, used for establishing the identities.
- Same variable names are used at the end of different participants. For instance when the protocol says "$\wp_C$ sends c, $\wp_R$ receives c," the two values of c need not be the same (though the protocol intends them to be). Later in the protocol when we refer to the same variable name c for the program $\wp_C$ it means the value it sent at that point; similarly when we use the variable name c for the program $\wp_R$ it refers to the value it received there.
- A functionality $\mathcal{F}$ is always accessed through a protocol $\langle\mathcal{F}\rangle$. Thus when specifying the behavior of $\wp_{\mathcal{F}}$, we shall also (implicitly or explicitly) specify the ideal protocol $\langle\mathcal{F}\rangle$. (In our case, except in the case of "semi-functionalities," $\langle\mathcal{F}\rangle$ is always a dummy ideal functionality, which transparently interfaces between the environment and $\langle\mathcal{F}\rangle$. See Section 3.3.3.)

2.6 Conclusion

The model of communication and computation presented in this chapter seeks to capture the complexities of a real life Network in an abstract way. So far we have not introduced any notions of security, though the model — endowed with an adversary, and incorporating restrictions on computation and access to the internals of various components — is geared towards including such notions. In the next chapter we shall see our main notion of security.

Notes

[1]Some practical implementations of secure function evaluation and other multi-party computation tasks are in vogue. However, while the functionality behind the tasks are well defined, these implementations severely lack in well defined security notions.

[2]In that we avoid low-level description of computation based on ITMs, our model resembles IO automata [SL94, Lyn96] and the model used in [PW01]. However in other respects, our model is closer to that in [Can01, Can05].

[3] We provide security guarantees only when the environment ensures that different sessions protocol sessions have unique session IDs. This is a reasonable requirement, because the local environments (the operating system or a wrapper protocol, for instance) of honest parties can ensure this. For instance a simple protocol wrapper in which each party provides a k-bit random string, and the choosing the session ID to be the concatenation of these strings can create such unique session IDs (with high probability).

[4] The program instance ID may include the session ID and an identifier for the participant in the session. For instance in a two party protocol with participants numbered 1 and 2, the IDs of program instances of a session with session ID sid can simply be $(1, sid)$ and $(2, sid)$.

[5]Though the different program instances participating in a protocol session would typically play different roles, for notational convenience we shall consider the protocol to be specified as a single function, which will behave differently depending on the input. The alternate option of defining a protocol as a collection of functions, one for each participant, is more cumbersome when considering general protocols, especially when the number of participants can be variable.

[6]In our framework where a party is the unit of corruption, it may be useful to have multiple parties within a single computer, so that even if some program is corrupted (possibly because of a bug in the code), only the party to which it belongs is corrupted.

[7]Note that though the honest parties are well-defined entities in a Network, each party in the Network may have a different view of the various parties in the Network, as the adversary can send messages claiming to come from parties with various identities. In our model (in which we do not use any prior setups), identity strings of the parties do not have any meanings, but is only used to identify different messages as belonging to the same party. As described later, this identification mechanism is abstracted as the Authenticated Message Delivery functionality $\mathcal{F}_{\text{AMD}}$, and can be implemented using a signature scheme (where in the identity strings correspond to signing keys).

[8]For the protocols in our work, the honest parties do not access the angel. However, the model allows this, and the composition theorems hold regardless.

[9]Our focus is in the internals of the environment, as our notion of security can be defined solely based on the behavior of the environment. As such, for the programs we will be concerned about accurately modeling only their input-output behavior, and not the internals. In our model, during the evolution of the system, the internals of the programs are not updated until they become visible to external entities.

[10]Modeling environment as scheduling the actions of a program – as opposed to previous formulations where scheduling was done by an external "controller function" which does not have access to the local inputs and outputs to the program – allows the model to be more widely applicable. Even so, since the environment does not have access to the internals of the program, the implicit assumption is that the scheduling of a program does not depend on the randomness used internally by the program. One way to impose such a condition in a real-world operating system, is to have a wrapper (as part of the individual programs or the operating system) that standardizes (deterministically or probabilistically) the scheduling of the input-output behavior, so that for the rest of the system, it appears independent of the internal randomness.

[11]For the purpose of defining security, we can consider the final output from the environment to be a single bit. See Section 3.3.2 for details.

[12]In previous models that appeared in the literature, the executions of the programs were explicitly sequentialized. In our vocabulary, this translates to the restricting that after a program $\wp$ is started by issuing a scheduling directive $\mathsf{startrun}_{\wp}$, the next scheduling directive must be $\mathsf{finisrun}_{\wp}$. That is to say, the *same*

program $\wp$ must be finished before another program can be started. Clearly, the new approach here models the real-life scenario more directly, by allowing different program runs to overlap with each other.

[13]The immediate invocation of virtual entities has the effect that two or more virtual entities – say $\mathcal{A}$ and a functionality – might go into an infinite loop of invoking each other, without ever letting the environment to continue. This scenario will not occur in the system, once we introduce restrictions on the total number of invocations for each virtual entity.

[14]In our protocols, we would use direct communication, without using the communication medium, mostly only between a program and its subroutines. But the model does allow any two programs within a party to indirectly communicate privately with each other via their ancestors. We make use of this when using *multi-instance sessions*, wherein within a party, programs belonging to multiple sessions will communicate with a single program (which is part of the multi-instance session) run by that party.

[15]As we have modeled it, the adversary can selectively discard some of the messages from a functionality. However, we could further insist that the adversary cannot deliver a message from a program, before delivering all previous messages from the functionality to that program. Our results in later chapters will continue to hold, by using an (implicit) convention that in all protocols communications from each party (or functionality) to every other party (or functionality) is numbered serially, and the protocols abort if any message is delivered out of order. (For messages between parties, it is crucial that the numbering is an integral part of the message as it is delivered to $\mathcal{F}_{\text{AMD}}$.)

[16]When capturing the notion of "fairness" — namely, one party can obtain the outputs from a functionality if any other party can — within the framework of Network-aware security, it is important that the "ideal world" is fair, and does not allow the adversary to block messages from the ideal functionality to any honest party. This is the convention used in [GMPY05].

[17][Can05] advocates this simplified framework where the functionalities directly communicate with the honest parties. Therein, one can require that the functionalities explicitly request the adversary for permission before delivering the messages to each honest program. Then all our results carry over to such a model. However it complicates the specification of the functionalities, and the statement of our main result.

[18]If one would like to use unreliable subroutines within a program (so that its corruption does not affect the rest of the parties), the former should be modeled as being executed by a separate party. In this case however, the communication between these two parties remains hidden from the adversary if both parties are honest. This communication can be modeled as being carried out through a private communication functionality which does not leak any information to the adversary.

[19]We insist that the environment knows about all corruptions, but it is free to ignore this information. Letting the environment know the set of corrupt parties ensures that when an adversary is replaced by a simulator, the simulator cannot corrupt any more parties than the original adversary would have corrupted. Further in the adaptive case it ensures that the corruption pattern also is indistinguishable.

Allowing the functionalities access to corruptlist allows the convenience of using (intermediate) functionalities which use this information. However a functionality using this information is termed unnatural. See section 5.2.

[20]Note that the guarantee of authenticated message delivery by $\mathcal{F}_{\text{AMD}}$ is significant only as long as the sender and receiver are uncorrupted. In the case the sender is corrupted after it sends a message msg to $\mathcal{F}_{\text{AMD}}$ (but before $\mathcal{F}_{\text{AMD}}$ delivers it), though we do not explicitly allow $\mathcal{A}$ to change the delivered message, $\mathcal{A}$ can choose to block the message msg, and send any message msg$'$ to $\mathcal{F}_{\text{AMD}}$ which may then be delivered to the receiver.

[21]We remark that though we shall impose all computational restrictions in terms of PPT Turing Machines (with or without non-uniform advice), it is possible to use other restrictions, by appropriately modifying the requirement that these functions be implementable using PPT Turing Machines. However, for our composition results to hold we will require that multiple entities in the Network could be combined in various ways, and still obtain an admissible entity. (For instance we would redefine the environment in a Network by combining the original environment and some of the programs, or the adversary.) If the class of environments and adversaries, and the functionalities and programs considered are such that such combinations do

yield admissible entities, then these results will continue to hold. Our results on protocols for multi-party computation are based on complexity theoretic assumptions regarding PPT computations. These results will continue to hold if on replacing PPT computations by another class of computations (in a way such that the composition results hold), we also change the computational assumptions appropriately. However PPT computations is by far the most useful and accepted model, and we will restrict ourselves to that.

[22]Though the Turing Machines we consider are randomized, we shall require all time bounds to be exact, and not expected.

[23]The environment (or adversary) accesses the angel through input/output, by sending a message to the angel and getting a response. As such a Turing Machine implementing the environment (or adversary) need not itself have access to the angel, but only need be able to produce queries for the angel and process the responses from the angel. Hence we can require it to be PPT.

[24]A similar convention was used in [PW01].

Chapter 3

Los Angeles Network-Aware Security

Los Angeles Network-aware security is the name for the new security framework in this thesis. As the name suggests, it is a variant — in fact a generalization — of the previous Network-aware security notions. We have already seen the abstract model of communication and computation on which this framework rests. In this chapter we present the security definitions, describe why these security guarantees are meaningful and satisfactory, and state and prove the Universal Composition theorem and an extension thereof.

3.1 A Philosophical Discussion on Defining Security

We start with a high-level philosophical discussion on the nature of security definitions developed for cryptographic protocols.

3.1.1 Relativistic Definition of Security

In modern cryptography the most robust notions of security are "relativistic." That is, instead of providing an absolute notion of security for a cryptographic construction, it is shown to be *as secure as* some other (idealized) situation. Here, the idealized scheme would typically employ idealized resources (like uncorruptable trusted parties) which are not available for a real life protocol. This "relativistic approach" immediately raises a couple of questions. First, how being "as secure as" some other scheme is defined. Second, why showing that one scheme is as secure as some other scheme is sufficient or even useful. In answering both these questions there is a leap of faith involved, moving from well-defined mathematical notions to intuitive notions of security.

"As Secure As": The Simulation Paradigm. Informally, the basic structure of the definition of "as secure as" is the following: if it is the case that whatever effect an adversary could produce when scheme 1 is employed, the same effects an adversary could produce when scheme 2 is employed, then scheme 1 is as secure as scheme 2. In other words,

for every adversary dealing with scheme 1, there is another adversary — a *simulator* — which when dealing with scheme 2 can produce (or simulate, as we shall say) the same effects as the former. Ignoring the (important) details involved in defining "the same effects" for now, we point out another important aspect of this paradigm, namely, the *class of adversaries* and the *class of simulators* with respect to which the simulation is defined. Note that naturally we will be concerned about a class of adversaries and not a particular adversary; correspondingly we will use a class of simulators rather than a single simulator.[1] What these classes should be directly relates to the next question we address below, regarding the usefulness of the "as secure as" relation. Briefly, the golden rule here is that the class of adversaries should include all adversaries which may be considered realistic threats to our deployment of scheme 1, and the class of simulators should include all adversaries which we are willing to tolerate as harmless in a deployment of scheme 2.

Significance and Meaningfulness of Relative Security. If scheme 2 is (in the intuitive sense) insecure, then the assurance that scheme 1 is as secure as scheme 2 is of little use. Then, one may wonder, how setting up this seemingly cyclic notion of relative security can be useful. The answer is that scheme 2 would be such that it would represent (intuitively) *the most secure* scheme for a particular functionality. Then, showing that scheme 1 is as secure as scheme 2 is the best possible security we can hope to have for scheme 1. Still, this might seem superfluous: why would we require the best possible security and not just a few security guarantees that would be sufficient for an application at hand? The reason — and this is perhaps the most compelling reason to use relative security notions — is that in a complex Network scenario it is often difficult to anticipate all possible attacks on a protocol and as such it is difficult to ensure whether a list of a few specific information security guarantees will be satisfactory or not. Indeed, history of cryptography has time and again witnessed definitions based on such specific guarantees failing to meet the challenges of complex interactions which were not anticipated while creating the definition.

Further, envisaging the most secure protocol (in an idealized situation) also serves the purpose of *defining the functionality* required of the protocol. Security then becomes a part of the functionality specification: *what functionality is provided by the protocol in the presence of an adversary*. Indeed, such a functionality specification (as opposed to the specification for only when there is no adversary in the Network) is desirable before deploying a protocol in a general Network.

3.1.2 Appropriate Classes for Adversaries and Simulators

We now return to the question of appropriate classes for adversaries and for simulators. Clearly the definition gets stronger when the class of adversaries grows or the class of simulators shrinks. But clearly, if we push these classes to their extremes, the definition becomes unsatisfiably strong. Thus we would like to have a definition strong enough to

be satisfactory, but weak enough to be satisfiable (with as simple complexity assumptions as possible). Early on, two conventions emerged regarding this: firstly, the class of adversaries was required to be (non-uniform) PPT machines and no larger. This is because on the one hand, in theory we model feasible computation as PPT machines (or rather super-PPT computation as the smallest robust class of infeasible computation) requiring the adversary class to be at least that big, and on the other hand, complexity theoretic assumptions of hardness against PPT adversaries are weaker and more widely studied than those against super-PPT adversaries, prompting one not to attempt providing security guarantees against larger adversary classes. The second convention is to set the class of simulators to be the same as that of the adversaries. In general, this is the best we could hope for, because irrespective of the security of the schemes under question, if the simulators must be able to simulate all the "effects" that the adversaries can produce, then they have to be at least as equipped as the adversaries.

Well-motivated and long-standing as these conventions are, in this thesis we deviate from them, and for good reason. The results of [Can01, CF01, CKL03, Lin03b] rule out the possibility of having protocols (in the standard model, without setups) that can securely realize any interesting non-trivial multi-party computation functionality.[2] Hence for us to even get started we should discard these restriction.

Security with respect to super-PPT classes. The ideal world formulation used to define a secure situation is the guarantee an end-user receives. As such, it is (typically) independent of any references to restrictions on the adversary's computational powers. It is also independent of any complexity theoretic assumptions. That is, the security is *information theoretic* in the ideal world. Thus, in this ideal world, providing the adversary with super-PPT resources does not affect the intuitive security guarantee provided. The implicit assumption here is that the only concern a user has regarding an adversary's computational power is whether it will help it attack the cryptographic constructs. Then, since in the ideal world all the cryptographic schemes will be replaced by idealized ones (using trusted entities) extra computational power to the adversary is no more of concern.

Simply allowing super-PPT classes in the security definition is not sufficient to obtain our results. If we weaken the security requirement by allowing the simulator class to be super-PPT (keeping the adversary class PPT), the impossibility results do not hold anymore, but the security definition loses the composition property. So we might consider restrengthening the definition by requiring security to hold with respect to super-PPT environment (and adversary) classes as well.[3] However a straight forward attempt at this will cause the impossibility results to return: the impossibility results from [Can01, CF01, CKL03, Lin03b] hold as long as the environment class is as powerful as the simulator class (say, both are exponential time).

The new tool which lets us define the adversary and simulator classes in such a way that composition holds and at the same time the impossibility results do not survive, is *the angel*, described below in Section 3.2. The angel, as we shall see provides the simulator,

the adversary, and the environment with a *carefully regulated access to super-PPT resources*.

Jumping ahead we mention that for the protocols in Chapters 5 and 6, the angel used provides (in a restricted way) collisions in a hash function. Note that it is plausible that having access to such collisions may not at all be useful to an adversary, outside of attacking the cryptographic schemes using it. Thus the security guarantees we provide using this angel are very close to that obtained by restricting to a PPT simulator class.

3.2 The Angel

The angel is simply an oracle with super-PPT computational powers that the adversary, simulator or environment can query and get a response from.[4] However, the angel's behavior depends on (certain very limited aspects of) the Network's state. This leads to the the unusual feature that the computational power available to the environment, adversary and simulator is dependent on the rest of the Network.

The purpose of allowing the simulator access to the angel is to facilitate the simulation by providing super-PPT computational power.[5] However for composition it is necessary that the environment also has access to the angel.[6] Typically we shall consider the "as secure as" relation with respect to classes of the form $(\mathcal{Z}^{\Gamma}, \mathcal{A}^{\Gamma}, \mathcal{S}^{\Gamma})$, which signifies that the environment, adversary and simulator all have access to the same angel Γ.[7] However, in stating the extended Universal Composition theorem (Theorem 4.2) we will find it useful to relax this convention.

The minimum features that we require of the angel (to fit the use of the angel in the following chapters) are as follows:

- The angel need not be PPT, and
- The angel keeps track of which identities in the Network are corrupted. (More generally, we could allow angels to have access to all the information that the adversary has access to.)

Looking ahead, in Chapter 5 we introduce the angel Ψ. The way Ψ depends on the state of the Network is that it refuses to answer certain queries if the query contains an identity that is not one of the corrupted identities. That is, before answering a query the angel applies a safety filter[8] to ensure that answering this query does not affect the security of the uncorrupted parties. (It is this considerate behavior, combined with its extra computational powers and the magical effect it has in overcoming the impossibilities, that gets this entity its name.)

3.3 Los Angeles Network-Aware Security

3.3.1 Ideal Functionalities and Ideal Protocols

To define security of a protocol, first we define a scenario which is considered secure, for comparison. This ideal or secure-by-definition world would be the "best possible" scenario for achieving the functionality we have in mind. Typically this ideal situation involves a trusted party with which the individual parties can communicate privately. Further such a trusted party releases to each party the minimum information required by the functionality: for instance, in a secure function evaluation functionality, the trusted party collects the inputs from all parties, and then sends each party only the outcome of the evaluation. (However, it might let the adversary specify that the output be not delivered to some of the parties.) This trusted party is the virtual entity called an ideal functionality.

To complete the specification of the ideal scenario, one needs to also specify how the parties interact with the ideal functionality: i.e., the ideal protocol. A typical ideal protocol is a simple protocol which when invoked with a session ID sid establishes a session with the ideal functionality using session ID sid, and subsequently hands (in this session) every input it receives from the environment to the functionality, and outputs every message it gets from the functionality to the environment. However we point out that it is possible, and indeed useful, to allow ideal protocols which do more than just mediate between the environment and an ideal functionality.[9]

3.3.2 A Protocol As Secure As Another

Recall that we would like to say that a protocol π is as secure as ρ if for every adversary dealing with π, there is another adversary — a *simulator* — which when dealing with ρ can produce the same effects as the former. The "effect" produced is defined as the behavior of the environment. We will say that the protocols have the same effect if *no environment can distinguish between using π and using ρ*.[10] Thus, without loss of generality,[11] the environment can be considered an experiment which outputs a single bit trying to distinguish one protocol from another.

We define the random variable $\text{EXEC}_{\pi,\mathcal{Z},\mathcal{A}}$ to stand for this one bit of output from the environment $\mathcal{Z}$ in a Network in which it invokes at most one session of the protocol π and interacts with an adversary $\mathcal{A}$. Using this notation we define the relation "as secure as."

Definition 3.1. *We say that a protocol π is as secure as ρ with respect to $(\mathcal{Z}^\Gamma, \mathcal{A}^\Gamma, \mathcal{S}^\Gamma)$ if and only if $\forall \mathcal{A} \in \mathcal{A}^\Gamma$, $\exists \mathcal{S} \in \mathcal{S}^\Gamma$ such that $\forall \mathcal{Z} \in \mathcal{Z}^\Gamma$, we have $\text{EXEC}_{\pi,\mathcal{Z},\mathcal{A}} \approx \text{EXEC}_{\rho,\mathcal{Z},\mathcal{S}}$.*

We point out that the simulator $\mathcal{S}$ can depend only on $\mathcal{A}$ and not on $\mathcal{Z}$. This ensures that $\mathcal{S}$ does not "know" any secrets that $\mathcal{Z}$ may hold.[12]

Significance of Considering All Environments. We point out that one of the main features of Network-aware security which makes it stand apart from previous definitions, is the requirement that security holds with respect to *all* environments (from a given complexity class). In contrast, the very first security definitions in cryptography used specific experiments to define security. This can be considered as requiring security with respect to a single environment.[13] This difference — between using specific environments and using general environments — is somewhat reminiscent of the difference between the older definitions of pseudorandomness in computation, which specified a battery of statistical tests for randomness, and the modern cryptographic (or complexity theoretic) definition, which considers *all* tests (of certain complexity). Significantly, this general approach makes the definition far more applicable, so that it continues to give security guarantees even when employed in arbitrary Networks. Further it is this generality that results in the Universal Composition property.[14]

Transitivity of "As Secure As" Relation. We point out that the "as secure as" relation is a partial ordering with a *bounded* transitivity property. Below is a more precise statement.

Lemma 3.1. *For any constant t (i.e, t independent of k), if we have protocols $\{\pi_i\}_{i=1}^{t}$, such that π_i is as secure as π_{i+1} with respect to $(\mathcal{Z}_i, \mathcal{A}_i, \mathcal{S}_i)$ (for $i = 1, \ldots, t-1$) and $\mathcal{S}_i \subseteq \mathcal{A}_{i+1}$, (for $i = 1, \ldots, t-2$), then π_1 is as secure as π_t with respect to $(\mathcal{Z}, \mathcal{A}_1, \mathcal{S}_t)$, where $\mathcal{Z} = \cap_{i=1}^{t} \mathcal{Z}_t$.*

PROOF: This is a consequence of the transitivity of the indistinguishability relationship used in defining "as secure as."

For $i = 1, \ldots, t$, for any $\mathcal{A}_i \in \mathcal{A}_i$ there exists $\mathcal{S}_i \in \mathcal{S}_i$ such that for all $\mathcal{Z} \in \mathcal{Z}$ we have $\text{EXEC}_{\pi_i, \mathcal{Z}, \mathcal{A}_i} \approx \text{EXEC}_{\pi_{i+1}, \mathcal{Z}, \mathcal{S}_i}$. Further for $i = 1, \ldots, t-1$ since $\mathcal{S}_i \subseteq \mathcal{A}_{i+1}$ we have $\mathcal{S}_i \in \mathcal{A}_{i+1}$. Letting $\mathcal{A}_{i+1} = \mathcal{S}_i$, given $\mathcal{A}_1 \in \mathcal{A}_1$ we can define $\{\mathcal{A}_i\}_{i=1}^{t}$ such that,

$$\begin{aligned} \text{EXEC}_{\pi_i, \mathcal{Z}, \mathcal{A}_i} &\approx \text{EXEC}_{\pi_{i+1}, \mathcal{Z}, \mathcal{A}_{i+1}} \quad \text{for } i = 1, \ldots, t-2 \\ \text{EXEC}_{\pi_{t-1}, \mathcal{Z}, \mathcal{A}_{t-1}} &\approx \text{EXEC}_{\pi_t, \mathcal{Z}, \mathcal{S}_{t-1}}. \end{aligned}$$

Since t is constant,[15] we can combine all these indistinguishabilities to get that for every $\mathcal{A}_1 \in \mathcal{A}_1$ there exists $\mathcal{S}_{t-1} \in \mathcal{S}_{t-1}$ such that for all $\mathcal{Z} \in \mathcal{Z}$ we have $\text{EXEC}_{\pi_1, \mathcal{Z}, \mathcal{A}_1} \approx \text{EXEC}_{\pi_t, \mathcal{Z}, \mathcal{S}_{t-1}}$. □

3.3.3 Secure Realization of a Functionality

To define a functionality completely, along with the ideal functionality $\mathcal{F}$, one must specify a (dummy) protocol $\langle \mathcal{F} \rangle$ to be followed by the parties in interacting with an instance $\wp_{\mathcal{F}}$ of the functionality. The dummy protocol $\langle \mathcal{F} \rangle$ typically involves just relaying to $\wp_{\mathcal{F}}$ all the inputs received, and outputting the messages received from $\wp_{\mathcal{F}}$. Then, a secure protocol for a functionality $\mathcal{F}$ is one which is as secure as $\wp_{\mathcal{F}}$.

However this does not guarantee that such a protocol will be functional: since we allow the class of adversaries to partially or completely block the communication in the protocol, we must allow the same for the class of simulators. However, with respect to such a simulator class, a trivial protocol which simply does nothing would also be a secure realization of any functionality: in the "ideal world" we can simply use the simulator $\mathcal{S}$ which blocks all messages, so that it is identical to the "real" execution. Note that even though the adversary $\mathcal{A}$ might not block any messages in the real execution, $\mathcal{S}$ is allowed to do so in the ideal world.

So we would like to have a notion of a protocol being *useful*. Interestingly, this can be done within the framework of the "as secure as" definition. For this consider a class of adversaries $\mathring{\mathcal{A}}$ under which functionality should be available when using the protocol π. Also consider a class of simulators $\mathring{\mathcal{S}}$ under which the functionality is available when using the ideal protocol $\langle\mathcal{F}\rangle$. Then, requiring that π is as secure as $\langle\mathcal{F}\rangle$ with respect to $(\mathcal{Z}^{\Gamma}, \mathring{\mathcal{A}}, \mathring{\mathcal{S}})$ ensures that with $\mathcal{A} \in \mathring{\mathcal{A}}$ the protocol remains as functional as $\langle\mathcal{F}\rangle$ is with some $\mathcal{S} \in \mathring{\mathcal{S}}$ (because if there is a difference in functionality, $\mathcal{Z}$ can notice it).

$\mathring{\mathcal{A}}$ corresponds to the conditions under which functionality must be guaranteed. We shall restrict ourselves to giving this guarantee only when this class of adversaries are "nice," i.e., they do not corrupt any parties or block any messages and do not communicate with environment.[16] Further it behaves nicely with all ideal functionalities in the Network, delivering all their messages promptly (and also for functionalities which require an explicit permission from the adversary before carrying out a task,[17]granting such permission promptly). $\mathring{\mathcal{S}}$ corresponds to the extent to which functionality guarantee is made. We shall set $\mathring{\mathcal{S}} = \mathring{\mathcal{A}}$.

Definition 3.2. *We say that a protocol π is a* useful realization *of a functionality $\mathcal{F}$ if π is as secure as $\langle\mathcal{F}\rangle$ with respect to $(\mathcal{Z}^{\Gamma}, \mathring{\mathcal{A}}, \mathring{\mathcal{S}})$.*

We combine the properties of security and usefulness into a single definition of securely realization.[18]

Definition 3.3. *We say that a protocol π*securely realizes *$\mathcal{F}$* with respect to *$(\mathcal{Z}^{\Gamma}, \mathcal{A}^{\Gamma}, \mathcal{S}^{\Gamma})$ if*

- *π is as secure as $\wp_{\mathcal{F}}$ with respect to $(\mathcal{Z}^{\Gamma}, \mathcal{A}^{\Gamma}, \mathcal{S}^{\Gamma})$,* and
- *π is a useful realization of $\mathcal{F}$. That is, π is as secure as $\langle\mathcal{F}\rangle$ with respect to $(\mathcal{Z}^{\Gamma}, \mathring{\mathcal{A}}, \mathring{\mathcal{S}})$.*

We remark that the idea of requiring simultaneous security with respect to multiple adversary and simulator classes can be further used to refine the security and functionality guarantees. [19]

3.3.4 Alternate Characterization of Security

Transvisor. The simulator required in Definition 3.1 is allowed to be very different from the adversary $\mathcal{A}$. However typically one builds such a simulator by using $\mathcal{A}$ in a "black-box" fashion. Indeed, as we shall see shortly, it will be the case that the definition in

fact implies the existence of a "black-box simulator." We formalize the notion of black-box simulation using an entity we call a *transvisor*.[20] Informally, a transvisor is an entity which translates the view of the adversary in one scenario to that in another. The translation is done in an *online* fashion. The transvisor mediates all communication of the parties and the functionalities with $\mathcal{A}$, *but does not interfere in the direct communication between* $\mathcal{A}$ *and* $\mathcal{Z}$*, or that between* $\mathcal{A}$ *and the angel*.[21] We shall denote the class of PPT transvisors with access to the angel Γ by $\mathcal{T}^{\Gamma}$.

Lemma 3.2. *A protocol π is as secure as ρ with respect to $\mathcal{Z}^{\Gamma}$, $\mathcal{A}^{\Gamma}$ and $\mathcal{S}^{\Gamma}$ if and only if, there exists a transvisor $\mathcal{T}^{\rho\to\pi} \in \mathcal{T}^{\Gamma}$ such that $\forall \mathcal{Z} \in \mathcal{Z}^{\Gamma}$ and $\forall \mathcal{A} \in \mathcal{A}^{\Gamma}$, the two random variables $\text{EXEC}_{\pi,\mathcal{Z},\mathcal{A}}$ and $\text{EXEC}_{\rho,\mathcal{Z},\mathcal{S}}$ are indistinguishable where $\mathcal{S}$ is $\mathtt{comb}(\mathcal{T}^{\rho\to\pi}, \mathcal{A})$ and $\mathcal{S} \in \mathcal{S}^{\Gamma}$.*

PROOF: The "if" part follows from the observation that since $\mathcal{A} \in \mathcal{A}^{\Gamma}$ and $\mathcal{T}^{\rho\to\pi} \in \mathcal{T}^{\Gamma}$, we have $\mathcal{S} = \mathtt{comb}(\mathcal{A}, \mathcal{T}^{\rho\to\pi}) \in \mathcal{S}^{\Gamma}$. The requirement on $\mathcal{S}$ (that $\text{EXEC}_{\pi,\mathcal{Z},\mathcal{A}} \approx \text{EXEC}_{\rho,\mathcal{Z},\mathcal{S}}$) is the same as what the definition of $\mathcal{T}^{\rho\to\pi}$ guarantees.

To prove the "only if" part we describe how a transvisor $\mathcal{T}$ is constructed, and then show that it indeed is a transvisor $\mathcal{T}^{\rho\to\pi}$.

Construction 3.1: $\mathcal{T} = \mathcal{T}^{\rho\to\pi}$

Consider $\mathcal{A}$ as split into two: a "dummy" adversary $\mathcal{A}_{\mathsf{dummy}}$ through which the rest of the adversary (which we continue to denote as $\mathcal{A}$) communicates with all the honest parties and with all functionalities. In case of adaptive corruption, $\mathcal{A}_{\mathsf{dummy}}$ mediates in the corruption process too: corrupting parties as directed by $\mathcal{A}$, and providing $\mathcal{A}$ with the the internal state of the corrupted party. That is, we insert a dummy entity to copy back and forth the communication between $\mathcal{A}$ and the parties and functionalities, and then consider $\mathtt{comb}(\mathcal{A}, \mathcal{A}_{\mathsf{dummy}})$ as the adversary.
Now, consider redefining the Network, with environment $\mathcal{Z}' = \mathtt{comb}(\mathcal{Z}, \mathcal{A})$ and adversary $\mathcal{A}' = \mathcal{A}_{\mathsf{dummy}}$.
Also note that $\mathcal{Z}' \in \mathcal{Z}^{\Gamma}$ and $\mathcal{A}' \in \mathcal{A}^{\Gamma}$. So, to this new Network, we can apply the security guarantee on π, to obtain a simulator $\mathcal{S}'$ such that

$$\text{EXEC}_{\pi,\mathcal{Z}',\mathcal{A}'} \approx \text{EXEC}_{\rho,\mathcal{Z}',\mathcal{S}'}. \tag{3.1}$$

We define $\mathcal{T}$ as $\mathcal{S}'$.

First we note that $\mathcal{T}$ is independent of $\mathcal{A}$. To see this, recall that the security guarantee ensures that $\mathcal{T}$ depends only on $\mathcal{A}'$ and not on $\mathcal{Z}'$. Further $\mathcal{A}' = \mathcal{A}_{\mathsf{dummy}}$ is independent of $\mathcal{A}$. This is true because of our convention that the programs and functionalities with which $\mathcal{A}$ communicates truncate their inputs to pre-specified polynomial lengths; then, $\mathcal{A}'$ also can use pre-specified polynomial bounds on input length depending only on the protocol specification, independent of the complexity of $\mathcal{Z}'$, and in particular inde-

pendent of how much input it may receive from $\mathcal{A}$. Thus $\mathcal{A}'$ is indeed fully specified independent of $\mathcal{Z}'$.

Next, note that the Network defined above using $\mathcal{Z}'$ and $\mathcal{A}'$ is identical to the original one using $\mathcal{Z}$ and $\mathcal{A}$, as we merely redefined the boundaries of the environment and adversary, without changing their behavior. (Recall that the angel's behavior is insensitive to such a change.) That is

$$\text{EXEC}_{\pi,\mathcal{Z},\mathcal{A}} = \text{EXEC}_{\pi,\mathcal{Z}',\mathcal{A}'}. \tag{3.2}$$

Now consider redefining the boundaries in the Network corresponding to $\text{EXEC}_{\rho,\mathcal{Z}',\mathcal{T}}$ so that $\mathcal{Z}'$ is split back into $\mathcal{Z}$ and $\mathcal{A}$; then let $\mathcal{Z}$ be the environment and $\mathcal{S} = \mathbb{comb}(\mathcal{T}, \mathcal{A})$ be the adversary. Again, this redefinition of the boundaries has no effect on the outcome of the execution. That is

$$\text{EXEC}_{\rho,\mathcal{Z}',\mathcal{T}} = \text{EXEC}_{\rho,\mathcal{Z},\mathcal{S}}. \tag{3.3}$$

Combining equations (3.1)-(3.3) we obtain that $\mathcal{T}$ is such that for any $\mathcal{A}$, with $\mathcal{S} = \mathbb{comb}(\mathcal{A}, \mathcal{T})$ we have $\text{EXEC}_{\pi,\mathcal{Z},\mathcal{A}} \approx \text{EXEC}_{\rho,\mathcal{Z},\mathcal{S}}$. Hence we conclude that $\mathcal{T}$ is indeed a transvisor $\mathcal{T}^{\rho\to\pi}$. □

3.4 Conclusion

In this chapter we presented our new definition of security. The essential difference from previous similar definitions is the use of angels in our model. We argued why this definition is meaningful and satisfactory for all practical purposes.

Before we can make full use of this definition we need to show that it enjoys the powerful universal composition property. While the security guarantees of the definition are just as meaningful without the universal composition property, composability enables us to design and analyse complex protocols. In the next chapter we shall state and prove this property, which will then be used in the subsequent chapters.

Notes

[1]Instead of simply requiring the adversary and simulator to fall into pre-defined classes, one could require that for every adversary the corresponding simulator is somehow related (in complexity) to the adversary. But this can be formulated as requiring security with respect to *multiple* pairs of fine-grain adversary and simulator classes. Such requirements provide a refined a notion of security, which requires that whatever adversary can do dealing with scheme 1, a simulator can do dealing with scheme 2 *without spending too much extra resources.*

[2]Informally, a non-trivial MPC functionality is one in which any one party getting the inputs of all other parties is not allowed. In this sense, encryption is an example — a "complete" one at that — of a trivial MPC. Indeed encryption is possible under the UC definition [Can01, Can05] with PPT adversary and simulator classes.

[3]As we shall see, for composition, we crucially depend on the fact that the environment class subsumes the adversary and simulator classes.

[4]Though we do not require in our use of the angel, one could allow angels to maintain internal state from answering one query to the next, or even establish sessions with the environment, adversary or the simulator.

[5] The explicit purpose of the angel is to help in simulation. As such, one could consider the "mother of all angels" which simply carries out the job of the transvisor for the various protocols in the Network. That is, a transvisor will act as a dummy front for the angel, which carries out the actual simulation. However, the specification of the angel directly enters the security guarantee offered by the Los Angeles Network-aware security framework (because the security is stated with respect to $(\mathcal{Z}^\Gamma, \mathcal{A}^\Gamma, \mathcal{S}^\Gamma)$, and the angel Γ is part of the specification of these classes). Thus the simpler the angel the better stated the security is.

[6]For composition the environment must be able to use the angel in the same way that an adversary or simulator uses it (so that it can internally simulate the Network). For this, one could explicitly require that the angel does not differentiate between whether it is the environment or the adversary that is communicating with it. Alternately, one could use the convention that all communication between the environment and the angel is carried out via the adversary (or the simulator).

[7]It does not make much difference if we give the adversary access to the angel or not, because one could always restrict to the dummy adversary $\mathcal{A}_{\mathsf{dummy}}$.

[8]One could incorporate the filter into $(\mathcal{Z}^\Gamma, \mathcal{A}^\Gamma, \mathcal{S}^\Gamma)$ instead of into Γ. That is, say $\mathcal{S} \in \mathcal{S}^\Gamma$ would query the angel only with inputs to which Γ would provide answers. Our notation of indicating the access to angel as part of the classes $(\mathcal{Z}^\Gamma, \mathcal{A}^\Gamma, \mathcal{S}^\Gamma)$ is consistent with this convention.

[9]An ideal protocol for a "semi-functionality" would partly resemble a usual (non-ideal) protocol, and partly a dummy protocol. For instance, for the functionality $\mathcal{F}_{\widetilde{\text{COM}}}$ in Section 5.5, the ideal protocol $\langle \mathcal{F}_{\widetilde{\text{COM}}} \rangle$ prescribes the same actions as the protocol BCOM to the party C, where as it prescribes a dummy behavior for the party R.

[10]According to our convention, the environment is oblivious to the actual code as well as the internal state of the programs in the protocol sessions it invokes. It merely gets pointers to locations where copies of the code of the protocols are kept. But note that the environment is allowed to "know" (i.e., depend on) the code of the various the protocols that we may substitute for the protocol it is invoking (though it does not know which of these protocols is actually used).

[11]The difference in effect of two protocols may be reflected only in the internals of an environment $\mathcal{Z}$. But since we consider all environments, there will be another environment $\mathcal{Z}'$ which behaves exactly like $\mathcal{Z}$, but will base its output on any difference (it can detect) in the internal state of $\mathcal{Z}$ that the protocols may cause. Thus it is enough to consider the final output of the environment. Also, allowing the environment to output more than one bit to an outside distinguisher does not make a difference because such an external distinguisher can be incorporated into the environment itself.

[12]The order of quantifiers in the security definition has drawn some attention in the literature. The standard order is $\forall\mathcal{A}\exists\mathcal{S}\forall\mathcal{Z}$. A weaker security definition is obtained by changing the order as $\forall\mathcal{A}\forall\mathcal{Z}\exists\mathcal{S}$. How-

ever, the security guarantee thus obtained is unsatisfactory because $\mathcal{S}$ is allowed to depend on any "secrets" that $\mathcal{Z}$ may hold. The composition theorem nevertheless continues to hold if the complexity of the simulator is bounded independent of that of $\mathcal{Z}$. [Can05] introduced a variant of the model, wherein $\mathcal{Z}$ accepts an input, and the security definition requires the simulation to work *for all inputs* to $\mathcal{Z}$. In this case even if $\mathcal{S}$ can depend on $\mathcal{Z}$, since it cannot depend on the input to $\mathcal{Z}$, the altered order of quantifiers offers the same security guarantee as the standard one.

[13]This view raises the following intriguing question of whether, for checking if a protocol is as secure as a particular given protocol (say $\langle \mathcal{F}_{\text{COM}} \rangle$), it is enough to consider a single "complete" environment instead of all environments. Indeed results relating older definitions of stand-alone security and newer Network-aware security definitions for tasks like encryption [Can01, Can05] and authentication [CK02] can be viewed as attempts at showing that the specific experiments described in the standalone definitions are complete environments for these tasks.

[14]Though Universal Composition is almost a direct consequence of Network-aware security, we stress that a few more technical requirements are required for it to hold: in particular, the class of environments should subsume the class of simulators.

[15]The transitivity of the indistinguishability relation holds even if $t = t(k)$ is a polynomial in k. However, if t varies with k, then the final simulator $\mathcal{S}_t$ is not well-defined anymore; instead it is a collection of simulators one for each value of k. Then, even if $\mathcal{S}_{t(k)}$ is well-defined (say if all $\mathcal{S}_i$ are the same), the new non-uniform simulator so defined may not lie in it. For instance, if $t = k$ and if all circuits in the family $\mathcal{S}_i$ are of size 2^i (independent of k, and hence constant sized circuits), the simulator defined as above is a circuit family which is of exponential size.

[16] Another good formulation of a nice adversary allows it to communicate with environment. In particular it can report all the messages that it sees, to the environment. This corresponds to an *honest but curious* setting. In this case, the simulation is non-trivial. In the protocols we present, it will be clear that the simulation used to prove security with respect to $\mathcal{A}^\Gamma$ will yield nice simulators in this sense when the adversary is nice in this sense.

[17] To be formal we should define a custom regarding how functionalities seek permission, or receive instructions, from the adversary. One could let functionalities specify an expected response from the adversary in all interactions. Then a nice adversary is defined to simply echo back this expected response.

[18]Our terminology is slightly different from that appearing elsewhere. The term "securely realizes" was used for a definition which made no reference to being functional. Instead a protocol was called "non-trivial" if it satisfied the security requirement with respect to $(\mathcal{Z}^\Gamma, \mathring{\mathcal{A}}, \mathring{\mathcal{S}})$ also.

[19]When the adversary is restricted in certain ways, we may expect to get more guarantees on the protocol, i.e., we would require the simulator also to be restricted. Here we consider only two extreme cases: when the adversarial model is very pessimistic ($\mathcal{A}^\Gamma$) and when the adversary is ideally harmless ($\mathring{\mathcal{A}}$). However, adversary (and simulator) classes for intermediate cases can be modeled similarly. For instance one could consider somewhat nice adversaries which alter the messages only by using a noisy channel, and nice simulators as before. Often our protocols can be modified so that they are secure for these classes too (in addition to being secure with respect to the standard classes). For instance, in the above example of a noisy channel adversary, a protocol could be made secure using error correcting codes to encode the communication.

[20]We avoid the name blackbox simulator because of the connotations attached to the term from the study of stand-alone security (especially of zero-knowledge proofs). There a blackbox simulator is allowed to set the randomness of $\mathcal{A}$ as well as "rewind" it. A transvisor implies a more restrictive blackbox simulator which can neither rewind nor set the randomness of the blackbox. Further a transvisor does not interfere in the communication between $\mathcal{A}$ and $\mathcal{Z}$, nor in the communication of $\mathcal{A}$ with the angel.

[21] A more general definition of transvisor would require that all communication between $\mathcal{A}$ and the angel also passes through the transvisor. This has the benefit that while carrying out compositions only the adversary needs to directly access the angel. Then we can remove the restriction that the angel's behavior is independent of whether it is interacting with the adversary or with the environment.

Chapter 4

Universal Composition

4.1 Introduction

One would like to have the guarantee that if a component in a system is replaced by another component which is as secure as the original one, then the new system would be as secure as the original system. Universal Composability of secure protocols refers to this simple, but powerful guarantee.

However, from the definition of security of protocols, no such guarantee can be immediately inferred. This is because the security of a larger system is not always implied by the security of individual components.[1] While Network-awareness guarantees security of a single session of the protocol in a general Network, Universal Composability *also* ensures simultaneous security of multiple sessions (of multiple protocols) interacting with each other. Nevertheless, in this chapter we shall see that our definition of security does offer this most remarkable property.

4.2 Composition

A simple form of protocol composition is when multiple protocol sessions (of possibly different protocols) run concurrently, but independently (except possibly for input-output interaction). A more general notion of composition also allows one protocol to use one or more other protocol sessions as subroutines. First we develop some conventions and notation to refer to composition of protocols, before stating and proving the composition theorems.

Program Invocation. A program $\wp$ is invoked (either by the environment or by another program) with initialization inputs (σ, sid, pid) where σ is the protocol (code) that specifies the actions of the program, sid is the session ID of the session to which the program belongs, and pid is a program ID, unique among all the programs in the session. Our convention regarding specifying the protocol (here σ) to the program is that, it is not the code

of σ itself that is passed, but rather a *pointer* to this code. The code itself is not read by (unreadable to) the environment (or another program invoking $\wp$ as a subroutine). The reason behind this convention is to ensure that the environment (or the calling program) does not immediately become aware if the protocol σ is substituted by another protocol ϕ.

Once invoked, a program will be given inputs and it will return outputs. If the environment $\mathcal{Z}$ starts the program, the input-output interaction is between the program and $\mathcal{Z}$.

Subroutines. Consider two program instances that belong to the same party: $\wp = \wp_{\Sigma_\pi}$ and $\wp' = \wp_{\Sigma_\sigma}$. That is $\wp$ belongs to a session Σ_π and $\wp'$ to Σ_σ. If program $\wp$ invokes $\wp'$, then the input-output interactions of $\wp'$ are with $\wp$. If all the programs in the session Σ_σ are invoked by programs in the session Σ_π, then we say that the session Σ_σ is a *subroutine* of the session Σ_π.

If in a session of protocol π, the participants may invoke one or more sessions of σ as subroutines (as instructed in the code of π) then we say the *protocol* π uses σ as a *subroutine protocol*.

Typically, when a protocol π uses a subroutine protocol σ, the latter will be formulated as an ideal protocol (interacting with an ideal functionality). Later, this subroutine may be instantiated by plugging in a real protocol ϕ (which in turn may use subroutines). We formulate this in terms of "protocol substitution."

Protocol Substitution. Suppose a protocol π uses another protocol σ as a subroutine. Then the protocol $\pi^{\phi/\sigma}$ (i.e., π using ϕ *instead of* σ.) is defined as the protocol obtained by changing invocations of each session of σ to a session of the protocol ϕ. Recall that invoking a subroutine session involves the programs passing the set of arguments (σ, sid, pid) to a subprogram, wherein the first argument is merely a pointer to a piece of code. Substitution involves replacing the code of σ at this location by that of ϕ. Here we use the term pointer in the sense of a formal object, and not necessarily a physical memory address.[2]

We also use the notion of protocol substitution when it is not another protocol, but the environment, which directly invokes σ. Our notation however will be implicit: $\text{EXEC}_{\phi,\mathcal{Z},\mathcal{A}}$ can be considered the result of substituting the protocol σ by ϕ in the Network for $\text{EXEC}_{\sigma,\mathcal{Z},\mathcal{A}}$.

4.3 Universal Composition: Basic Form

Before presenting the full-fledged form of the Universal Composition theorem, first we present a basic form.

Theorem 4.1. *If σ is as secure as ϕ with respect to $(\mathcal{Z}^\Gamma, \mathcal{A}^\Gamma, \mathcal{S}^\Gamma)$, then $\pi^{\sigma/\phi}$ is as secure as π with respect to $(\mathcal{Z}^\Gamma, \mathcal{A}^\Gamma, \mathcal{S}^\Gamma)$.*

PROOF: We need to show that $\exists \mathcal{S}^* \in \mathbb{S}^\Gamma$ such that $\forall \mathcal{Z} \in \mathbb{Z}^\Gamma$, $\forall \mathcal{A} \in \mathcal{A}^\Gamma$, we have

$$\text{EXEC}_{\pi^{\sigma/\phi},\mathcal{Z},\mathcal{A}} \approx \text{EXEC}_{\pi,\mathcal{Z},\mathcal{S}^*} \tag{4.1}$$

Equivalently, by Lemma 3.2, we need to show that there exists a transvisor $\mathcal{T}^* = \mathcal{T}^{\pi \to \pi^{\sigma/\phi}}$ such that $\forall \mathcal{Z} \in \mathbb{Z}^\Gamma$ and $\forall \mathcal{A} \in \mathcal{A}^\Gamma$, we have $\mathcal{S}^* = \mathtt{comb}(\mathcal{A}, \mathcal{T}^*)$ which satisfies equation (4.1).

Recall that the environments we are considering, $\mathcal{Z} \in \mathbb{Z}^\Gamma$, start exactly one session of the protocol (π or $\pi^{\sigma/\phi}$). However, a single session of π (resp. $\pi^{\sigma/\phi}$) may start multiple sessions of its subroutine ϕ (resp. σ). So, loosely speaking, the proof involves showing that if a single of session of σ is no less secure than one session of ϕ (this is the guarantee that the protocol σ is as secure as ϕ), then all the multiple sessions of σ are simultaneously no less secure than that many sessions of ϕ. This is achieved through a hybrid argument, as described below.

Let m be an upperbound on the number of ϕ sessions started by π, in the execution $\text{EXEC}_{\pi,\mathcal{Z},\mathcal{S}}$. We shall refer to them as Σ_ϕ^j, $j = 1, \ldots, m$. (The numbering of the sessions is simply by the order in which they are invoked, and is not tied to the session IDs.) Similarly we shall refer to the sessions of σ in $\text{EXEC}_{\pi^{\sigma/\phi},\mathcal{Z},\mathcal{A}}$ as Σ_σ^j, $j = 1, \ldots, m$.

Incrementally, we shall construct environments $\mathcal{Z}_i$ for $i = 1, \ldots, m$, such that

$$\text{EXEC}_{\sigma,\mathcal{Z}_1,\mathcal{A}} = \text{EXEC}_{\pi^{\sigma/\phi},\mathcal{Z},\mathcal{A}} \tag{4.2}$$

$$\text{EXEC}_{\phi,\mathcal{Z}_m,\mathcal{S}_m} = \text{EXEC}_{\pi,\mathcal{Z},\mathcal{S}^*} \tag{4.3}$$

$$\text{EXEC}_{\phi,\mathcal{Z}_i,\mathcal{S}_i} = \text{EXEC}_{\sigma,\mathcal{Z}_{i+1},\mathcal{A}} \quad \text{for } i = 1, \ldots, m-1 \tag{4.4}$$

where $\mathcal{S}_i$ and $\mathcal{S}^*$ are as described below.

Construction 4.1: Environment $\mathcal{Z}_i$, Transvisor $\mathcal{T}^*$ and Simulator $\mathcal{S}^*$

The construction of $\mathcal{Z}_i$ is illustrated in Figure 4.1. The construction is incrementally defined. $\mathcal{Z}_i$ internally contains sessions Σ_ϕ^j for $j < i$ and sessions Σ_σ^j for $j > i$, and will externally start a single session of σ corresponding to Σ_σ^i. Sessions Σ_ϕ^j, $j < i$, are accompanied by a copy of the transvisor $\mathcal{T}^{\phi \to \sigma}$, denoted by $\mathcal{T}_j^{\phi \to \sigma}$. We define $\mathcal{S}_j = \mathtt{comb}(\mathcal{A}, \mathcal{T}_j^{\phi \to \sigma})$.
More precisely,

$$\mathcal{Z}_i = \mathtt{comb}(\mathcal{Z}, \Sigma_{\pi \setminus \phi}, \{\Sigma_\sigma^j\}_{m \geq j > i}, \{\Sigma_\phi^j\}_{0 < j < i}, \{\mathcal{T}_j^{\phi \to \sigma}\}_{0 < j < i}),$$

where $\Sigma_{\pi \setminus \phi}$ stands for the collection of programs in Σ_π with their ϕ subroutine programs omitted. We write Σ_ϕ^j or Σ_σ^j with understanding that it is instantiated only if $\Sigma_{\pi \setminus \phi}$ invokes the j-th subroutine session.
Finally, we set $\mathcal{T}^* = \mathtt{comb}(\{\mathcal{T}_i^{\phi \to \sigma}\}_{i=1,\ldots,m})$, and $\mathcal{S}^* = \mathtt{comb}(\mathcal{A}, \mathcal{T}^*)$.

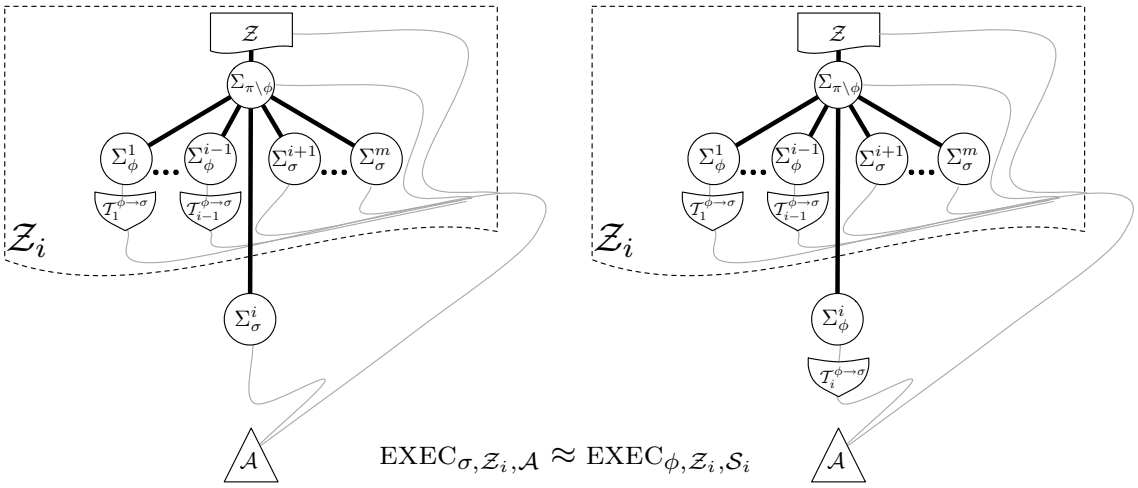

Figure 4.1: Networks corresponding to the executions $\text{EXEC}_{\sigma,\mathcal{Z}_i,\mathcal{A}}$ and $\text{EXEC}_{\phi,\mathcal{Z}_i,\mathcal{S}_i}$. $\mathcal{Z}_i$ is shown within dotted lines. The circles indicate the sessions (which consist of the protocol programs of the honest parties), as labeled. The thin lines indicate the view of $\mathcal{A}$. $\mathcal{S}_i$ (not indicated in the figure) is $\text{comb}(\mathcal{A}, \mathcal{T}_i^{\phi\to\sigma})$. Output is produced by $\mathcal{Z}$. This output will be indistinguishable between the two Networks.

We mention that the numbering in $\mathcal{T}_j^{\phi\to\sigma}$ is only to indicate which session of Σ_ϕ^j it is associated with (i.e., which session it communicates with). All $\mathcal{T}_j^{\phi\to\sigma}$ are instances of the same transvisor $\mathcal{T}^{\phi\to\sigma}$. In the same way, for all j, $\mathcal{S}_j$ is simply $\mathbb{comb}$ ($\mathcal{A}$,$\mathcal{T}^{\phi\to\sigma}$), but the numbering is used to keep track of which session of ϕ it communicates with.

Firstly, note that $\mathcal{T}^{\phi\to\sigma} \in \mathfrak{T}^\Gamma$, and so $\mathcal{Z}_i \in \mathfrak{Z}^\Gamma$ and $\mathcal{S}^* \in \mathfrak{S}^\Gamma$. Below we point out why this construction satisfies the equalities mentioned above. (Also see Figure 4.1.) To see why equation (4.2) holds, consider the Network defining the random variable $\text{EXEC}_{\sigma,\mathcal{Z}_1,\mathcal{A}}$. Note that $\mathcal{Z}_1$ does not contain any sessions Σ_ϕ^j or copies of $\mathcal{T}_j^{\phi\to\sigma}$ (because $\mathcal{Z}_i$ has Σ_ϕ^j and $\mathcal{T}_j^{\phi\to\sigma}$ only for $0 < j < i$). It internally contains all sessions Σ_σ^j except for $j = 1$, which is started externally in the execution $\text{EXEC}_{\mathcal{A},\mathcal{Z},1}$ σ. Using loose (but self-explanatory) notation, this Network (excluding $\mathcal{A}$) can be written as

$$\begin{aligned}\mathbb{comb}(\mathcal{Z}_1, \Sigma_\sigma^1) &= \mathbb{comb}(\mathcal{Z}, \Sigma_{\pi\setminus\phi}, \{\Sigma_\sigma^j\}_{m\geq j>1}, \Sigma_\sigma^1)\\ &= \mathbb{comb}(\mathcal{Z}, \Sigma_{\pi\setminus\phi}, \{\Sigma_\sigma^j\}_{m\geq j\geq 1})\\ &= \mathbb{comb}(\mathcal{Z}, \Sigma_{\pi^{\sigma/\phi}})\end{aligned}$$

which is the same as the Network (excluding $\mathcal{A}$) defining the variable $\text{EXEC}_{\pi^{\sigma/\phi},\mathcal{Z},\mathcal{A}}$.

Similarly, equation (4.3) can be verified by observing that the Network in the execu-

tion $\text{EXEC}_{\phi,\mathcal{Z}_m,\mathcal{S}_m}$ can be written as

$$\begin{aligned}\mathbb{comb}(\mathcal{Z}_m, \Sigma_\phi^m, \mathcal{S}_m) &= \mathbb{comb}(\mathcal{Z}, \Sigma_{\pi\setminus\sigma}, \{\Sigma_\phi^j\}_{0<j<m}, \{\mathcal{T}_i^{\phi\to\sigma}\}_{0<j<m}, \Sigma_\phi^m, \mathcal{S}_m)\\ &= \mathbb{comb}(\mathcal{Z}, \Sigma_{\pi\setminus\sigma}, \{\Sigma_\sigma^j\}_{0<j\le m}, \{\mathcal{T}_i^{\phi\to\sigma}\}_{0<j<m}, \mathcal{S}_m)\\ &= \mathbb{comb}(\mathcal{Z}, \Sigma_\pi, \mathcal{S}^*)\end{aligned}$$

which is the same as the Network (excluding $\mathcal{A}$) in the execution $\text{EXEC}_{\pi,\mathcal{Z},\mathcal{S}^*}$. Here we used the fact that $\mathcal{S}_m$ is in fact $\mathbb{comb}(\mathcal{A}, \mathcal{T}_m^{\phi\to\sigma})$ and $\mathcal{S}^* = \mathbb{comb}(\mathcal{A}, \{\mathcal{T}_i^{\phi\to\sigma}\}_{i=1,\ldots,m})$.

Equation (4.4) can be verified by observing that the Network on the right-hand side can be written as

$$\begin{aligned}&\mathbb{comb}(\mathcal{Z}_{i+1}, \Sigma_\sigma^{i+1}, \mathcal{A})\\ &\quad= \mathbb{comb}(\mathcal{Z}, \Sigma_{\pi\setminus\sigma}, \{\Sigma_\sigma^j\}_{m\ge j>i+1}, \{\Sigma_\phi^j\}_{0<j<i+1}, \{\mathcal{T}_i^{\phi\to\sigma}\}_{0<j<i+1}, \Sigma_\sigma^{i+1}, \mathcal{A})\\ &\quad= \mathbb{comb}(\mathcal{Z}, \Sigma_{\pi\setminus\sigma}, \{\Sigma_\sigma^j\}_{m\ge j>i}, \{\Sigma_\phi^j\}_{0<j<i}, \{\mathcal{T}_i^{\phi\to\sigma}\}_{0<j<i}, \Sigma_\phi^i, \mathcal{T}_i^{\phi\to\sigma}, \mathcal{A})\\ &\quad= \mathbb{comb}(\mathcal{Z}_i, \Sigma_\phi^i, \mathcal{S}_i)\end{aligned}$$

which is the same as the Network on the left-hand side.

Finally, we shall relate $\text{EXEC}_{\sigma,\mathcal{Z}_i,\mathcal{A}}$ and $\text{EXEC}_{\phi,\mathcal{Z}_i,\mathcal{S}}$ to complete the chain of comparisons we need to establish equation (4.1). Recall that $\mathcal{Z}_i \in \mathcal{Z}^\Gamma, \mathcal{A} \in \mathcal{A}^\Gamma$ and then $\mathcal{T}_i^{\phi\to\sigma} \in \mathcal{T}^\Gamma$ was defined, using the guarantee that σ is as secure as ϕ, as a copy of $\mathcal{T}^{\phi\to\sigma}$ such that

$$\text{EXEC}_{\sigma,\mathcal{Z}_i,\mathcal{A}} \approx \text{EXEC}_{\phi,\mathcal{Z}_i,\mathcal{S}_i} \quad \text{for } i = 1, \ldots, m \tag{4.5}$$

Combining the chain of comparisons given by equation (4.4) and equation (4.5) we obtain that, for any polynomial $m = m(k)$, $\text{EXEC}_{\sigma,\mathcal{Z}_1,\mathcal{A}} \approx \text{EXEC}_{\phi,\mathcal{Z}_m,\mathcal{S}_m}$. Then using equation (4.2) and equation (4.3), we obtain equation (4.1), as we set out to show. □

4.4 Universal Composition: Extended Form

Now we show that Los Angeles Network-aware Security is preserved under a much more general composition operation. There are various aspects in which we generalize the composition.

- Simultaneous security of multiple protocol sessions invoked by the environment: one can extend the security to hold for environments $\mathcal{Z} \in \mathcal{Z}_*^\Gamma$.
- Dealing with different $\mathcal{Z}^\Gamma$ and $\mathcal{A}^\Gamma$: one can compose protocols which may be secure with respect to different classes $(\mathcal{Z}^\Gamma, \mathcal{A}^\Gamma, \mathcal{S}^\Gamma)$. (See below for details on what ways they can be different.)
- Nested subroutines: one can allow subroutines to be nested, up to any polynomial depth of nesting.

- Joint-state subroutines: one can allow a subroutine session to be shared among multiple protocol sessions which invoke it.

We elaborate on each of these informally, before proceeding to give a formal statement and proof of the extended Universal Composition theorem. Recall that the purpose of such a composition theorem is to reason about the security of a complex system of protocols, and compare them to an idealized system. Our *modus operandi* will be to start from the idealized system, which uses idealized components, use the composition or substitution operation to replace them with more realistic components, until no more idealized components remain in the system. Finally we would like to argue that the final realistic system obtained in this way is as secure as the original idealized system we started with.

Simultaneous Security of Multiple Sessions. In defining security of a protocol we considered environments which start a single session of that protocol. But in a general Network, we would like to analyze the security of multiple sessions of the same or different protocols. It is easy to extend our security definition to such a setting, by considering (a) "protocol collections" (instead of protocols) *and* (b) a class of environments which may start multiple sessions of protocols in a protocol collection. Informally, we want a Universal Composition theorem by which the security guarantees of the individual protocols in the collection imply security by this extended definition.

First, we need to redefine the notion of "as secure as" to accommodate the fact that in a multi-session execution, the simulator's running time will depend on the number of sessions, which in turn depends on the specific environment in the Network. (Recall that originally we require the simulator — and in particular its running time — to be independent of the environment.) For this first we define the relation "as multi-session secure as." There are two changes involved: first, of course, we allow environments to start multiple sessions of the different protocols from a protocol collection; this class of environments is denoted by $\mathcal{Z}_*^\Gamma$. Secondly, we allow the simulator to be PPT in input length as well as k. That is, the simulator does not truncate its input to a polynomial fixed *a priori*.[3] We denote this class of simulators by $\mathcal{S}_{\text{iPPT}}^\Gamma$. The random variable $\text{EXEC}_{\Pi,\mathcal{Z},\mathcal{A}}$ stands for the output of the environment when it uses protocols from the protocol collection Π, in a Network with adversary $\mathcal{A}$.

Definition 4.1. *A protocol collection* Π *is said to be* as multi-session secure as Π'*, with respect to* $(\mathcal{Z}_*^\Gamma, \mathcal{A}^\Gamma, \mathcal{S}_{\text{iPPT}}^\Gamma)$ *if and only if* $\forall \mathcal{A} \in \mathcal{A}^\Gamma$, $\exists \mathcal{S} \in \mathcal{S}_{\text{iPPT}}^\Gamma$ *such that* $\forall \mathcal{Z} \in \mathcal{Z}_*^\Gamma$*, we have* $\text{EXEC}_{\Pi,\mathcal{Z},\mathcal{A}} \approx \text{EXEC}_{\Pi',\mathcal{Z},\mathcal{S}}$.

Nested Subroutines. One could apply the Basic UC theorem repeatedly to show that security is preserved if multiple protocols are substituted one after the other. This is useful if the protocols are nested, and one substitution introduces another protocol as a subroutine. However, repeated application of the UC theorem is possible only if the number of such applications is a constant (independent of the security parameter k): this

is because the transitivity of the "as secure as" relation holds only for a constant number of applications.[4] This is not satisfactory if the number of substitutions required is related to k. (Such a scenario would occur if we wish to handle recursive subroutine calls; subroutines invoked at different depths of recursion will be considered different protocols.) The extended UC theorem below shows that the security of repeated substitutions can be argued directly.

Multi-Instance Join of a Protocol. For any protocol ρ, we define the "multi-instance join" of ρ as a new protocol $\hat{\rho}$, such that a single session $\Sigma_{\hat{\rho}}$ emulates multiple sessions of ρ using internally simulated *subsessions*. Since there is only one session of $\hat{\rho}$ in the Network, it can use a fixed session ID sid_0.[5] The messages to and from this single $\hat{\rho}$ session will be tagged by a subsession ID to indicate which of the internally simulated sessions of ρ they corresponds to. We also define a *dummy protocol* $\langle\hat{\rho}\rangle$, each session of which interfaces with $\Sigma_{\hat{\rho}}$: it simply forwards inputs it receives to $\Sigma_{\hat{\rho}}$, after adding its own session ID as the subsession ID, and outputs messages it receives from $\Sigma_{\hat{\rho}}$ after stripping the subsession ID tag.

We observe that substituting ρ by $\langle\hat{\rho}\rangle$ is essentially just a syntactic change. More formally, $\langle\hat{\rho}\rangle$ is as secure as ρ, as long as $\mathcal{A}^\Gamma \subseteq \mathcal{S}^\Gamma$, because the external behavior of the programs do not change at all by this substitution. Then, in the extended UC theorem stated below, we need not make any extra provision for handling the substitution of ρ by $\langle\hat{\rho}\rangle$.[6]

Different Classes of Environment and Adversary. There are two purposes served by allowing different environment and adversary classes. Firstly, some "lower level" protocols (for instance a protocol $\pi = \text{AMD}$ for the functionality $\mathcal{F} = \mathcal{F}_{\text{AMD}}$) may be secure only with a restricted class of environments and adversaries (which do not access the angel or accesses a "weaker" angel only, for instance), compared to the rest of the protocols in the Network. Then, before substituting $\langle\mathcal{F}\rangle$ by π (after having substituted the higher level functionalities by protocols), we need to restrict ourselves to these smaller classes.[7] The second application of using a different class is that one might want to present the security guarantees with respect to an easy to understand triple $(\mathcal{Z}^\Gamma, \mathcal{A}^\Gamma, \mathcal{S}^\Gamma)$. For instance, one might want to state the final security guarantee of a protocol collection using a simulator class with access to an "upgraded" angel (as it may be easier to define). Then we would like to be able to compose it with protocols whose security is with respect to "weaker" angels. In particular we might want to state the security guarantee with respect to simulators accessing exponential time angels, where as our protocols are typically not secure against environments with access to such an angel.

In applications of interest to us, it is only the angel that we will be different in the security of the different protocols. Hence, for simplicity, we restrict ourselves to composing protocols whose security guarantees are provided with respect to different angels.

However, if necessary, it is not hard to generalize the conditions on the environment, adversary and simulator classes in our composition result to the extent to which our proof of the composition theorem still holds.

We remark that there is a severe restriction on how different the angels can be for secure composition to hold: the security of a protocol substituted later ("lower level") must use a weaker angel compared to the angel used for "higher level protocols. By a weaker angel, we mean that it should be simulatable by an environment which has access to the stronger angel. We say that the weaker angel is *subsumed* by the stronger angel.

Extended Substitution Operation. To formally describe the composition operation, we define a *t-extended substitution* operation ($t = t(k)$ typically being a polynomial in k). It is specified by a sequence of up to t tuples $\{(\phi_i, \sigma_i, \Gamma_i)\}_i$, where angels Γ_1,...,Γ_t (which may or may not be different) are such that

- Γ_i subsumes Γ_{i+1} for all i, and
- protocol σ_i is as secure as ϕ_i with respect to $(\mathcal{Z}^{\Gamma_i}, \mathcal{A}^{\Gamma_i}, \mathcal{S}^{\Gamma_i})$.

Applying this t-extended substitution to a protocol collection Π_0 involves applying the sequence of t substitutions $\Pi_i^{\sigma_i/\phi_i}$ in sequence to obtain new protocol collections $\Pi_1, \ldots, \Pi_t$.[8] The extended UC theorem will guarantee us that Π_t is as secure as Π_0 with respect to $(\mathcal{Z}_*^{\Gamma_t}, \mathcal{A}^{\Gamma_t}, \mathcal{S}_{\text{iPPT}}^{\Gamma_1})$.

Theorem 4.2. Extended UC Theorem. *Let $t = t(k)$ be bounded by a polynomial in k. Suppose the protocol collection Π_t is obtained by applying a t-extended substitution $\{(\phi_i, \sigma_i, \Gamma_i)\}_i$ to a protocol collection Π_0. Then, Π_t is as multi-session secure as Π_0 with respect to $(\mathcal{Z}_*^{\Gamma_t}, \mathcal{A}^{\Gamma_t}, \mathcal{S}_{\text{iPPT}}^{\Gamma_1})$.*

PROOF: The proof is quite similar to the proof of Theorem 4.1, but we need to accommodate the extensions into the proof. We need to show that $\exists \mathcal{S}^* \in \mathcal{S}_{\text{iPPT}}^{\Gamma_1}$ such that $\forall \mathcal{Z} \in \mathcal{Z}_*^{\Gamma_t}$, $\forall \mathcal{A} \in \mathcal{A}^{\Gamma_t}$,

$$\text{EXEC}_{\Pi_t, \mathcal{Z}, \mathcal{A}} \approx \text{EXEC}_{\Pi_0, \mathcal{Z}, \mathcal{S}^*} \tag{4.6}$$

We shall show that there exists a transvisor $\mathcal{T}^{\Pi_0 \to \Pi_t}$ such that $\forall \mathcal{Z} \in \mathcal{Z}_*^{\Gamma_t}$ and $\forall \mathcal{A} \in \mathcal{A}^{\Gamma_t}$, we have $\mathcal{S}^* = \mathtt{comb}(\mathcal{A}, \mathcal{T}^{\Pi_0 \to \Pi_t}) \in \mathcal{S}_{\text{iPPT}}^{\Gamma_1}$ which satisfies equation (4.6).

The proof proceeds along a series of hybrid arguments: an outer hybrid sequence for the t protocol substitutions, and for each of the t substitutions a sequence of m hybrids to substitute the sessions of that protocol in a Network execution. Here $m = m(k)$ is an upperbound (dependent on $\mathcal{Z}$) on the number of sessions of any protocol in Π_i, for all i. Note that since $\mathcal{Z}$ is polynomial time in k, and all the protocols are polynomial time as well, m is bounded by a polynomial in k.

As in the proof of Theorem 4.1, we describe environments which will be used to argue the security of composition. However, corresponding to the two levels of hybrids, now we have doubly indexed environments $\mathcal{Z}_{i,j}$.

Construction 4.2: Environment $\mathcal{Z}_{i,j}$, Transvisor $\mathcal{T}^{\Pi_0 \to \Pi_t}$ and Simulator $\mathcal{S}^*$

$\mathcal{Z}_{i,j}$ internally simulates the following:

- the original environment $\mathcal{Z}$
- all m sessions of protocols ϕ_ℓ, for $0 < \ell < i$,
- all m sessions of protocols σ_ℓ, for $t \geq \ell > i$,
- the first $j-1$ sessions of ϕ_i and all but the first j sessions of σ_i.
- transvisors $\mathcal{T}_{\ell,l} = \mathcal{T}^{\phi_\ell \to \sigma_\ell}$ accompanying the sessions of ϕ_ℓ for $\ell \leq i$ (the second index l in $\mathcal{T}_{\ell,l}$ merely indicates which session of ϕ_ℓ the transvisor communicates with).

$\mathcal{Z}_{i,j}$ externally starts one session of ϕ_i, which is connected to the internally simulated Network as the j-th session of ϕ_i.
We define $\mathcal{S}_{\ell,l} = \mathtt{comb}(\mathcal{A}, \mathcal{T}_{\ell,l})$.
Finally, we set $\mathcal{T}^{\Pi_0 \to \Pi_t} = \mathtt{comb}(\{\mathcal{T}_{i,j}\}_{i=1,\ldots,t;j=1,\ldots,m})$, and $\mathcal{S}^* = \mathtt{comb}(\mathcal{A}, \mathcal{T}^{\Pi_0 \to \Pi_t})$.

Similar to the Equations 4.2-4.5, we set up a chain of relations to prove equation (4.6). However now we have more relations corresponding to the longer sequence of hybrids used. For ease of reading we use the following shorthands: we denote $\text{EXEC}_{\sigma_i,\mathcal{Z}_{i,j},\mathcal{A}}$ by $\text{EXEC}_{i,j}$ and $\text{EXEC}_{\phi_i,\mathcal{Z}_{i,j},\mathcal{S}_{i,j}}$ by $\text{EXEC}'_{i,j}$. From the construction of $\mathcal{Z}_{i,j}$, one can verify the following equalities, which are all obtained by observing that the left and right-hand sides describe the same execution (in the same way we derived the Equalities 4.2-4.4).

$$\text{EXEC}_{t,1} = \text{EXEC}_{\Pi_t,\mathcal{Z},\mathcal{A}} \tag{4.7}$$

$$\text{EXEC}'_{1,m} = \text{EXEC}_{\Pi_0,\mathcal{Z},\mathcal{S}^*} \tag{4.8}$$

$$\text{EXEC}_{i,j} = \text{EXEC}'_{i,j+1} \qquad 1 \leq i \leq t, \quad 1 \leq j < m \tag{4.9}$$

$$\text{EXEC}_{i,m} = \text{EXEC}'_{i-1,1} \qquad 1 < i \leq t \tag{4.10}$$

Further, from the security guarantee that σ_i is as secure as ϕ_i, by definition of $\mathcal{S}_{i,j}$, we have

$$\text{EXEC}_{i,j} \approx \text{EXEC}'_{i,j} \;\; 1 \leq i \leq t, \; 1 \leq j \leq m \tag{4.11}$$

For this, we need to verify that the environments we constructed and the adversary are indeed in the respective classes as the security guarantee requires. Note that $\mathcal{Z}_{i,j}$ needs to internally contain transvisors $\mathcal{T}^{\phi_\ell \to \sigma_\ell}$ for $\ell \geq i$, and the original environment $\mathcal{Z}$. These transvisors access angels Γ_ℓ, $\ell \geq i$, and $\mathcal{Z} \in \mathcal{Z}_*^{\Gamma_t}$ accesses Γ_t, where as $\mathcal{Z}_{i,j}$ must access

only Γ_i. But since we imposed the condition that for $\ell \geq i$, Γ_ℓ must be subsumed by Γ_i, this is not a problem: $\mathcal{Z}_{i,j}$ can indeed simulate access to Γ_ℓ even though $\mathcal{Z}_{i,j} \in \mathcal{S}^{\Gamma_i}$. Finally note that $\mathcal{A} \in \mathcal{A}^{\Gamma_t}$ can be considered to be in $\mathcal{A}^{\Gamma_\ell}$ for all $\ell \leq t$ (formally by replacing $\mathcal{A}$ by an adversary which internally runs $\mathcal{A}$ and simulates access to Γ_t using access to Γ_ℓ). Thus indeed the transvisors $\mathcal{T}_{i,j}$ and hence $\mathcal{S}_{i,j}$ are well-defined and equation (4.11) holds.

Combining this chain of relations we obtain equation (4.6). To complete the proof we need only verify that $\mathcal{S}^* \in \mathcal{S}^{\Gamma_1}_{\text{iPPT}}$ and that $\mathcal{S}^*$ depends only on $\mathcal{A}$ (and not on $\mathcal{Z}$). The first of these follows because $\mathcal{T}^{\Pi_0 \to \Pi_t}$ is a combination of at most mt transvisors each of which is PPT and accesses an angel Γ_i which is subsumed by Γ_1. Note that $m = m(k)$ is a polynomial which depends not only on the specification of the protocols involved in the extended substitution, but also the environment. However since $\mathcal{Z}$ must communicate with $\mathcal{A}$ (and hence $\mathcal{S}^*$) for each session it starts, and since $\mathcal{S}^*$ can polynomially depend on its input size, it can charge the running time of each internal $\mathcal{T}_{i,j}$ against a communication from $\mathcal{Z}$ (multiplied by polynomial factors determined by the protocols). Thus indeed $\mathcal{S}^* \in \mathcal{S}^{\Gamma_1}_{\text{iPPT}}$. Finally, note that the specification of $\mathcal{S}^*$ involves only $\mathcal{A}$ and $\{\mathcal{T}^{\phi_\ell \to \sigma_\ell}\}_{\ell=1}^t$, which are all independent of $\mathcal{Z}$. □

4.5 Conclusion

Universal Composition theorems we have proved for Los Angeles Network-aware security closely follows the proof for the same result in the UC framework of [Can01, Can05]. Presence of the angel does not present any serious complications as the original proofs relativize with respect to the angel. Among the extensions we considered above, polynomial nesting was considered in [Can01, Can05] and multi-instance joins in [CR03].

In the following chapters, we shall use the basic composition theorem (Theorem 4.1) to prove security of complex protocols built through many substitutions. The extended composition theorem can be used to argue security of complex systems in which multiple protocols (i.e., a protocol collection) are deployed, by using a suitable extended substitution sequence. The composition theorems allow a "user" to think of a protocol as an idealized one (which it is as secure as), and assure that in virtually all situations that does not lead to hiding any security holes.

Notes

[1]This depends on the definition of security of the larger system. If it is *defined* as the security of all individual components, then clearly the composability property is trivial. However, the natural definition of simultaneous security of multiple protocols is a straight forward extension of defining security of a single protocol: one defines *a single ideal world* offering all the ideal functionalities, and requires that the real world be as secure as this ideal world.

[2]An important point regarding modeling the protocol substitution operation is that the environment need not be aware of the substitution. Indeed, we would require that the environment should not have any means to realize when a protocol substitution is carried out, except through the input-output behavior of the adversary and the programs. For the sake of being concrete one could consider a simple way of achieving this: a table for translating the protocol pointers to locations where the actual code of the protocol is kept. The environment does not get (direct) access to this table or the memory where the code is kept. Substitution of σ by ϕ simply involves altering the contents of the translation table so that the entry for σ is made to point to a location containing the code of ϕ (possibly adding this code to that location if it does not already exist). The location of code is kept unreadable only so that we can substitute with a protocol ϕ which may be "new" (i.e., fixed after fixing the environment and the unsubstituted codes).

[3]The convention of truncating the input was important only for the programs of the protocols and the functionalities. For simplicity we had imposed the condition on all entities (other than the angel). However, this convention is not suitable for simulation in the multi-session case, and hence we remove this restriction from the simulator. We could remove the restriction from the adversary as well.

[4]See Lemma 3.1 and the endnote in its proof.

[5]A fixed value of sid_0 can be used if all parties in the Network shall use a single session of $\hat{\rho}$. For instance, in the case of $\mathcal{F}_{\text{AMD}}$, it is enough to have one multi-instance session. However, if multiple sessions of $\hat{\rho}$ (with disjoint states) is desired, then the protocol should specify a way for choosing a unique session ID. See Endnote 3 of Chapter 2 on choosing unique session IDs.

[6]Our formulation of the multi-instance protocol with an accompanying dummy protocol allows us to treat the case of "Joint UC" in the same way as regular protocol substitutions. In other words, this highlights that the essence of the Joint-UC theorem from [CR03] is the fact that $\langle\hat{\rho}\rangle$ is as secure as ρ, and the composition itself follows then from the basic UC theorem.

[7]In proving the composition theorem, it may be more natural to think in terms of substituting real protocols by ideal protocols (rather than the opposite, and usual, direction in which we use substitutions). Then the "lower level" protocols (for functionalities like $\mathcal{F}_{\text{AMD}}$) are substituted first, before introducing non-trivial angels into the Network. Then, it is easier to see intuitively that for us to move from the "real world" to the "ideal world" through a chain of "as secure as" relations, these lower level protocols need not be secure against environments and adversaries having access to a non-trivial angel.

[8]Consider a protocol collection Π of n protocols, and an *ideal* protocol collection Π' in which each protocol of Π has an idealized protocol (which it is as secure as). Protocols in Π may use protocols in Π' as subroutines. Typically our aim would be to start with Π and replace the Π' subroutines with protocols from Π (which are as secure as them). Each substitution can introduce further protocols from Π' into the collection. If there are no cycles (i.e., if the relation "as secure as a subroutine used by" defines a directed acyclic graph), then in a finite number of substitutions we can remove all protocols of Π' from the collection. When recursive calls are involved, cycles can be avoided by using a bound on the depth of recursion, and defining a new protocol for each depth of recursion.

Chapter 5

Network-Aware Secure Multi-Party Computation

5.1 Introduction

In this chapter we present the central result of this thesis, namely, a protocol for general multi-party computation, which will be proven to be secure under our framework of security. The protocol uses no trust setups, and allows arbitrary corruption of parties (but statically).

In our construction we use a few basic results from previous works on multi-party computations [GMW87, CLOS02, Gol04] (whose protocols where either secure only without a general Network, or required trust on a common reference string). We use new complexity theoretic primitives, which we describe in Section 5.3. In Section 5.4 we give an overview of the entire construction before going into the details of the new component protocols and proofs relating their security to the new assumptions.

5.2 Natural Functionalities and Realistic Protocols

Our aim is to construct secure protocols for *all* (PPT) ideal functionalities. However, the formulation of ideal functionalities allows some "unnatural" functionalities as well, which are impossible to securely realize. In particular, the model allows the ideal functionality to read the list of corrupted parties; so a functionality which tells the honest parties which parties are corrupt is admissible. On the other hand, such a functionality is impossible to securely realize (with a useful protocol) if the corrupt parties are allowed to behave (semi) honestly. We classify all functionalities $\mathcal{F}$ which require $\wp_{\mathcal{F}}$ to access corruptlist , as *unnatural*. More generally, we say $\mathcal{F}$ is a natural functionality if and only if $\wp_{\mathcal{F}}$ is PPT and uses only inputs received from the participants in that session (including the adversary).[1] We shall restrict ourselves to the goal of securely realizing *all natural functionalities*.

The model allows not only unnatural functionalities, but also unrealistic protocols. An unrealistic protocol is one which communicates with an ideal functionality (possibly through one of its subroutines). Unless a trust setup is used, it is not possible to directly implement an unrealistic protocol. We shall call a protocol *realistic* if it accesses no ideal functionality (directly or though its subroutines). Our goal is to provide realistic protocols that securely realize all natural functionalities.

5.3 Complexity Assumptions and the Angel Used

Our complexity theoretic assumptions involve an (abstract) hash function. As we shall describe below, it is required to have a collision resistance property, along with some important enhancements. The angel involved in the security statement of our protocols relates to finding collisions in this hash function.

5.3.1 Non-Self-Reducible Collision-Resistant Hash Function

We denote our hash function by $\mathcal{H}$. For each value of security parameter k, it maps $\{0,1\}^k$ to $\{0,1\}^{\ell(k)}$. The k-bit input to $\mathcal{H}$ is considered to be an element $(\mu, r, x, b) \in \mathcal{I}_k \times \{0,1\}^{k_1} \times \{0,1\}^{k_2} \times \{0,1\}$, where $\mathcal{I}_k$ is the set of IDs used for the parties[2] (which is parameterized by k, and contains exponentially many valid IDs); k_1, k_2, ℓ are all polynomially related to k. $\mathcal{H}$ should be PPT computable, either uniformly or non-uniformly (accordingly our protocols will be uniform or non-uniform). Our assumptions on $\mathcal{H}$ are as follows.

A1 (Collisions and Indistinguishability): For every[3] $\mu \in \mathcal{I}_k$ and $r \in \{0,1\}^{k_1}$, there is a distribution $\mathcal{D}_r^\mu$ over $\{(x,y,z)|\mathcal{H}(\mu,r,x,0) = \mathcal{H}(\mu,r,y,1) = z\} \neq \phi$, such that for all non-uniform PPT distinguishers M

$$\{(x,z)|(x,y,z) \leftarrow \mathcal{D}_r^\mu\} \stackrel{M}{\approx} \{(x,z)|x \leftarrow \{0,1\}^{k_2}, z = \mathcal{H}(\mu,r,x,0)\}$$

$$\{(y,z)|(x,y,z) \leftarrow \mathcal{D}_r^\mu\} \stackrel{M}{\approx} \{(y,z)|y \leftarrow \{0,1\}^{k_2}, z = \mathcal{H}(\mu,r,y,1)\}.$$

Further, even if the distinguisher is given sampling access to the set of distributions $\{\mathcal{D}_{r'}^{\mu'}|\mu' \in \mathcal{I}_k, r' \in \{0,1\}^{k_1}\}$, these distributions still remain indistinguishable.

A2 (Difficult to find collisions with matching prefix): For all PPT circuits M and every ID[4] $\mu \in \mathcal{I}_k$, for a random $r \leftarrow \{0,1\}^{k_1}$, probability that $M(r)$ outputs (x,y) such that $\mathcal{H}(\mu,r,x,0) = \mathcal{H}(\mu,r,y,1)$ is negligible. This remains true even when M is given sampling access to the set of distributions $\{\mathcal{D}_{r'}^{\mu'}|\mu' \neq \mu, r' \in \{0,1\}^{k_1}\}$.

The first assumption simply states that there are collisions in the hash function, which are indistinguishable from a random hash of 0 or 1. Note that this assumption implies that for every $\mu \in \mathcal{I}_k$ and every $r \in \{0,1\}^{k_1}$ $\mathcal{H}(\mu,r,\{0,1\}^{k_2},0)$ and $\mathcal{H}(\mu,r,\{0,1\}^{k_2},1)$ are indistinguishable (because they are indistinguishable from $\{z|(x,y,z) \leftarrow \mathcal{D}_r^\mu\}$). The second assumption is an augmented collision resistance property: the augmentation requires a

non-self-reducibility property, which requires that finding collisions with one prefix (identity) cannot be reduced to finding collisions with other prefixes. However requiring that a pair of colliding values have a matching prefix makes the assumption weaker.

We term such a hash function a "non-self-reducible collision-resistant hash function."

5.3.2 The angel Ψ

Suppose $\mathfrak{X}$ is the set of corrupted parties. (Since we are dealing with static adversaries, this is a fixed set). On query (μ, r) the angel Ψ checks if $\mu \in \mathfrak{X}$, i.e., if the party with ID μ is corrupted or not. If it is, Ψ draws a sample from $\mathcal{D}_r^\mu$ described above and returns it; else it returns $\perp$. All the results in this chapter use Ψ as the angel; i.e., the security guarantees are with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\mathsf{static}}, \mathcal{S}^\Psi_{\mathsf{static}})$.

For convenience we introduce the following related notations. Ψ^* is similar to Ψ except that it answers all queries (μ, r) with a sample from $\mathcal{D}_r^\mu$, irrespective of $\mathfrak{X}$. $\Psi_{\setminus\mu}$ is exactly like Ψ^*, except that only for queries (μ', r), $\mu' \neq \mu$ does it provide a sample from $\mathcal{D}_r^{\mu'}$ (otherwise returning $\perp$). Then in Assumption A1 the adversary in question can be denoted by M^{Ψ^*}, and the one in Assumption A2 by $M^{\Psi_{\setminus\mu}}$.

5.3.3 Trapdoor Permutations

We make one more cryptographic assumption for our constructions, informally stated below:

A3 There exists a collection of trapdoor permutations T, which remains secure even if the adversary has sampling access to $\mathcal{D}_r^\mu$ for all μ and r.

More formally, we assume that there exists a PPT algorithm T which, on input 1^k, outputs a pair of deterministic polynomial sized (in k) circuits (f, f^{-1}) such that $f : \{0,1\}^k \to \{0,1\}^k$ is a permutation and f^{-1} is its inverse.[5] (In notation, we denote this as $(f, f^{-1}) \leftarrow \mathsf{T}(1^k)$ or $(f, f^{-1}) = \mathsf{T}(1^k; r)$ where r is the randomness used by T to sample (f, f^{-1}).) Also we assume there is an associated boolean function B. The hardness assumption regarding T and B is that, for any PPT machine M^{Ψ^*} (with sampling access to $\mathcal{D}_r^\mu$ for all μ and r), the following quantities are negligible:

$$\mathbf{Pr}_{(f,f^{-1})\leftarrow\mathsf{T}(1^k),x\leftarrow\{0,1\}^k}\left[M^{\Psi^*}(f,x) = f^{-1}(x)\right]$$

$$\mathbf{Pr}_{(f,f^{-1})\leftarrow\mathsf{T}(1^k),x\leftarrow\{0,1\}^k}\left[M^{\Psi^*}(f,x) = \mathsf{B}(f^{-1}(x))\right] - \frac{1}{2}$$

In fact, it is enough to assume that there is a collection of trapdoor permutations T' which satisfies the first of the above two conditions. The existence of T with an associated hard-core predicate B follows from it. For instance, the Goldreich-Levin construction [GL89]

(which takes T' and yields T and B) works even when the adversary M^{Ψ^*} is given sampling access to $\mathcal{D}_r^\mu$. This is easy to see because the proof in [GL89] uses M^{Ψ^*} as a black-box.

Also, we need a perfectly binding (non-interactive) commitment scheme C, whose hiding property (in a stand-alone setting) holds against PPT adversaries M^{Ψ^*} with access to the distributions $\mathcal{D}_r^\mu$ for all μ and r. More formally, we assume that there is a PPT algorithm C such that for all strings a, a', r, r', if $a \neq a'$ then $\mathsf{C}(a;r) \neq \mathsf{C}(a';r')$. Also, for any PPT machine M^{Ψ^*} (with sampling access to $\mathcal{D}_r^\mu$ for all μ and r), the following quantity is negligible:

$$|\,\mathbf{Pr}_{a\leftarrow\{0,1\}^k, c\leftarrow\mathsf{C}(a)}\left[M^{\Psi^*}(c,a)=1\right] - \mathbf{Pr}_{a,a'\leftarrow\{0,1\}^k, c\leftarrow\mathsf{C}(a')}\left[M^{\Psi^*}(c,a)=1\right]|.$$

Here we write $\mathsf{C}(a;r)$ to denote the commitment to the string a using randomness r, and $\mathsf{C}(a)$ stands for $\mathsf{C}(a;r)$ with a random r. Such a scheme C can be readily constructed from T and B (see for instance [Gol01, Sec. 4.4.1.2]).

5.3.4 Remarks on the Assumptions

To begin with, it will be useful to point out how our assumptions relate to the convention of modeling hash functions as "random oracles." The random oracle model, as put forth in [BR93], has become an extremely popular way to build efficient "secure" protocols in practice. If our hash function $\mathcal{H}$ were to be replaced by access to a random oracle, then Assumption A1 and Assumption A2 become unconditionally true; Assumption A3 reduces to the standard assumption on trapdoor permutations. However, an increasing body of work [CGH98, GT03, BBP04] points out the fundamental problems in the random oracle model. These problems stem from the fact that computation using a random oracle offers many properties which a computation in the standard model does not have. Our assumptions on the other hand are concrete complexity theoretic assumptions which fit the standard model just like all other complexity theoretic assumptions used in cryptography. Thus — and this has been one motivation in proposing them — these assumptions capture some of the intuition behind modeling hash functions as random oracle, while staying fully within the standard model of assumptions. We list below in intuitive terms the properties of $\mathcal{H}$ suggested by our assumptions.

- *Collisions and Indistinguishability.* A length reducing hash function (say $\ell = k/2$) will typically have exponentially many pre-images for each point in the range. Then it is plausible that collisions can be sampled such that each of the two pre-images seems to be distributed as if it was sampled "naturally." Note that this sampling process is allowed to be inefficient.

- *Collision Resistance.* Collision resistance by itself is a natural assumption on hash functions. What makes our assumption non-standard is that even when the ad-

versary has access to samples of collisions, it should be infeasible for it to find *new collisions with a given random* r. In fact, our assumption is weaker: we require the collisions to have the same "prefix" (the first argument to $\mathcal{H}$, namely μ), and the adversary gets only collisions with other prefixes. The intuition here is that once the prefix is changed, the hash function behaves entirely differently, and collisions with another prefix give no significant advantage to the adversary. We also point out that the collision is required not only to match in the first two arguments (μ and r, the latter being a random challenge to the adversary), but also should differ in the last single-bit argument.[6]

- *Trapdoor Permutation.* Again, trapdoor permutation by itself is a standard assumption. The intuition behind strengthening this assumption is that collision samples from a hash function, which could have much less "structure" than a trapdoor function, and presumably based on very different mathematical objects, do not give much advantage to an adversary in breaking the security of the trapdoor permutation. Indeed, by appropriately choosing the security parameters associated with the construction of $\mathcal{H}$ and of the trapdoor permutation, we can replace Assumption A3 with a more standard assumption of trapdoor permutations secure against super-polynomial (sub-exponential) adversaries. See Section 5.10 for details.

By way of plausibility of the assumptions we point out that in a recent work [MMY05], it was shown that a hash function satisfying our assumptions can be instantiated based on (non-standard) number theoretic assumptions (an enhanced version of the Discrete Log Assumption).

Finally, we stress that our model of Los Angeles Network-aware security is useful independent of the specific assumptions we have made for the constructions in this chapter. It is possible that alternate constructions (employing different assumptions and a different angel) can yield the same results. Indeed, recently Barak and Sahai [BS05] showed that based on collision resistant hash functions and trapdoor functions which are secure against super-polynomial (sub-exponential) adversaries (with no reference to any angel), Los Angeles Network-aware secure MPC protocols can be constructed.[7]

5.4 Overview of the Protocol

An overview of the construction of our Secure multi-party computation protocol is shown in Table 5.4. The overall structure of our protocol follows the structure in [CLOS02], which in turn follows that in [GMW87, Gol04]. But our construction differs from that [CLOS02] in a very crucial manner: *none of our protocols use a Common Random String*.

The differences in our construction appear after the first three rows in Table 5.4. Correspondingly, we present our construction in two parts: a protocol OMCP-REAL which captures all the rows after the first three in Table 5.4, and a protocol for multi-party computation which uses OMCP-REAL.

$\pi_1 = \text{GMW}(\mathcal{F})$	π_1 is as secure as $\langle\mathcal{F}\rangle$ with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{SH-static}}, \mathcal{S}^\Psi_{\text{SH-static}})$	From [GMW87, CLOS02, Gol04] (Lemma 5.2)
$\pi_2 = \pi_1^{\text{OT}/\langle\mathcal{F}_{\text{OT}}\rangle}$	π_2 is as secure as π_1 with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{SH-static}}, \mathcal{S}^\Psi_{\text{SH-static}})$	From [GMW87, CLOS02, Gol04] (Lemma 5.3)
$\pi_3 = \text{COMPILER}(\pi_2)$	π_3 is as secure as π_2 with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{static}}, \mathcal{S}^\Psi_{\text{SH-static}})$	From [GMW87, CLOS02, Gol04] (Lemma 5.4)
$\pi_4 = \pi_3^{\text{OMCP}/\langle\mathcal{F}_{\text{CP}}^{\text{1:M}}\rangle}$	π_4 is as secure as π_3 with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{static}}, \mathcal{S}^\Psi_{\text{static}})$	Lemma 5.14. Also [CLOS02]
$\pi_5 = \pi_4^{(\langle\text{BC}\rangle,\langle\text{ZK}\rangle)/(\mathcal{F}_{\text{BC}},\mathcal{F}_{\text{ZK}})}$	π_5 is as secure as π_4 with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{static}}, \mathcal{S}^\Psi_{\text{static}})$	From [GL02, CF01]
$\pi_6 = \pi_5^{\text{COM}/\langle\mathcal{F}_{\text{COM}}\rangle}$	π_6 is as secure as π_5 with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{static}}, \mathcal{S}^\Psi_{\text{static}})$	Lemma 5.11
$\pi_7 = \pi_6^{\text{BZK}/\langle\mathcal{F}_{\widetilde{\text{ZK}}}\rangle}$	π_7 is as secure as π_6 with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{static}}, \mathcal{S}^\Psi_{\text{static}})$	Lemma 5.9
$\pi_8 = \pi_7^{\text{BMCOM}/\langle\mathcal{F}_{\widetilde{\text{MCOM}}}\rangle}$	π_8 is as secure as π_7 with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{static}}, \mathcal{S}^\Psi_{\text{static}})$	Lemma 5.8
$\pi_9 = \pi_8^{\text{BCOM}/\langle\mathcal{F}_{\widetilde{\text{COM}}}\rangle}$	π_9 is as secure as π_8 with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{static}}, \mathcal{S}^\Psi_{\text{static}})$	Lemma 5.5
$\pi_{10} = \pi_9^{\text{ENC}/\langle\mathcal{F}_{\text{ENC}}\rangle}$	π_{10} is as secure as π_9 with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{static}}, \mathcal{S}^\Psi_{\text{static}})$	From [DDN00, Sah99, Lin03c, Can01] (Lemma 5.16)
$\pi_{11} = \pi_{10}^{\text{AMD}/\langle\mathcal{F}_{\text{AMD}}\rangle}$	π_{11} is as secure as π_{10} with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{static}}, \mathcal{S}^\Psi_{\text{static}})$	Lemma 5.18

Table 5.1: Overview of the construction of a secure protocol for a functionality $\mathcal{F}$. $\pi_1 1$ does not use any subroutines. Here $\mathcal{S}^\Psi_{\text{static}} = \mathcal{A}^\Psi_{\text{static}}$, and $\mathcal{S}^\Psi_{\text{SH-static}} = \mathcal{A}^\Psi_{\text{SH-static}}$. Combining these relations we get π_{11} is as secure as $\langle\mathcal{F}\rangle$ with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{static}}, \mathcal{S}^\Psi_{\text{SH-static}})$.

5.4.1 One-to-many Commit-and-prove

We construct a realistic protocol OMCP-REAL which securely realizes $\mathcal{F}_{\text{CP}}^{\text{1:M}}$ with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{static}}, \mathcal{S}^\Psi_{\text{static}})$, where $\mathcal{F}_{\text{CP}}^{\text{1:M}}$ is the "One-to-Many Commit-and-Prove" functionality

shown in Figure 5.11 (see Section 5.7). The main components used in this construction are built in Sections 5.5 and 5.6. In Section 5.7 we use them to complete the construction of OMCP-REAL. There we prove the following lemma, which encapsulates all the major new components in our protocol.

Lemma 5.1. *Under assumptions A1, A2 and A3, there is a realistic protocol* OMCP-REAL *which securely realizes* $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ *with respect to* $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\text{static}}, \mathcal{S}^{\Psi}_{\text{static}})$.

5.4.2 MPC from Commit-and-prove

Given Lemma 5.1, the rest of the construction closely follows that in [CLOS02]. First, using results from [GMW87, CLOS02], we begin with a protocol which securely realizes $\mathcal{F}$ with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\text{SH-static}}, \mathcal{S}^{\Psi}_{\text{SH-static}})$. $\mathcal{A}^{\Psi}_{\text{SH-static}}$ (resp., $\mathcal{S}^{\Psi}_{\text{SH-static}}$) is the class of semi-honest adversaries (resp., simulators) which do not alter the behavior of the parties they corrupt. Then we construct a *protocol compiler* which can take a protocol secure against semi-honest (static) adversaries and generate a protocol secure against general (static) adversaries, thereby completing the proof. These two steps are further elaborated below.

MPC for Semi-Honest Parties. In general, the proofs for the semi-honest case from [CLOS02] are mostly information-theoretic, and immediately imply their analogs that we use.

For any multi-party functionality $\mathcal{F}$, Goldreich, Micali and Wigderson [GMW87] gave a protocol construction for securely realizing $\mathcal{F}$ against *semi-honest* adversaries. Canetti et. al. [CLOS02] formulate (an extension of) this protocol — which we shall call $\text{GMW}(\mathcal{F})$— as a protocol accessing the Oblivious Transfer functionality (denoted by $\mathcal{F}_{\text{OT}}$), and show that for any natural functionality $\mathcal{F}$, the protocol $\text{GMW}(\mathcal{F})$ securely realizes $\mathcal{F}$ with respect to $(\mathcal{Z}^{\Gamma}, \mathcal{A}^{\Gamma}_{\text{SH-static}}, \mathcal{S}^{\Gamma}_{\text{SH-static}})$. As observed there, the proof is information theoretic and holds not just for PPT classes. In particular it holds with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\text{SH-static}}, \mathcal{S}^{\Psi}_{\text{SH-static}})$. We record this below.

Lemma 5.2. (Following [GMW87, CLOS02, Gol04]): *For any multi-party functionality* $\mathcal{F}$*, the protocol* $\text{GMW}(\mathcal{F})$ *(which uses* $\langle \mathcal{F}_{\text{OT}} \rangle$ *as a subroutine) securely realizes* $\mathcal{F}$ *with respect to* $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\text{SH-static}}, \mathcal{S}^{\Psi}_{\text{SH-static}})$.

[GMW87, CLOS02] show that there is a protocol which securely realizes $\mathcal{F}_{\text{OT}}$ with respect to $(\mathcal{Z}, \mathcal{A}_{\text{SH-static}}, \mathcal{S}_{\text{SH-static}})$. We observe that in fact the same protocol securely realizes $\mathcal{F}_{\text{OT}}$ with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\text{SH-static}}, \mathcal{S}^{\Psi}_{\text{SH-static}})$, if we use the trapdoor permutations from Assumption A3 in that protocol. The reason for this is that the only computational assumption used there is that of the security of trapdoor permutations. Using the trapdoor permutations from Assumption A3, the same security holds when all the entities involved have access to Ψ. We record this observation below.

Lemma 5.3. (Following [GMW87, CLOS02]): *Under Assumption A3, there is a protocol* OT *which securely realizes* $\mathcal{F}_{\text{OT}}$ *with respect to* $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\text{SH-static}}, \mathcal{S}^{\Psi}_{\text{SH-static}})$.

Protocol Compiler As mentioned above, to complete the construction, we need to show how to convert the above protocol for semi-honest parties into one secure against malicious parties. Following [GMW87], [CLOS02] gives a "compiler" which carries out this conversion, and produces a new protocol which uses $\langle \mathcal{F}_{\text{CP}}^{1:\text{M}} \rangle$ as a subroutine. The proof given in [CLOS02] that this compiled protocol is secure (given access to $\mathcal{F}_{\text{CP}}^{1:\text{M}}$) is again information-theoretic, and holds not just for PPT adversaries and environments, but also when they have access to an angel. Hence we have the following.

Lemma 5.4. (Following [CLOS02]): *There exists a protocol compiler* COMPILER *which takes a multi-party protocol π, and outputs a protocol* COMPILER(π) *(which uses $\langle \mathcal{F}_{\text{CP}}^{1:\text{M}} \rangle$ as a subroutine) such that, for every protocol π, the compiled protocol* COMPILER(π) *is as secure as π with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\text{static}}, \mathcal{S}^{\Psi}_{\text{SH-static}})$.*

Our main theorem readily follows from the above results, using the Universal Composition theorem (Theorem 4.1).

Theorem 5.1. *Under assumptions A1, A2 and A3, for any natural multi-party functionality $\mathcal{F}$, there is a realistic protocol which securely realizes $\mathcal{F}$ with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\text{static}}, \mathcal{S}^{\Psi}_{\text{static}})$.*

PROOF: Consider any natural multi-party functionality $\mathcal{F}$. By Lemma 5.2 there is a protocol π which securely realizes $\mathcal{F}$ with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\text{SH-static}}, \mathcal{S}^{\Psi}_{\text{SH-static}})$. Applying Lemma 5.4, we obtain a protocol $\pi' = \text{COMPILER}(\pi)$ which securely realizes $\mathcal{F}$ with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\text{static}}, \mathcal{S}^{\Psi}_{\text{static}})$ (i.e., secure against malicious adversaries as well). $\langle \mathcal{F}_{\text{CP}}^{1:\text{M}} \rangle$ is the only part of the protocol which accesses an ideal functionality. Finally using Lemma 5.1 and the composition theorem, we get that $\pi'^{\text{OMCP-REAL}/\mathcal{F}_{\text{CP}}^{1:\text{M}}}$ securely realizes $\mathcal{F}$ with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\text{static}}, \mathcal{S}^{\Psi}_{\text{static}})$. Observing that $\pi'^{\text{OMCP-REAL}/\mathcal{F}_{\text{CP}}^{1:\text{M}}}$ is realistic completes the proof. □

5.5 Basic Building Blocks

This section develops the basic tools we will need to securely realize the *commitment functionality* (Section 5.6). Here we introduce a new modeling and proof technique based on intermediate non-standard functionalities called *semi-functionalities*. Viewed as standard functionalities these intermediate functionalities do not fully capture the security properties we need from our protocols for their later application. Hence we shall formulate some of their security properties outside the security definition ("as secure as") that we have developed. (In Chapter 6 these proof techniques are extended to a full fledged definitional framework.)

5.5.1 Basic Commitment Protocol

In Figure 5.1 we give a protocol BCOM for commitment, which uses $\langle \mathcal{F}_{\text{ENC}} \rangle$ as a subroutine. $\mathcal{F}_{\text{ENC}}$ is the encryption functionality, which receives a message from a program and

delivers it to the destination program (if instructed by the adversary), after publishing only the length of the message to the adversary. (See Section 5.8.)

Protocol BCOM

The participants are a sender or committer $\wp_C$, and a receiver $\wp_R$. The security parameter is k, and k_1, k_2 are two parameters such that k, k_1 and k_2 are polynomially related. The sender $\wp_C$ gets as input a bit b, which it must commit to.

COMMIT PHASE:

1. $\wp_R$ picks $r \leftarrow \{0,1\}^{k_1}$ and sends it to $\wp_C$.
2. $\wp_C$ chooses $r' \leftarrow \{0,1\}^{k_2}$ and computes $c = \mathcal{H}(\mu_R, r, r', b)$. $\wp_C$ and $\wp_R$ start a subroutine session $\Sigma_{\langle \mathcal{F}_{\text{ENC}} \rangle}$. Let $\wp_{\mathcal{F}_{\text{ENC}}}$ denote the instance of the $\mathcal{F}_{\text{ENC}}$ functionality. $\wp_C$ sends c to $\wp_R$ via $\wp_{\mathcal{F}_{\text{ENC}}}$.[8]
3. $\wp_R$ receives c from $\wp_{\mathcal{F}_{\text{ENC}}}$ and accepts the commitment.

REVEAL PHASE:

1. $\wp_C$ sends (b, r') to $\wp_R$ via $\wp_{\mathcal{F}_{\text{ENC}}}$. The receiver $\wp_R$ receives (b, r').
2. $\wp_R$ checks if $\mathcal{H}(\mu_R, r, r', b) = c$. If so it accepts b as revealed.

Figure 5.1: The Basic Commitment Protocol BCOM (which uses $\mathcal{F}_{\text{ENC}}$).

Protocol $\langle \mathcal{F}_{\widetilde{\text{COM}}} \rangle$ and Functionality $\mathcal{F}_{\widetilde{\text{COM}}}$

A sender C and a receiver R run the protocol $\langle \mathcal{F}_{\widetilde{\text{COM}}} \rangle$ to interact with $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$. The adversary is $\mathcal{S}$. Input to $\wp_C$ is a bit b to which it commits (and later an instruction to reveal the commitment). The security parameter is k, and k_1, k_2 are two parameters such that k, k_1 and k_2 are polynomially related.

COMMIT PHASE:

1. $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ picks $r \leftarrow \{0,1\}^{k_1}$ and sends it to $\wp_C$.
2. $\wp_C$ chooses $r' \leftarrow \{0,1\}^{k_2}$ and computes $c = \mathcal{H}(\mu_R, r, r', b)$. $\wp_C$ sends c to $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$.
3. On receiving c from $\wp_C$, $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ sends the message commit to $\wp_R$

REVEAL PHASE:

1. $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ receives (b, r') from $\wp_C$
2. $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ checks if $\mathcal{H}(\mu_R, r, r', b) = c$. If so it sends the message (reveal , b) to $\wp_R$ and the adversary $\mathcal{S}$.

Figure 5.2: A functionality realized by the protocol BCOM

We will use protocol BCOM as a component in later protocols. Thus we would like to show some sort of composable security for this protocol. But this protocol cannot be a secure protocol for the commitment functionality.[9] So we introduce a novel technique to formalize and analyze the security of this protocol. In Figure 5.2 we show a functionality

$\mathcal{F}_{\widetilde{\text{COM}}}$ specifically designed to capture the hiding property BCOM. It also retains any binding property that BCOM has. In the two following lemmas, we show that BCOM indeed is as secure as the dummy protocol $\langle \mathcal{F}_{\widetilde{\text{COM}}} \rangle$ (which accesses a copy of $\mathcal{F}_{\widetilde{\text{COM}}}$), and also that $\mathcal{F}_{\widetilde{\text{COM}}}$ offers some binding guarantee.

Lemma 5.5. *Protocol* BCOM *is as secure as* $\langle \mathcal{F}_{\widetilde{\text{COM}}} \rangle$ *with respect to* $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$*, under Assumption A1.*

PROOF: For every adversary $\mathcal{A} \in \mathcal{A}^{\Psi}_{\mathsf{static}}$ we must demonstrate a PPT transvisor $\mathcal{T} \in \mathcal{T}^{\Psi}$ such that no environment $\mathcal{Z} \in \mathcal{Z}^{\Psi}$ can distinguish between interacting with the parties running BCOM and $\mathcal{A}$ (the "real" world) on the one hand, and interacting with the parties running $\langle \mathcal{F}_{\widetilde{\text{COM}}} \rangle$ and $\mathcal{S} = \mathtt{comb}(\mathcal{A}, \mathcal{T})$ (the "ideal" world) on the other. First, we describe $\mathcal{T}$.

Construction 5.1: Transvisor $\mathcal{T} = \mathcal{T}^{\langle \mathcal{F}_{\widetilde{\text{COM}}} \rangle \to \text{BCOM}}$

$\mathcal{T}$ simulates the honest participants internally. Below $\tilde{\wp}_C$ stands for the program of the sender simulated internally by $\mathcal{T}$ (if C is honest) and $\tilde{\wp}_R$ for that of the receiver (simulated if R is honest). Also it internally simulates $\wp_{\mathcal{F}_{\text{ENC}}}$ (denoted by $\tilde{\wp}_{\mathcal{F}_{\text{ENC}}}$). It behaves as follows depending on which parties are corrupted.

Both C, R corrupt. If $\mathcal{A}$ corrupts both participants, $\mathcal{T}$ acts transparently. Then the simulation is trivially perfect.

Both C, R honest. If $\mathcal{A}$ corrupts neither of the two participants C and R, then in the "real world" all it sees are the random string r from $\wp_R$ to $\wp_C$, and the message from $\mathcal{F}_{\text{ENC}}$ giving the length of the commit and reveal messages from $\wp_C$. So using the parameters k_1, k_2 and the lengths of the messages, $\mathcal{T}$ (i.e., $\tilde{\wp}_C$, $\tilde{\wp}_R$ and $\tilde{\wp}_{\mathcal{F}_{\text{ENC}}}$) can perfectly simulate the protocol to $\mathcal{A}$. (Encryption is used in the protocol specifically to take care of this situation where the adversary corrupts neither participants.)

R honest, C corrupted. Suppose the adversary corrupts only the sender C. Note that there is very little difference between what $\wp_C$ sees in the real and ideal executions. In fact $\tilde{\wp}_R$ will simply act as an intermediary, letting $\wp_C$ directly talk to $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$: it will simply forward the messages from corrupt $\wp_C$ (received through $\tilde{\wp}_{\mathcal{F}_{\text{ENC}}}$) to $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ and reports back to $\mathcal{A}$ the messages from $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$. It is easily verified that this is a perfect simulation.

C honest, R corrupted. Finally, suppose that the adversary corrupts the receiver alone. When, on behalf of $\wp_R$, $\mathcal{A}$ sends out the first message r in the protocol, $\mathcal{T}$ intercepts it, and sends a query (μ_R, r) to the angel Ψ. Since R is cor-

rupted, Ψ will respond with $(x, y, z) \leftarrow \mathcal{D}_r^{\mu_R}$, where $\mathcal{D}_r^{\mu_R}$ is the distribution over $\{(x, y, z) | \mathcal{H}(\mu_R, r, x, 0) = \mathcal{H}(\mu_R, r, y, 1) = z\}$ as specified in Assumption A1. Then, when $\wp_{\widetilde{\mathcal{F}_{\text{COM}}}}$ gives the commit message, $\tilde{\wp}_C$ sends z to $\mathcal{A}$. Later if $\wp_{\widetilde{\mathcal{F}_{\text{COM}}}}$ gives the message (reveal , 0), then $\tilde{\wp}_C$ sends $(0, x)$ to $\mathcal{A}$, and if $\wp_{\widetilde{\mathcal{F}_{\text{COM}}}}$ gives the message (reveal , 1), then $\tilde{\wp}_C$ sends $(1, y)$ to $\mathcal{A}$.

Now we argue that $\mathcal{Z}$ cannot distinguish between the real execution and the simulation: i.e., $\text{EXEC}_{\text{BCOM},\mathcal{Z},\mathcal{A}} \approx \text{EXEC}^{\langle \widetilde{\mathcal{F}_{\text{COM}}} \rangle, \mathcal{Z}, \mathcal{S}}$ where $\mathcal{S} = \mathbb{comb}(\mathcal{A}, \mathcal{T})$. Suppose otherwise. Then it must be the case that in this Network, with non-negligible probability $\mathcal{A}$ corrupts R but not C (i.e., the last case considered above), because in all other cases the simulation is perfect. Then we shall build non-uniform machines[10] $M_0^{\Psi^*}$ and $M_1^{\Psi^*}$ such that at least one of them provides a counter example to Assumption A1. We describe $M_0^{\Psi^*}$ now. ($M_1^{\Psi^*}$ is built similarly, with 0 and 1 interchanged).

Construction 5.2: $M_0^{\Psi^*}$ to break Assumption A1.

$M_0^{\Psi^*}$ takes as input (x, z) such that either $(x, y, z) \leftarrow \mathcal{D}_r^{\mu}$ or $x \leftarrow \{0,1\}^{k_2}, z = \mathcal{H}(\mu, r, x, 0)$. $M_0^{\Psi^*}$ internally simulates the entire Network corresponding to $\text{EXEC}^{\langle \widetilde{\mathcal{F}_{\text{COM}}} \rangle, \mathcal{Z}, \mathcal{S}}$, with the following modifications:

- If the bit b provided by $\mathcal{Z}$ as input to $\wp_C$ is 1 $M_0^{\Psi^*}$ bails out by outputting a uniformly random bit.
- Otherwise it modifies $\mathcal{T}$ so that instead of using (x, z) obtained from Ψ in the simulation it uses (x, z) obtained as input, and outputs what $\mathcal{Z}$ outputs.

Now, with $\mu = \mu_R$, if (x, z), the input to $M_0^{\Psi^*}$, is such that $(x, y, z) \leftarrow \mathcal{D}_r^{\mu}$ then $M_0^{\Psi^*}$ does a perfect simulation of the Network execution $\text{EXEC}^{\langle \widetilde{\mathcal{F}_{\text{COM}}} \rangle, \mathcal{Z}, \mathcal{S}}$, conditioned on $\mathcal{Z}$ giving 0 as input to $\wp_C$. If on the other hand the input (x, z) is such that $x \leftarrow \{0,1\}^{k_2}, z = \mathcal{H}(\mu, r, x, 0)$ then, conditioned on $\mathcal{Z}$ giving 0 as input to $\wp_C$, this is a perfect simulation of the Network execution $\text{EXEC}_{\text{BCOM},\mathcal{Z},\mathcal{A}}$. Thus[11]

$$
\begin{aligned}
&\mathbf{Pr}_{(x,y,z)\leftarrow\mathcal{D}_r^{\mu}}\left[M_0^{\Psi^*}(x,z)=0\right] - \mathbf{Pr}_{x\leftarrow\{0,1\}^{k_2}, z=\mathcal{H}(\mu,r,x,0)}\left[M_0^{\Psi^*}(x,z)=0\right] \\
&\quad = \mathbf{Pr}_{(x,y,z)\leftarrow\mathcal{D}_r^{\mu}}\left[M_0^{\Psi^*}(x,z)=0 \wedge \mathsf{input}_{\wp_C}=0\right] \\
&\qquad\qquad - \mathbf{Pr}_{x\leftarrow\{0,1\}^{k_2}, z=\mathcal{H}(\mu,r,x,0)}\left[M_0^{\Psi^*}(x,z)=0 \wedge \mathsf{input}_{\wp_C}=0\right] \\
&\quad = \mathbf{Pr}\left[\text{EXEC}^{\langle \widetilde{\mathcal{F}_{\text{COM}}} \rangle, \mathcal{Z}, \mathcal{S}}=0 \wedge \mathsf{input}_{\wp_C}=0\right] \\
&\qquad\qquad - \mathbf{Pr}\left[\text{EXEC}_{\text{BCOM},\mathcal{Z},\mathcal{A}}=0 \wedge \mathsf{input}_{\wp_C}=0\right].
\end{aligned}
$$

Similarly,

$$\begin{aligned}&\mathbf{Pr}_{(x,y,z)\leftarrow\mathcal{D}_r^\mu}\left[M_1^{\Psi^*}(x,z)=0\right]-\mathbf{Pr}_{x\leftarrow\{0,1\}^{k_2},z=\mathcal{H}(\mu,r,x,0)}\left[M_1^{\Psi^*}(x,z)=0\right]\\&\quad=\mathbf{Pr}\left[\text{EXEC}_{\langle\mathcal{F}_{\widetilde{\text{COM}}}\rangle,\mathcal{Z},\mathcal{S}}=0\wedge\text{input}_{\wp_C}=1\right]-\mathbf{Pr}\left[\text{EXEC}_{\text{BCOM},\mathcal{Z},\mathcal{A}}=0\wedge\text{input}_{\wp_C}=1\right].\end{aligned}$$

Hence, if $|\mathbf{Pr}\left[\text{EXEC}_{\langle\mathcal{F}_{\widetilde{\text{COM}}}\rangle,\mathcal{Z},\mathcal{S}}=0\right]-\mathbf{Pr}\left[\text{EXEC}_{\text{BCOM},\mathcal{Z},\mathcal{A}}=0\right]|$ is non-negligible, then the difference $\mathbf{Pr}_{(x,y,z)\leftarrow\mathcal{D}_r^\mu}\left[M_b^{\Psi^*}(x,z)=0\right]-\mathbf{Pr}_{x\leftarrow\{0,1\}^{k_2},z=\mathcal{H}(\mu,r,x,0)}\left[M_b^{\Psi^*}(x,z)=0\right]$ is non-negligible for either $b=0$ or $b=1$.[12] Further note that $M_0^{\Psi^*}$ and $M_1^{\Psi^*}$ do little more than simulate the Network and are hence PPT, with access to Ψ. But this contradicts Assumption A1, leading us to conclude $\text{EXEC}_{\text{BCOM},\mathcal{Z},\mathcal{A}}\approx\text{EXEC}_{\langle\mathcal{F}_{\widetilde{\text{COM}}}\rangle,\mathcal{Z},\mathcal{S}}$. □

A priori the functionality $\mathcal{F}_{\widetilde{\text{COM}}}$ does not offer any guarantee that the commitment is binding on a corrupt sender. The following lemma formulates the binding property outside the our security framework (i.e., we do not give a functionality reflecting the binding property).

Lemma 5.6. *Consider a session of $\langle\mathcal{F}_{\widetilde{\text{COM}}}\rangle$ in a Network with environment $\mathcal{Z}\in\mathcal{Z}_*^\Psi$, multiple other sessions of the same or other protocols (which can all be w.l.o.g considered part of the environment), and adversary $\mathcal{A}\in\mathcal{S}_{\text{static}}^\Psi$. Then, if the receiver R is honest (the sender C may be corrupt), after $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ finishes the commit phase, there is a fixed bit b^* (determined by the entire Network state at that point, excluding as yet unsampled random bits), such that $\wp_C$ can make $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ accept a reveal to $1-b^*$ with only negligible probability.*

PROOF: We define the *value* of the commitment b^* as follows: let α denote the state of the Network at the end of the commitment phase. Define random variable p_0^α (respectively, p_1^α) as the probability that starting from the state α, $\wp_C$ successfully reveals the commitment as 0 (respectively, 1). Let $b^*=0$ if $p_0^\alpha\geq p_1^\alpha$; else let $b^*=1$. (Note that b^* is a function of α.)

We say that the binding is broken if the sender manages to reveal the commitment to $1-b^*$. We shall demonstrate a (non-uniform) PPT machine $M^{\Psi\setminus\mu}$ (where $\mu=\mu_R$) which accepts $r\leftarrow\{0,1\}^k$ and outputs (x,y) such that $\mathcal{H}(\mu,r,x,0)=\mathcal{H}(\mu,r,y,1)$, with a probability polynomially related to the probability of the sender breaking the binding.

Construction 5.3: $M^{\Psi\setminus\mu}$ to output (x,y) such that $\mathcal{H}(\mu,r,x,0)=\mathcal{H}(\mu,r,y,1)$, on input r

$M^{\Psi\setminus\mu}$ simulates the Network internally, starting at the point the session is initiated (which is given to it non-uniformly). It uses access to $\Psi_{\setminus\mu}$ to simulate calls to Ψ (note that $\mu=\mu_R$ and R is honest in this Network; then $\Psi_{\setminus\mu}$ subsumes Ψ). The Network is simulated with the following modifications:

- Recall that in the actual Network $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ chooses a random string $r\leftarrow\{0,1\}^k$ and sends it to $\wp_C$, the (corrupted) sender. But while simulating the Network,

$M^{\Psi\backslash\mu}$ will accept r as an input and send that as the first message to $\wp_C$. (Note that the simulation is perfect if this external input r is chosen randomly from $\{0,1\}^k$.)

- On receiving the message r, $\wp_C$ may respond with a string c. At this point $M^{\Psi\backslash\mu}$ makes two copies of the Network, and runs them with independent randomness.
- If $\wp_C$ eventually reveals the commitment as $(x,0)$ in the first run and as $(y,1)$ in the second run, then $M^{\Psi\backslash\mu}$ outputs (x,y) and *succeeds*. Else it fails and terminates.

Then we have,

$$\mathbf{Pr}\left[M^{\Psi\backslash\mu} \text{ succeeds}\right] = \mathop{\mathbf{E}}_{\alpha}\left[p_0^\alpha p_1^\alpha\right]$$

because after forking two copies of the Network, $M^{\Psi\backslash\mu}$ succeeds (i.e., it manages to output (x,y) such that $\mathcal{H}(r,x,0) = \mathcal{H}(r,y,1)$) when in the first run the event with probability p_0^α occurs and in the second the event with probability p_1^α. Here $\mathbf{E}_\alpha\left[f(\alpha)\right]$ stands for $\sum_\alpha \mathbf{Pr}\left[\alpha\right] f(\alpha)$. Note that the randomness involved in determining $\mathbf{Pr}\left[\alpha\right]$ includes all the randomness used by $M^{\Psi\backslash\mu}$ to simulate the Network up to the point α, and the input r that it receives (equivalently, it involves all the randomness in the actual Network). On the other hand,

$$\begin{aligned}\mathbf{Pr}\left[\wp_C \text{ reveals } 1-b^*\right] &= \mathop{\mathbf{E}}_{\alpha}\left[\min\{p_0^\alpha, p_1^\alpha\}\right] \leq \mathop{\mathbf{E}}_{\alpha}\left[\sqrt{p_0^\alpha p_1^\alpha}\right] \\ &\leq \sqrt{\mathop{\mathbf{E}}_{\alpha}\left[p_0^\alpha p_1^\alpha\right]} = \sqrt{\mathbf{Pr}\left[M^{\Psi\backslash\mu} \text{ succeeds}\right]}.\end{aligned} \tag{5.1}$$

Since the running time of $M^{\Psi\backslash\mu}$ is linearly related to the running time of the entire Network, $M^{\Psi\backslash\mu}$ is a PPT machine, with access to $\Psi_{\backslash\mu}$. Hence the probability that $M^{\Psi\backslash\mu}$ succeeds is negligible by Assumption A2. Hence, by (5.1), the probability that $\wp_C$ reveals $1-b^*$ (and makes $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ accept it) is also negligible. □

5.5.2 Multi-bit Commitment with Selective Reveal.

We define a multi-bit version of the functionality $\mathcal{F}_{\widetilde{\text{COM}}}$, called $\mathcal{F}_{\widetilde{\text{MCOM}}}$ given in Figure 5.3. Informally, $\mathcal{F}_{\widetilde{\text{MCOM}}}$ is equivalent to n parallel sessions of $\mathcal{F}_{\widetilde{\text{COM}}}$ (where n is specified by the sender $\wp_C$). $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ internally runs n copies of $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ and interacts with the sender $\wp_C$ according to them. In the commit phase, it sends commit if all n parallel copies of $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ output COMMIT (as the message for $\wp_R$). In the reveal phase, the sender can choose to run the reveal phase of any subset $\{i_1,\ldots,i_t\} \subseteq [n]$ of parallel sessions. Then if in reveal phase of each of the t chosen sessions, $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ of that session outputs the reveal message (reveal , b_j)

(intended for the receiver), then $\wp_{\widetilde{\mathcal{F}_{\text{MCOM}}}}$ sends the message $(\mathsf{reveal}, (i_1, b_1), \ldots, (i_t, b_t))$ to $\wp_R$.

When a protocol is started from within another protocol, the latter should specify how the session ID of the sub-protocol is generated and agreed upon. We make explicit our conventions regarding how this is done, by specifying the details for the functionality $\mathcal{F}_{\widetilde{\text{MCOM}}}$ and a protocol BMCOM for it, in Figure 5.3 and Figure 5.4. Similar schemes can be employed for all other protocols in this work.

Functionality $\mathcal{F}_{\widetilde{\text{MCOM}}}$ and Protocol $\langle \mathcal{F}_{\widetilde{\text{MCOM}}} \rangle$

A sender $\wp_C$ and a receiver $\wp_R$ run the protocol $\langle \mathcal{F}_{\widetilde{\text{MCOM}}} \rangle$ to interact with $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$. The adversary is $\mathcal{S}$. Let the session ID (already agreed up on by $\wp_C$ and $\wp_R$, on instantiating this copy of $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$) be sid. Let the input to $\wp_C$ at the commit phase be $(b_1, \ldots, b_n)$, the bits to which it wishes to commit, and that at the reveal phase be $\{i_1, \ldots, i_t\} \subseteq [n]$, the set of commitments which it wishes to reveal.

COMMIT PHASE:

1. $\wp_C$ sends the number n to $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$.
2. $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ internally starts n copies of the program for $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ namely $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}, sid_1}, \ldots, \wp_{\mathcal{F}_{\widetilde{\text{COM}}}, sid_n}$, with $\wp_C$ as the sender and using the ID of R as the receiver-ID. (sid_i could simply be $sid \circ i$.)
3. $\wp_C$ interacts with the t parallel sessions of $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ in the commit phase.
4. If all the n copies of $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ produce the commit message, then $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ sends the message commit to $\wp_R$ and $\mathcal{S}$.

REVEAL PHASE:

1. $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ receives the set $\{i_1, \ldots, i_t\}$ from $\wp_C$.
2. $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ lets the t parallel sessions $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}, sid_{i_1}}, \ldots, \wp_{\mathcal{F}_{\widetilde{\text{COM}}}, sid_{i_t}}$ interact with $\wp_C$ in the reveal phase.
3. On receiving message (reveal, b_i) from the session of $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ with session-ID (sid_i), for all $i \in \{i_1, \ldots, i_t\}$, $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ sends $(\mathsf{reveal}, (i_1, b_{i_1}), \ldots, (i_t, b_{i_t}))$ to $\wp_R$ and $\mathcal{S}$.

Figure 5.3: Multi-bit Commitment With Selective Reveal Functionality.

Then it is easy to show the following lemmas.

Lemma 5.7. *In a setting as in Lemma 5.6, after finishing the commit phase with $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$, there is a fixed string $\overline{b^*} \in \{0,1\}^n$ (where n is the number of bits as specified by $\wp_C$ at the beginning of the protocol) such that $\wp_C$ can make $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ accept a (selective) reveal inconsistent with $\overline{b^*}$ with only negligible probability.*

PROOF: Let $\overline{b^*} = (b_1^*, \ldots, b_n^*)$, where b_i^* is the bit guaranteed by Lemma 5.6 for $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}, sid_i}$. Then, if $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ accepts a selective reveal inconsistent with $\overline{b^*}$, then for some i $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}, sid_i}$

Protocol BMCOM

The parties are sender C and receiver R. Let the session ID (already agreed up on by $\wp_C$ and $\wp_R$, on instantiating this copy of BMCOM) be sid. Let the input to $\wp_C$ at the commit phase be $(b_1, \ldots, b_n)$, the bits to which it wishes to commit, and that at the reveal phase be $\{i_1, \ldots, i_t\} \subseteq [n]$, the set of commitments which it wishes to reveal.

COMMIT PHASE:

1. $\wp_C$ sends the number n to $\wp_R$.
2. $\wp_C$ and $\wp_R$ initiate n parallel sessions of $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ with session-IDs $(sid_1), \ldots, (sid_n)$.
3. $\wp_C$ interacts (in parallel) with the n sessions of $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ to commit to the bits $b_1, \ldots, b_n$.
4. On receiving commit message from all the n sessions, $\wp_R$ accepts the commitment.

REVEAL PHASE:

1. $\wp_C$ sends the set $\{i_1, \ldots, i_t\}$ to $\wp_R$.
2. $\wp_C$ interacts in parallel with the sessions of $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ with session-IDs $(sid_{i_1}), \ldots, (sid_{i_t})$ to reveal to the bits $b_{i_1}, \ldots, b_{i_t}$.
3. On receiving message (reveal, b_i) from the session with session-ID (sid_i), for all $i \in \{i_1, \ldots, i_t\}$, $\wp_R$ accepts $(i_1, b_{i_1}), \ldots, (i_t, b_{i_t})$ as the revealed information.

Figure 5.4: Protocol BMCOM (which uses $\mathcal{F}_{\widetilde{\text{COM}}}$).

must accept $b \neq b_i^*$. By Lemma 5.6 and a the union bound, the probability of this happening is negligible in k, because n is polynomial in k. □

Lemma 5.8. *Protocol* BMCOM *is as secure as* $\langle \mathcal{F}_{\widetilde{\text{MCOM}}} \rangle$ *with respect to* $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\mathsf{static}}, \mathcal{S}^\Psi_{\mathsf{static}})$.

We omit a detailed description of the transvisor $\mathcal{T}^{\langle \mathcal{F}_{\widetilde{\text{MCOM}}} \rangle \to \text{BMCOM}}$, as perfect simulation is easily achieved. We also remark that $\mathcal{F}_{\widetilde{\text{MCOM}}}$ could be considered as a multi-instance join of $\mathcal{F}_{\widetilde{\text{COM}}}$.

5.5.3 Basic Zero Knowledge Proof

Consider a proto-typical 3-round Zero Knowledge Proof protocol (a Σ-protocol) for proving membership in an NP-complete language (like 3-colorability or Hamiltonicity), in which the prover uses the basic commitment functionality $\mathcal{F}_{\widetilde{\text{COM}}}$ from above, to carry out the commitments (first round) and the reveals (last round). Let us denote this protocol by BZK. Then, like we defined $\mathcal{F}_{\widetilde{\text{COM}}}$ from BCOM, we can define a basic Zero Knowledge Proof functionality $\mathcal{F}_{\widetilde{\text{ZK}}}$ from BZK. The description of the functionality is simple: $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ interacts with the prover according to the protocol BZK, playing the verifiers role. If the prover completes the proof according to the protocol, $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ sends a message proven to the verifier.

Figure 5.5 gives the protocol BZK, and Figure 5.6 gives the functionality $\mathcal{F}_{\widetilde{\text{ZK}}}$ (and the corresponding ideal protocol $\langle \mathcal{F}_{\widetilde{\text{ZK}}} \rangle$), for proving Hamiltonicity. Note that the protocol

BZK uses $\langle \mathcal{F}_{\widetilde{\text{MCOM}}} \rangle$ as a subroutine. Correspondingly, the functionality program $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ runs $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ as a subroutine.

Protocol BZK

The participants are prover P and verifier V. The common input is a graph $G([n], E)$. In addition, the prover $\wp_P$ gets a witness WITNESS which is a Hamiltonian cycle in G. The size of the graph n is polynomial in the security parameter k. t is a parameter which grows as $\omega(\log k)$ (say $t(k) = \log^2 k$).
$\wp_P$ verifies that WITNESS is a valid Hamiltonian cycle of G. (Else it aborts the protocol.)

1. Repeat the following t times in parallel:
   (a) $\wp_P$ picks a permutation ϕ of $[n]$, uniformly at random. Let $\phi(G)$ be the graph isometric to G obtained by permuting the labels on the vertices according to ϕ. Let M_1 be the adjacency matrix of $\phi(G)$. Let M_2 be a bit representation of ϕ (say as an $n \times n$ matrix).
   (b) $\wp_P$ interacts with $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ to commit (M_1, M_2) to $\wp_V$.
   (c) $\wp_V$ picks a random bit $b \leftarrow \{0, 1\}$ and sends it to $\wp_P$.
   (d) If $b = 0$ then $\wp_P$ interacts with $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ to reveal (M_1, M_2) to $\wp_V$. If $b = 1$ then $\wp_P$ interacts with $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ to reveal $M_1|_\zeta$ to $\wp_V$, where ζ corresponds to the edges of the cycle ϕ(WITNESS). $\wp_V$ receives the reveal messages from $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$.
2. $\wp_V$ checks if in all $n^2 t$ parallel repetitions:

$$b = 0 \implies M_2 \text{ represents a permutation } \phi, \text{ and } M_1 \text{ represents a graph such that } \phi(G) = M_1$$
$$b = 1 \implies \zeta \text{ corresponds to the edges of a Hamiltonian cycle, and } \forall (i,j) \in \zeta, M_{1ij} = 1.$$

   If so $\wp_V$ accepts the proof and outputs proven .

Figure 5.5: Basic ZK Proof Protocol BZK (which uses $\mathcal{F}_{\widetilde{\text{MCOM}}}$).

Lemma 5.9. *Protocol* BZK *is as secure as* $\mathcal{F}_{\widetilde{\text{ZK}}}$ *with respect to* $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\text{static}}, \mathcal{S}^\Psi_{\text{static}})$.

PROOF: For every adversary $\mathcal{A} \in \mathcal{A}^\Psi_{\text{static}}$ we demonstrate a PPT transvisor $\mathcal{T} = \mathcal{T}^{\langle \mathcal{F}_{\widetilde{\text{ZK}}} \rangle \to \text{BZK}}$ such that no environment $\mathcal{Z} \in \mathcal{Z}^\Psi$ can distinguish between interacting with the parties running BZK and with $\mathcal{A}$ (the "real" world), and interacting with the parties running $\langle \mathcal{F}_{\widetilde{\text{ZK}}} \rangle$ and with $\mathcal{S} = \mathtt{comb}(\mathcal{A}, \mathcal{T})$ (the "ideal" world).

Construction 5.4: Transvisor $\mathcal{T} = \mathcal{T}^{\langle \mathcal{F}_{\widetilde{\text{ZK}}} \rangle \to \text{BZK}}$

It behaves as follows depending on which parties are corrupted.
Both P, V corrupt: If $\mathcal{A}$ corrupts both participants, $\mathcal{T}$ acts transparently. Then the simulation is trivially perfect.

Functionality $\mathcal{F}_{\widetilde{\text{ZK}}}$ and Protocol $\langle \mathcal{F}_{\widetilde{\text{ZK}}} \rangle$

A prover $\wp_P$ and a verifier $\wp_V$ run the protocol $\langle \mathcal{F}_{\widetilde{\text{ZK}}} \rangle$ to interact with $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$. The adversary is $\mathcal{S}$. The common input (to $\wp_P$, $\wp_V$ and $\mathcal{S}$) is a graph $G([n], E)$. The size of the graph n is polynomial in the security parameter k. t is a parameter which grows as $\omega(\log k)$ (say $t(k) = \log^2 k$).
$\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ a simulates a verifier $\tilde{\wp}_V$ and lets a subroutine $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ interact with $\wp_P$ and $\tilde{\wp}_V$, below.

1. Repeat the following n^2t times in parallel:
   (a) $\wp_P$ picks a permutation ϕ of $[n]$, uniformly at random. Let $\phi(G)$ be the graph isometric to G obtained by permuting the labels on the vertices according to ϕ. Let M_1 be the adjacency matrix of $\phi(G)$. Let M_2 be a bit representation of ϕ (say as an $n \times n$ matrix).
   (b) $\wp_P$ interacts with the subroutine $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ to commit (M_1, M_2).
   (c) $\tilde{\wp}_V$ picks a random bit $b \leftarrow \{0, 1\}$ and sends it to $\wp_P$.
   (d) If $b = 0$ then $\wp_P$ interacts with $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ to reveal (M_1, M_2). If $b = 1$ then $\wp_P$ interacts with $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ to reveal $M_1|_\zeta$, where ζ corresponds to the edges of the cycle ϕ(WITNESS). $\tilde{\wp}_V$ receives the reveal messages from $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$.
2. $\tilde{\wp}_V$ checks if in all n^2t parallel repetitions:

$$b = 0 \implies M_2 \text{ represents a permutation } \phi, \text{ and } M_1 \text{ represents a graph such that } \phi(G) = M_1$$
$$b = 1 \implies \zeta \text{ corresponds to the edges of a Hamiltonian cycle, and } \forall (i,j) \in \zeta, M_{1ij} = 1.$$

If so $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ sends proven to $\wp_V$ and $\mathcal{S}$.

Figure 5.6: A functionality $\mathcal{F}_{\widetilde{\text{ZK}}}$ realized by the BZK protocol

Both P, V honest: In this case $\mathcal{T}$ internally runs a simulation of $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ (denoted by $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$), and of the honest BZK program for V (denoted by $\tilde{\wp}_V$).[13] Note that in the protocol BZK the only message sent by $\wp_V$ are the random bits b, where as messages sent by $\wp_P$ are all to the functionality $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ (whose contents are hidden from $\mathcal{A}$). Apart from this all that $\mathcal{A}$ sees are the commit and reveal messages sent out by $\wp_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$, the latter, depending on the values b sent by $\tilde{\wp}_V$: in each of the parallel repetitions, if $b = 0$, $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ picks a random permutation ϕ, sets M_1 and M_2 as prescribed by the protocol (which does not require the knowledge of WITNESS) and sends them in a reveal message; if $b = 1$, then $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{MCOM}}}}$ picks a random Hamiltonian cycle of the complete graphs and sends a reveal message for all bits in the first matrix along that Hamiltonian cycle, revealing them as one.

to either a random permutation and the graph permuted according to it, or to all bits being one along a random Hamiltonian cycle in the complete graph. Clearly

$\mathcal{T}$ (i.e., $\tilde{\wp}_{\widetilde{\mathcal{F}_{\text{MCOM}}}}$ and $\tilde{\wp}_V$) can perfectly simulate all these messages.
V honest, P corrupted: In this case $\mathcal{T}$ acts transparently: it will simply forward the messages from corrupt $\wp_P$ to $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ and reports back to $\wp_P$ the messages from $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$. This is a perfect simulation because when V is honest, the only difference between running the protocol BZK and $\langle \mathcal{F}_{\widetilde{\text{ZK}}} \rangle$ is that in the latter the BZK program for V is internally run by $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$.
P honest, V corrupted: Finally, we consider the case when the adversary corrupts the verifier alone. Recall that when P is honest, $\mathcal{A}$ does not see any messages from P, but only the messages it receives from $\wp_{\widetilde{\mathcal{F}_{\text{MCOM}}}}$. These messages are simulated by $\tilde{\wp}_{\widetilde{\mathcal{F}_{\text{MCOM}}}}$: it sends commit messages to the corrupted $\wp_V$. Then if $\wp_V$ responds with b, $\wp_{\widetilde{\mathcal{F}_{\text{MCOM}}}}$ carries out the simulation exactly as in the case when both P and V are honest: i.e., sending a reveal message for (M_1, M_2) or to all ones along a random Hamiltonian cycle of the complete graph. Clearly this is a perfect simulation of an interaction with the honest prover $\wp_P$ and $\wp_{\widetilde{\mathcal{F}_{\text{MCOM}}}}$.

Since the simulation is perfect in all four cases, we have $\text{EXEC}_{\text{BZK},\mathcal{Z},\mathcal{A}} = \text{EXEC}^{\langle \mathcal{F}_{\widetilde{\text{ZK}}} \rangle,\mathcal{Z},\mathcal{S}}$, where $\mathcal{S} = \mathtt{comb}(\mathcal{A}, \mathcal{T})$. □

The functionality $\mathcal{F}_{\widetilde{\text{ZK}}}$ does not make any guarantees of soundness, *a priori*. But as with $\mathcal{F}_{\widetilde{\text{COM}}}$, we shall demonstrate this property outside the Network-aware security framework.

Lemma 5.10. *Consider a session of $\langle \mathcal{F}_{\widetilde{\text{ZK}}} \rangle$ in a Network with environment $\mathcal{Z} \in \mathcal{Z}_*^{\Psi}$, multiple other sessions of the same or other protocols (which can all be w.l.o.g considered part of the environment), and adversary $\mathcal{A} \in \mathcal{S}_{\mathsf{static}}^{\Psi}$. Then, if the verifier V is honest (the prover P may be corrupt), then $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ accepts the proof to a false statement with negligible probability.*

PROOF: In each of the $n^2\omega(\log k)$ repetitions of the commitment in the interaction of $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ with $\wp_P$, consider the point at which $\wp_{\widetilde{\mathcal{F}_{\text{MCOM}}}}$ sends the commit message to $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$. Then, by the binding property of $\wp_{\widetilde{\mathcal{F}_{\text{MCOM}}}}$ (Lemma 5.7), we know that there exist values $(b_1^*, \ldots, b_n^*)$ such that the probability that $\wp_{\widetilde{\mathcal{F}_{\text{MCOM}}}}$ will send a reveal message with (i, b_i) for $b_i \neq b_i^*$ is negligible.

For convenience, we define the following events: unsound is the event that $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ accepts the proof of an incorrect statement; badcom is the event that $\wp_{\widetilde{\mathcal{F}_{\text{MCOM}}}}$ will send a reveal message with (i, b_i) where $b_i \neq b_i^*$; ALLGOODQUERIES is the event that in each of the n sessions, for the bit selected by V the bits b_i^* define a valid answer (i.e., for the challenge bit $b = 0$ the bits b_i^* define $M_1 = \phi(G), M_2 = \phi$ and for challenge $b = 1$ they define a matrix M_1 with Hamiltonian cycle. If any pair (M_1, M_2) defined by b_i^* define a valid answer for both $b = 0$ and $b = 1$, it implies that the graph is Hamiltonian; else we call the pair (M_1, M_2) "bad." Let allbadpairs be the event that in all the n sessions, bits

recorded by the commitment monitors give bad pairs of matrices. Then

$$\begin{aligned}
\mathbf{Pr}\,[\mathsf{unsound}] &\leq \mathbf{Pr}\left[\mathsf{allbadpairs} \wedge \wp_{\mathcal{F}_{\widetilde{\mathrm{ZK}}}} \text{ sends } \mathsf{proven}\right] \\
&\leq \mathbf{Pr}\,[\mathsf{allbadpairs} \wedge (\,\mathsf{allgoodqueries} \vee \mathsf{badcom})] \\
&\leq \mathbf{Pr}\,[(\mathsf{allbadpairs} \wedge \mathsf{allgoodqueries}) \vee (\mathsf{allbadpairs} \wedge \mathsf{badcom})] \\
&\leq \mathbf{Pr}\,[(\mathsf{allbadpairs} \wedge \mathsf{allgoodqueries}) \vee \mathsf{badcom}] \\
&\leq \mathbf{Pr}\,[\mathsf{allbadpairs} \wedge \mathsf{allgoodqueries}] + \mathbf{Pr}\,[\mathsf{badcom}] \\
&\leq \mathbf{Pr}\,[\,\mathsf{allgoodqueries}|\mathsf{allbadpairs}] + \mathbf{Pr}\,[\mathsf{badcom}].
\end{aligned}$$

If a pair is bad it can define a valid answer for at most one of the two possible queries. That is, with probability at most $\frac{1}{2}$, $\wp_V$ makes a good query on that pair. So,

$$\mathbf{Pr}\,[\,\mathsf{allgoodqueries}|\mathsf{allbadpairs}] \leq 2^{-t}.$$

As $t = \omega(k)$ this is negligible in k. Since $\mathbf{Pr}\,[\mathsf{badcom}]$ is also negligible, we conclude that $\mathbf{Pr}\,[\mathsf{unsound}]$ is negligible. □

5.6 Commitment Functionality

The basic protocols and non-standard functionalities given in the previous section now allow us to securely realize the "fully" ideal commitment functionality $\mathcal{F}_{\text{COM}}$ shown in Figure 5.7. Since for the sake of simplicity in describing our protocols we allowed the session IDs to be implicit, we do the same in specifying the functionality.

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

The parties are a sender C and a receiver R, with adversary $\mathcal{S}$. The input to $\wp_C$ is a bit b.

COMMIT PHASE:

$\wp_C \rightarrow \wp_{\mathcal{F}_{\text{COM}}}$: b

$\wp_{\mathcal{F}_{\text{COM}}} \rightarrow \wp_R$: commit

REVEAL PHASE:

$\wp_C \rightarrow \wp_{\mathcal{F}_{\text{COM}}}$: reveal

$\wp_{\mathcal{F}_{\text{COM}}} \rightarrow \wp_R, \mathcal{S}$: (reveal, b)

Figure 5.7: The Commitment Functionality

Let T stand for a collection of trapdoor-permutations, B a hardcore predicate for this collection and C a perfectly binding commitment scheme, as described in Section 5.3.3. Recall that these primitives are assumed to be secure against adversaries with access to Ψ^*. The protocol BCOM in Figure 5.8 uses these primitives. It is based on the "commit-with-extract" protocol from [BL02].

Lemma 5.11. *Protocol* COM *is as secure as* $\mathcal{F}_{\text{COM}}$ *with respect to* $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$.

Protocol COM

The parties are a sender C and a receiver R; adversary is $\mathcal{S}$. The input to $\wp_C$ is a bit b. k is the security parameter.

COMMIT PHASE:

C1 $\wp_R$ draws $r_R \leftarrow \{0,1\}^k$ and sends $c = \mathsf{C}(r_R; r)$ where r is also drawn at random.

C2 $\wp_C$ draws $(f, f^{-1}) \leftarrow \mathsf{T}(1^k)$ and sends f to $\wp_R$. $\wp_C$ interacts with $\wp_{\mathcal{F}_{\widetilde{\mathrm{ZK}}}}$ to prove to $\wp_R$ that $(\exists r', g : (f, g) = \mathsf{T}(1^k; r'))$. $\wp_R$ receives the message proven from $\wp_{\mathcal{F}_{\widetilde{\mathrm{ZK}}}}$.

C3 $\wp_C$ draws $r_C \leftarrow \{0,1\}^k$ and sends it to $\wp_R$.

C4 $\wp_R$ sends r_R to $\wp_C$.

C5 $\wp_R$ interacts with $\wp_{\mathcal{F}_{\widetilde{\mathrm{ZK}}}}$ to prove to $\wp_C$ that $(\exists r' : c = \mathsf{C}(r_R; r'))$ $\wp_C$ receives the message proven from $\wp_{\mathcal{F}_{\widetilde{\mathrm{ZK}}}}$.

C6 Let b be the bit $\wp_C$ wants to commit to. $\wp_C$ computes $b' = \mathsf{B}(f^{-1}(r_R \oplus r_C)) \oplus b$ and sends b' to $\wp_R$. $\wp_R$ accepts the commitment.

REVEAL PHASE:

R1 $\wp_C$ sends the bit b to $\wp_R$.

R2 $\wp_C$ interacts with $\wp_{\mathcal{F}_{\widetilde{\mathrm{ZK}}}}$ to prove to $\wp_R$ that $(\exists t : f(t) = r_R \oplus r_C \bigwedge b' = \mathsf{B}(t) \oplus b)$.

R3 Up on receiving the message proven from $\wp_{\mathcal{F}_{\widetilde{\mathrm{ZK}}}}$, $\wp_R$ accepts b as the revealed bit.

Figure 5.8: Protocol COM

First we present an overview of the proof, before giving the formal arguments.

PROOF OVERVIEW: We need to construct a PPT transvisor $\mathcal{T} = \mathcal{T}^{\langle \mathcal{F}_{\mathrm{COM}} \rangle \to \mathrm{COM}}$. As usual when both C and R are corrupt $\mathcal{T}$ acts transparently. When both C and R are honest, $\mathcal{T}$ simulates the protocol exactly until the step where the bit b is used (Step C6). At this step, it simulates the sender sending out a random bit as b'. In the reveal phase $\mathcal{T}$ can easily simulate a proof from $\wp_{\mathcal{F}_{\widetilde{\mathrm{ZK}}}}$, to open it either way. The hiding property of the hard-core bit B can be used to show that the entire simulation is indistinguishable from an actual execution. When R is corrupt and C is honest, the same simulator works, for the same reasons. However the reduction to the security of B becomes slightly more involved in this case.

When R is honest and C corrupt, $\mathcal{T}$ should be able to *extract* the committed bit. The idea here is that $\mathcal{T}$ (playing the part of $\wp_R$ in the protocol) will cheat in the proof using the simulated $\wp_{\mathcal{F}_{\widetilde{\mathrm{ZK}}}}$ in Step C5 (reveal phase of the coin-flipping part), and have $r_R \oplus r_C$ match a random string r such that it knows $\mathsf{B}(r_R \oplus r_C)$. This will allow it to extract the bit b. Soundness of $\mathcal{F}_{\widetilde{\mathrm{ZK}}}$ (Lemma 5.10) ensures that $\wp_C$ cannot feasibly open to a bit other than b. Also it ensures that f is a permutation. Then the hiding property of the commitment C ensures that the simulation is indistinguishable from an actual execution. $\triangleleft$

PROOF: For every adversary $\mathcal{A} \in \mathcal{A}^{\Gamma}_{\mathsf{static}}$ we demonstrate a PPT transvisor $\mathcal{T}$ such that no environment $\mathcal{Z} \in \mathcal{Z}^{\Gamma}$ can distinguish between interacting with the parties running COM and with $\mathcal{A}$ (the "real" world), and interacting with the parties running $\langle \mathcal{F}_{\text{COM}} \rangle$ and with $\mathcal{S} = \mathtt{comb}(\mathcal{A}, \mathcal{T})$ (the "ideal" world). First we present the construction for $\mathcal{T} = \mathcal{T}^{\langle \mathcal{F}_{\text{COM}} \rangle \to \text{COM}}$ and then argue the indistinguishability.

Construction 5.5: Transvisor $\mathcal{T} = \mathcal{T}^{\langle \mathcal{F}_{\text{COM}} \rangle \to \text{COM}}$

$\mathcal{T}$ simulates the honest participants internally, simulating one or both of $\wp_C$ (if C is honest) and $\wp_R$ (if R is honest) as required. Also it internally simulates the other functionality used in the protocol $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$. We denote the simulated programs by $\tilde{\wp}_C$, $\tilde{\wp}_R$ and $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ respectively. $\mathcal{T}$ behaves as follows depending on which parties are corrupted.

Both C, R corrupt. If $\mathcal{A}$ corrupts both participants, $\mathcal{T}$ acts transparently. Then the simulation is trivially perfect.

Both C, R honest.[14] $\mathcal{T}$ internally runs simulations $\tilde{\wp}_C$, $\tilde{\wp}_R$ and $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$. The steps in the protocol are simulated as follows:

- Recall that $\mathcal{T}$ does not have access to the input bit b that $\wp_C$ receives from $\mathcal{Z}$. But in the commit phase, until the last message, b is not used. So $\tilde{\wp}_C$ and $\tilde{\wp}_R$, can follow the protocol exactly until last step of the commitment phase, Step C6.
- In Step C6, $\tilde{\wp}_C$ sends out a random bit as b'.
- In the reveal phase, on receiving (reveal, b) from $\wp_{\mathcal{F}_{\text{COM}}}$, $\tilde{\wp}_C$ sends b to $\tilde{\wp}_R$ to simulate Step R1.
- Then it must simulate the interaction between $\wp_C$, $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ and $\wp_R$ with the statement $\exists t : f(t) = r_R \oplus r_C \wedge b' = \mathsf{B}(t) \oplus b$ as common input. Note that in this interaction all that $\mathcal{A}$ sees is the message proven from $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$. So $\mathcal{T}$ simulates this step by having $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ send that message to $\mathcal{A}$.

R honest, C corrupted. In this case the idea is that $\mathcal{T}$ can *extract* the bit being committed to by the corrupted sender $\wp_C$. Here $\mathcal{T}$ runs the simulations $\tilde{\wp}_R$ and $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$.

- $\tilde{\wp}_R$ starts off following the programs of the honest receiver and $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ is also simulated faithfully initially. But after receiving r_C in Step C3, instead of sending r_R, the value to which c is a commitment, $\tilde{\wp}_R$ picks a random string $u \leftarrow \{0,1\}^k$ and sends $\tilde{r}_R = f(u) \oplus r_C$.

- Then $\mathcal{T}$ simulates an interaction with $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$, with the statement $(\exists r' : c = \mathsf{C}(\tilde{r}_R; r'))$ as common input: for this $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ simply sends the message proven to $\wp_C$.
- When $\wp_C$ sends back b', $\mathcal{T}$ calculates $b^* = b' \oplus \mathsf{B}(u)$ (this is the *extracted* commitment). $\mathcal{T}$ then sends b^* to $\wp_{\mathcal{F}_{\text{COM}}}$ to commit that bit to $\wp_{\langle \mathcal{F}_{\text{COM}} \rangle, R}$.
- Later, if $\wp_C$ sends b as revealed, then $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ interacts with $\wp_C$ as it tries to prove that $(\exists t : f(t) = \tilde{r}_R \oplus r_C \wedge b' = \mathsf{B}(t) \oplus b)$. If $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ accepts the proof, then $\mathcal{T}$ will make $\wp_{\langle \mathcal{F}_{\text{COM}} \rangle, R}$ accept b too. For this $\mathcal{T}$ sends reveal instruction to $\wp_{\mathcal{F}_{\text{COM}}}$, which will send (reveal, b^*) to $\wp_{\langle \mathcal{F}_{\text{COM}} \rangle, R}$ (in which case, as we shall see, $b^* = b$ with high probability).

C honest, R corrupted. In this case the transvisor behaves the same way as in the case when both $\wp_C$ and $\wp_R$ are honest.

Now we argue that the above simulation is good: i.e.,

$$|\mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}}\right] - \mathbf{Pr}\left[\text{EXEC}_{\langle \mathcal{F}_{\text{COM}} \rangle,\mathcal{Z},\mathcal{S}}\right]| \tag{5.2}$$

is negligible, where $\mathcal{S} = \mathtt{comb}(\mathcal{A}, \mathcal{T})$. Suppose otherwise. Then at least one of the following is non-negligible.[15]

$$|\mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} = 0 \wedge \mathfrak{X} = \{C, R\}\right] - \mathbf{Pr}\left[\text{EXEC}_{\langle \mathcal{F}_{\text{COM}} \rangle,\mathcal{Z},\mathcal{S}} = 0 \wedge \mathfrak{X} = \{C, R\}\right]| \tag{5.3}$$

$$|\mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} = 0 \wedge \mathfrak{X} = \{\}\right] - \mathbf{Pr}\left[\text{EXEC}_{\langle \mathcal{F}_{\text{COM}} \rangle,\mathcal{Z},\mathcal{S}} = 0 \wedge \mathfrak{X} = \{\}\right]| \tag{5.4}$$

$$|\mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} = 0 \wedge \mathfrak{X} = \{C\}\right] - \mathbf{Pr}\left[\text{EXEC}_{\langle \mathcal{F}_{\text{COM}} \rangle,\mathcal{Z},\mathcal{S}} = 0 \wedge \mathfrak{X} = \{C\}\right]| \tag{5.5}$$

$$|\mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} = 0 \wedge \mathfrak{X} = \{R\}\right] - \mathbf{Pr}\left[\text{EXEC}_{\langle \mathcal{F}_{\text{COM}} \rangle,\mathcal{Z},\mathcal{S}} = 0 \wedge \mathfrak{X} = \{R\}\right]| \tag{5.6}$$

The first of these four expressions is in fact equal to 0, because when both C and R are corrupted and $\mathcal{T}$ acts transparently, the simulation is perfect. Below we rule out each of the other three possibilities.

Both C, R honest. Note that the only difference between the executions $\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}}$ and $\text{EXEC}_{\langle \mathcal{F}_{\text{COM}} \rangle,\mathcal{Z},\mathcal{S}}$, is that $\mathsf{B}(f^{-1}(r_R \oplus r_C)) \oplus b$ may not be equal to b' in the latter (where as it is always the case in the former). So if the environment $\mathcal{Z}$ can distinguish between the two executions, it can be used to build a distinguisher M^{Ψ^*} to distinguish between a random bit and $\mathsf{B}(f^{-1}(r))$, for a randomly chosen f and r with non-negligible probability, as follows.

Construction 5.6: M^{Ψ^*} to distinguish between $h \leftarrow \{0,1\}$ and $h = \mathsf{B}(f^{-1}(r))$

The distinguisher M^{Ψ^*} takes (f, r, h) as input where f is randomly drawn from

$\mathsf{T}(1^k)$, $r \leftarrow \{0,1\}^k$ and h is either a random bit (Experiment 1) or $h = \mathsf{B}(f^{-1}(r))$ (Experiment 2). It tries to distinguish between the two experiments as follows. On receiving the inputs (f, r, h) it simply simulates the entire Network execution corresponding to $\text{EXEC}_{\langle \mathcal{F}_{\text{COM}} \rangle, \mathcal{Z}, \mathcal{S}}$. If either party is corrupted by $\mathcal{A}$, then M^{Ψ^*} outputs a uniformly random bit (thereby bailing out). Otherwise, it proceeds with simulation, modifying it as follows:

- At Step C2, $\tilde{\wp}_C$ sends f (received as input) to $\tilde{\wp}_R$ (instead of picking $(f, f^{-1}) \leftarrow \mathsf{T}(1^k)$).
- At Step C2, $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ sends the message proven to $\wp_R$, without $\tilde{\wp}_C$ having to interact with it.
- At Step C3, $\tilde{\wp}_C$ sends $r_C = r \oplus r_R$ instead of picking a random r_C.
- At Step C6, $\tilde{\wp}_C$ sends $b' = h \oplus b$, instead of sending a random bit.

Finally M^{Ψ^*} outputs the bit output by $\mathcal{Z}$.

First we observe that in Experiment 1 (i.e., the inputs f and r are chosen randomly, and h is a random bit), the output of M^{Ψ^*} is identical to $\text{EXEC}_{\langle \mathcal{F}_{\text{COM}} \rangle, \mathcal{Z}, \mathcal{S}}$ (both conditioned on $\mathfrak{X} = \{\}$). This is true because, when f, r and h are randomly picked, the modifications above do not make any difference at all. Also, clearly the modification in interaction with $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ has no effect in the output, because in $\text{EXEC}_{\langle \mathcal{F}_{\text{COM}} \rangle, \mathcal{Z}, \mathcal{S}}$, $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ always accepts the proof as $\tilde{\wp}_C$ follows the protocol for an honest party.

Next, we claim that in Experiment 2 (i.e., the inputs f and r are chosen randomly, and h is the hardcore bit $\mathsf{B}(f^{-1}(r))$) the output of M^{Ψ^*} is identical to $\text{EXEC}_{\text{BCOM}, \mathcal{Z}, \mathcal{A}}$ (both conditioned on $\mathfrak{X} = \{\}$). To see this note that in Step C6 the simulated $\tilde{\wp}_C$ does indeed behave like $\wp_C$, because $b' = h \oplus b = \mathsf{B}(f^{-1}(r))$ and $r = r_C \oplus r_R$. In Step C3 also the behavior is that of $\wp_C$, because r is a random string independent of r_R, and as such choosing $r_C = r \oplus r_R$ is equivalent to choosing a random string directly. The only other differences are in interactions among $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$, $\tilde{\wp}_C$ and $\tilde{\wp}_R$. However since $\tilde{\wp}_C$ and $\tilde{\wp}_R$ are simulations of honest parties, $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ accepting their proofs leads to a perfect simulation.

Finally recall that conditioned on $\mathfrak{X} \neq \{\}$, $\mathbf{Pr}\left[M^{\Psi^*}(f, r, h) = 0\right] = 1/2$ in both Experiment 1 and Experiment 2. Hence,

$$\begin{aligned}
&\mathbf{Pr}\left[M^{\Psi^*}(f, r, h = \mathsf{B}(f^{-1}(r)) = 0\right] - \mathbf{Pr}\left[M^{\Psi^*}(f, r, h \leftarrow \{0,1\}) = 0\right] \\
&\quad = \mathbf{Pr}\left[\text{EXEC}_{\text{COM}, \mathcal{Z}, \mathcal{A}} = 0 \wedge \mathfrak{X} = \{\}\right] - \mathbf{Pr}\left[\text{EXEC}_{\langle \mathcal{F}_{\text{COM}} \rangle, \mathcal{Z}, \mathcal{S}} = 0 \wedge \mathfrak{X} = \{\}\right].
\end{aligned}$$

Note that M^{Ψ^*} does nothing much more than simulate the Network, and as such is a PPT machine with access to Ψ^*. Thus by Assumption A3, (5.4) is negligible.

R honest, C corrupted. Here we shall use the hiding property of the commitment scheme C and the soundness of $\mathcal{F}_{\widetilde{\text{ZK}}}$ (Lemma 5.10). When R is not corrupted, by the soundness condition of $\mathcal{F}_{\widetilde{\text{ZK}}}$ (Lemma 5.10), if the protocol continues beyond Step C2, except with negligible probability, f is indeed a permutation. This has two consequences: firstly in the simulation, the sender can reveal only to the bit extracted by the simulator (using the soundness of $\mathcal{F}_{\widetilde{\text{ZK}}}$ again). Secondly, $\tilde{r}_R = f(u) \oplus r_C$ as picked by $\tilde{\wp}_R$ is uniformly randomly distributed independent of c, even though f and r_C may depend on c.

To compare $\mathbf{Pr}\left[\text{EXEC}_{\langle\mathcal{F}_{\text{COM}}\rangle,\mathcal{Z},\mathcal{S}} = 0 \wedge \mathfrak{X} = \{C\}\right]$ and $\mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} = 0 \wedge \mathfrak{X} = \{C\}\right]$, we consider an intermediate situation, where $\mathcal{F}_{\text{COM}}$ is replaced by a functionality $\mathcal{F}'$ which behaves exactly like $\mathcal{F}_{\text{COM}}$, except that if $\mathfrak{X} = \{C\}$, then in the reveal stage it accepts a bit b from the sender and sends (reveal, b) to the receiver, irrespective of what bit it received during the commitment phase. Also, we modify $\mathcal{T}$, so that (if $\mathfrak{X} = \{C\}$) in the reveal stage, on behalf of the corrupt $\wp_C$ it sends (reveal, b) to $\mathcal{F}'$, where b is the bit sent by $\wp_C$ in Step R1. (Note that originally $\mathcal{T}$ just sends the message reveal to $\wp_{\mathcal{F}_{\text{COM}}}$, which results in $\wp_{\mathcal{F}_{\text{COM}}}$ sending b^*, the extracted bit, to $\wp_{\langle\mathcal{F}_{\text{COM}}\rangle,R}$ as the revealed bit.) Let $\mathcal{T}'$ denote the modified transvisor, and let $\mathcal{S}' = \mathtt{comb}(\mathcal{A}, \mathcal{T}')$.

As noted above, if the protocol continues beyond Step C2 when R is honest, except with negligible probability, f is indeed a permutation. Conditioned on f being a permutation, the statement defined in Step R2 of the Reveal Phase is true only if $b = b^*$. So if $b \neq b^*$ the probability that $\wp_C$ can make $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ in Step R2 accept the proof is negligible. Hence, only if $b = b^*$ does the execution ($\text{EXEC}_{\langle\mathcal{F}'\rangle,\mathcal{Z},\mathcal{S}'}$ as well as $\text{EXEC}_{\langle\mathcal{F}_{\text{COM}}\rangle,\mathcal{Z},\mathcal{S}}$) proceed beyond that step, except with negligible probability. Thus

$$|\,\mathbf{Pr}\left[\text{EXEC}_{\langle\mathcal{F}'\rangle,\mathcal{Z},\mathcal{S}'} \wedge \mathfrak{X} = \{C\}\right] - \mathbf{Pr}\left[\text{EXEC}_{\langle\mathcal{F}_{\text{COM}}\rangle,\mathcal{Z},\mathcal{S}} \wedge \mathfrak{X} = \{C\}\right]| \tag{5.7}$$

is negligible, because conditioned on $b = b^*$ or on execution not proceeding to the last step, the two executions are identical.

Now, we show that

$$|\,\mathbf{Pr}\left[\text{EXEC}_{\langle\mathcal{F}'\rangle,\mathcal{Z},\mathcal{S}'} \wedge \mathfrak{X} = \{C\}\right] - \mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} \wedge \mathfrak{X} = \{C\}\right]| \tag{5.8}$$

is negligible as well. Suppose not. Then we build a distinguisher M^{Ψ^*} (non-uniform PPT machine, with access to Ψ^*) which can distinguish between the following experiments: in one experiment it is given (c, a) where a is a random string and $c = \mathsf{C}(a)$ is a random commitment to a, and in the other it gets (c, a) where a is as before, but $c = \mathsf{C}(a')$, where a' is an independently and uniformly chosen random string.

Construction 5.7: M^{Ψ^*} to take (c, a) and distinguish between $c = \mathsf{C}(a)$ and $c = \mathsf{C}(a')$.

M^{Ψ^*} accepts (c, a) as input. Then it internally simulates the Network corresponding to the execution $\text{EXEC}_{\langle\mathcal{F}'\rangle,\mathcal{Z},\mathcal{S}'}$ internally. If $\mathfrak{X} \neq \{C\}$ it bails out and outputs a random bit. Otherwise it proceeds with the simulation, but with the following

differences:

- In Step C1, the simulated program $\tilde{\wp}_R$ sends c that M^{Ψ^*} gets as input (instead of picking a random r_R and sending $c = \mathsf{C}(r_R)$).
- In Step C4, $\tilde{\wp}_R$ sends a instead of $\tilde{r}_R$.
- After Step C6, $\mathcal{T}'$ does not attempt to extract b^* (which it cannot because it would not know $f^{-1}(a \oplus r_C)$). Instead it simply sends an arbitrary bit (say 0) to $\mathcal{F}'$.

Finally, M^{Ψ^*} outputs what $\mathcal{Z}$ outputs.

Firstly, we observe that when the input is (c, a) such that a is random and $c = \mathsf{C}(a)$, the outcome of the simulated execution is identical to $\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}}$. This is because, after this change, the only difference between the actual execution $\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}}$ and the simulated execution is that in Step C5, in the former $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ and $\wp_R$ carry out the proof (for the statement $(\exists r' : c = \mathsf{C}(a; r'))$), where as in the latter $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ and $\tilde{\wp}_R$ do not carry out this proof. Nevertheless in both cases, the outcome of this interaction is the same: in the former $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ sends a message proven to $\wp_C$, and in the latter $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ sends this message to $\wp_C$. Thus this difference does not have any further effect in the executions.

Secondly, we observe that when the input is (c, a) such that a is random and $c = \mathsf{C}(a')$ where a' is another independent random string, then the outcome of the simulated execution is identical to $\text{EXEC}^{\langle \mathcal{F}' \rangle, \mathcal{Z}, \mathcal{S}'}$. To see this, we set $r_R = a'$ and $\tilde{r}_R = a$. In $\text{EXEC}^{\langle \mathcal{F}' \rangle, \mathcal{Z}, \mathcal{S}'}$, the transvisor $\mathcal{T}'$ picks $\tilde{r}_R$ as $f(u) \oplus r_C$, where u is independently chosen. But if f is a permutation (which is the case, except with negligible probability), setting $\tilde{r}_R$ directly to a uniformly randomly chosen string has the same effect. Note that though $\mathcal{T}'$ does not have access to the values a' and $u = f^{-1}(a \oplus r_C)$, it does not need either of these values (unlike the actual receiver $\wp_R$ and unlike the original transvisor $\mathcal{T}$).

Thus the advantage of M^{Ψ^*} in distinguishing between $(c = \mathsf{C}(a), a)$ and $(c = \mathsf{C}(a'), a)$ is the same as the expression in (5.8). Further, M^{Ψ^*} is a PPT machine with access to Ψ. Hence, from the hiding property of C (assumed to hold against PPT circuits with access to Ψ^*), this advantage must be negligible.

Since the expressions in (5.7) and (5.8) are both negligible, we obtain that the expression (5.5) is also negligible.

C honest, R corrupted. Recall that in this case, the simulation is the same as that in the case when both C and R are honest. The proof of indistinguishability is also almost the same as in that case, except now the receiver is corrupt, and so M^{Ψ^*} cannot choose r_R. This is taken care of by making use of the non-uniform nature of the distinguisher.

Consider an execution $\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}}$. Let α be the random variable denoting the state of the entire Network after Step C1. This state excludes the randomness that is not yet

sampled by the honest party, $\wp_C$. (Indeed the only non-trivial contents of α are the states of $\mathcal{Z}$ and $\mathcal{A}$, and the value c received by $\wp_C$ in Step C1.) Note that α can also be considered as the state of the Network corresponding to the execution $\text{EXEC}_{\langle \text{COM} \rangle, \mathcal{Z}, \mathcal{S}}$ by ascribing the state of the honest party $\wp_C$ to that of the simulated program $\tilde{\wp}_C$. This is so because $\mathcal{T}$ faithfully follows the protocol for $\wp_C$ until Step C6.

Let α^* be the value of α such that

$$|\mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} = 0 \wedge \mathfrak{X} = \{R\} | \alpha = \alpha^*\right] - \mathbf{Pr}\left[\text{EXEC}_{\langle \mathcal{F}_{\text{COM}} \rangle, \mathcal{Z}, \mathcal{S}} = 0 \wedge \mathfrak{X} = \{R\} | \alpha = \alpha^*\right]| \tag{5.9}$$

is maximized. This quantity is no less than that in (5.6), and so we need to only show that this is negligible.

Let c^* be the first message in the protocol according to the state α^*, and let r^* be such that $c^* = \mathsf{C}(r^*; r')$. If no such r^* exists, then the probability that the execution proceeds beyond Step C5 is negligible, and then the expression in (5.9) is negligible. So we shall assume it exists and proceed. Also, we assume that $\mathfrak{X} = \{R\}$ according to α^*, because otherwise both the probabilities in (5.9) are 0.

Now we describe a non-uniform PPT machine $M_{\alpha^*}^{\Psi^*}$ such that, if (5.9) is non-negligible, then $M_{\alpha^*}^{\Psi^*}$ can distinguish between a random bit and $\mathsf{B}(f^{-1}(r))$, for a randomly chosen f and r with non-negligible probability.

Construction 5.8: $M_{\alpha^*}^{\Psi^*}$ to distinguish between $h \leftarrow \{0,1\}$ and $h = \mathsf{B}(f^{-1}(r))$

$M_{\alpha^*}^{\Psi^*}$ gets α^* and r^* as non-uniform advice. It takes (f, r, h) as input where f is randomly drawn from $\mathsf{T}(1^k)$, $r \leftarrow \{0,1\}^k$ and h is either a random bit (Experiment 1) or $h = \mathsf{B}(f^{-1}(r))$ (Experiment 2). It tries to distinguish between the two experiments as follows. On receiving the inputs (f, r, h) it simulates the entire Network execution corresponding to $\text{EXEC}_{\langle \mathcal{F}_{\text{COM}} \rangle, \mathcal{Z}, \mathcal{S}}$, starting at state α^*, but modifies the execution as follows (similar to M^{Ψ^*} described in Construction 5.6):

- At Step C2, $\tilde{\wp}_C$ sends f (received as input) to $\tilde{\wp}_R$ (instead of picking $(f, f^{-1}) \leftarrow \mathsf{T}(1^k)$).
- At Step C2, the simulated $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ instance sends the message proven to $\wp_R$, without $\tilde{\wp}_C$ having to interact with it.
- At Step C3, $\tilde{\wp}_C$ sends $r_C = r \oplus r^*$, instead of picking a random r_C.
- If r_R sent by $\wp_R$ in Step C4 is not the same as r^* *and* in Step C5 if $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ accepts the proof from $\wp_R$ (i.e., $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ sends the message proven to $\tilde{\wp}_C$), then $M_{\alpha^*}^{\Psi^*}$ aborts (say it outputs 0).
- At Step C6, $\tilde{\wp}_C$ sends $b' = h \oplus b$, instead of sending a random bit.

Finally $M_{\alpha^*}^{\Psi^*}$ outputs what $\mathcal{Z}$ outputs.

Firstly, we point out that the probability of $M_{\alpha^*}^{\Psi^*}$ aborting (on the simulation reaching Step C5) is negligible. To see this, note that C is a perfectly binding commitment scheme. So to make $M_{\alpha^*}^{\Psi^*}$ abort, it must be that $r_R \neq r^*$ and $\wp_R$ must prove to $\tilde{\wp}_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ a false statement. But by the soundness of $\mathcal{F}_{\widetilde{\text{ZK}}}$ (Lemma 5.10) this happens only with negligible probability.

Conditioned on $M_{\alpha^*}^{\Psi^*}$ not aborting, we observe that it behaves just like M^{Ψ^*} in Construction 5.6 in Experiments 1 and 2. Following the same arguments as there, but observing that $M_{\alpha^*}^{\Psi^*}$ starts from α^*, we obtain

$$\begin{aligned}&\mathbf{Pr}\left[M_{\alpha^*}^{\Psi^*}(f, r, h = \mathsf{B}(f^{-1}(r)) = 0\right] - \mathbf{Pr}\left[M_{\alpha^*}^{\Psi^*}(f, r, h \leftarrow \{0,1\}) = 0\right] \\ &\quad = \mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} = 0 \wedge \mathfrak{X} = \{\} | \alpha = \alpha^*\right] - \mathbf{Pr}\left[\text{EXEC}_{\langle\mathcal{F}_{\text{COM}}\rangle,\mathcal{Z},\mathcal{S}} = 0 \wedge \mathfrak{X} = \{\} | \alpha = \alpha^*\right].\end{aligned}$$

Since $M_{\alpha^*}^{\Psi^*}$ does little more than simulate the Network, starting with a state of the Network as non-uniform advice, it is a PPT machine with access to Ψ. Thus by Assumption A3, (5.9) is negligible, and hence so is (5.6).

To conclude the proof, we have shown that (5.3), (5.4), (5.5) and (5.6) are all negligible. This implies that (5.2) is indeed negligible as we set out to prove.[16] □

5.7 One-to-Many Commit and Prove Functionality

In this section we outline the proof of Lemma 5.1, which completes the proof of our main theorem- Theorem 5.1. Following the approach in [CLOS02], we use two other functionalities, namely Zero-Knowledge ($\mathcal{F}_{\text{ZK}}$) and Authenticated Broadcast ($\mathcal{F}_{\text{BC}}$) to securely realize $\mathcal{F}_{\text{CP}}^{1:\text{M}}$.

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

$\mathcal{F}_{\text{ZK}}$ proceeds as follows, running with a prover P, a verifier V and an adversary $\mathcal{S}$, and parameterized with a relation R:

- Upon receiving (prove, x, w) from P, do: if $(x, w) \in R$, then send (proven, x) to V and $\mathcal{S}$ and halt. Otherwise, halt.

Figure 5.9: $\mathcal{F}_{\text{ZK}}$ functionality

Zero Knowledge Functionality Canetti and Fischlin [CF01] show that a simple parallel repetition of the well-known 3-round protocol for zero knowledge proofs suggested by Blum [Blu82], in which a commitment subroutine is replaced by $\langle\mathcal{F}_{\text{COM}}\rangle$, yields a protocol ZK which securely realizes $\mathcal{F}_{\text{ZK}}$. The proof, as noted there, is information theoretic and independent of computational assumptions. As such it is easily verified that the same protocol securely realizes $\mathcal{F}_{\text{ZK}}$ with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}_{\mathsf{static}}^{\Psi}, \mathcal{S}_{\mathsf{static}}^{\Psi})$. We state this below.

Lemma 5.12. *(Following [CF01]) There is a protocol* ZK *using* $\langle \mathcal{F}_{\text{COM}} \rangle$ *as a subroutine which securely realizes* $\mathcal{F}_{\text{ZK}}$ *with respect to* $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$.

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

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

- Upon receiving a message $(\mathcal{P}, x)$ from P_i, where $\mathcal{P}$ is a set of parties, send $(P_i, \mathcal{P}, x)$ to all parties in $\mathcal{P}$ and to $\mathcal{S}$, and halt.

Figure 5.10: The ideal broadcast functionality

Authenticated Broadcast The functionality $\mathcal{F}_{\text{BC}}$ ensures that all the parties to which a message is addressed receive the same message (if they do receive the message). Following [CLOS02], we use the protocol from [GL02]. The protocol in [GL02] securely realizes $\mathcal{F}_{\text{BC}}$ in an information-theoretic manner: it does not require any computational restrictions on the class of adversaries. Thus, in particular, we have the following.

Lemma 5.13. *([GL02]) Protocol* BC *securely realizes* $\mathcal{F}_{\text{BC}}$ *with respect to* $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$.

Functionality $\mathcal{F}_{\text{CP}}^{1:\text{M}}$

The parties are a sender C and a set of possible receivers $P_1, \ldots, P_n$, with an adversary $\mathcal{S}$. The functionality is parameterized by a relation R. The security parameter is k.

COMMIT PHASE

- Upon receiving a message $(\mathsf{commit}, \mathcal{P}, w)$ from C where $\mathcal{P}$ is a set of parties and $w \in \{0,1\}^k$, append the value w to the list $\overline{w}$, record $\mathcal{P}$, and send the message $(\mathsf{receipt}, C, \mathcal{P})$ to all parties $P \in \mathcal{P}$ and to $\mathcal{S}$. (Initially, the list $\overline{w}$ is empty). But, if a commit message has already been received with a different set of parties $\mathcal{P}' \neq \mathcal{P}$ ignore this message.

PROVE PHASE

- Upon receiving a message (prove, x) from C, where $x \in \{0,1\}^{\text{poly}(k)}$, compute $R(x, \overline{w})$. If $R(x, \overline{w}) = 1$, then send the message (proven, x) to all parties $P_i \in \mathcal{P}$ and to $\mathcal{S}$. Otherwise, ignore the message.

Figure 5.11: The One-to-many commit-and-prove functionality

In Figure 5.12 we present the protocol OMCP, which uses $\langle \mathcal{F}_{\text{BC}} \rangle$ and $\langle \mathcal{F}_{\text{ZK}} \rangle$ as subroutines. To commit to a value w, the sender C computes a commitment c to w under a perfectly binding commitment C obtained from the trapdoor-permutation of Assumption A3 (which remains hiding even to adversaries with sampling access to $\mathcal{D}_r^{\mu}$ for all μ and r). Then it broadcasts c and proves to each party separately, using the $\mathcal{F}_{\text{ZK}}$ functionality, that c is indeed a valid commitment. Each party on receiving this proof broadcasts this fact. If all parties accept the respective proofs and announce it, they all proceed to

accept the commitment by adding c to a list $\overline{c}$. Later, to prove $R(x, \overline{w})$, where x is an input and $\overline{w}$ is the list of all commitments made so far, the C proofs the statement (formulated in terms of x and $\overline{c}$) to each party separately using the $\mathcal{F}_{\text{ZK}}$ functionality. As before, on accepting the $\mathcal{F}_{\text{ZK}}$ proof, each party broadcasts this fact. Finally each party accepts the proof if it receives this broadcast message from all parties.

Protocol OMCP

The parties are a sender C and a set of possible receivers $P_1, \ldots, P_n$, with an adversary $\mathcal{A}$. The protocol is parameterized by a relation R. The security parameter is k.

COMMIT PHASE

C1 On input $(\mathsf{commit}, \mathcal{P}, w)$ (where $w \in \{0,1\}^k$), the sender C computes $c = \mathsf{C}(w; r)$ using a randomly chosen r. It adds r to a list $\overline{r}$ (initially empty).

C2 C broadcasts the message $(\mathcal{P}, c)$ by sending it to $\mathcal{F}_{\text{BC}}$.

C3 For each $P_j \in \mathcal{P}$, party C sends the message $(\mathsf{prove}, c, (w, r))$ to a copy of $\mathcal{F}_{\text{ZK}}$ invoked with P_j as the verifier, and parameterized by the relation $R' = \{(c, (w, r)) \mid c = \mathsf{C}(w; r)\}$.

C4 On receiving $(C, \mathcal{P}, c)$ from $\mathcal{F}_{\text{BC}}$ and (proven, c) from $\mathcal{F}_{\text{ZK}}$, (after verifying that both values of c are the same) each $P_j \in \mathcal{P}$ broadcasts the message $\mathsf{acceptable}$.

C5 On receiving the message $\mathsf{acceptable}$ from all parties in $\mathcal{P}$, party P_j adds c to a list $\overline{c}$ (initially $\overline{c}$ is empty; C also maintains this list).

PROVE PHASE

P1 On input (prove, x), the C broadcasts the message $(\mathcal{P}, x)$ by sending it to $\mathcal{F}_{\text{BC}}$.

P2 For each $P_j \in \mathcal{P}$, party C sends the message $(\mathsf{prove}, (x, \overline{c}), (\overline{w}, \overline{r}))$ to a copy of $\mathcal{F}_{\text{ZK}}$ invoked with P_j as the verifier and parameterized by the relation $R'' = \{((x, \overline{c}), (\overline{w}, \overline{r})) \mid R(x, \overline{w}) \bigwedge \overline{c}_j = \mathsf{C}(\overline{w}_j; \overline{r}_j) \text{ for all } j\}$

P3 On receiving $(\mathcal{P}, x)$ from $\mathcal{F}_{\text{BC}}$ and $(\mathsf{proven}, (x, \overline{c}))$ from $\mathcal{F}_{\text{ZK}}$, (after verifying that both x are the same, and $\overline{c}$ and $\mathcal{P}$ match the locally stored values) each $P_j \in \mathcal{P}$ broadcasts the message true x.

P4 On receiving the message true x from all parties in $\mathcal{P}$, party P_j accepts x.

Figure 5.12: A protocol which uses protocols $\langle \mathcal{F}_{\text{ZK}} \rangle$ and $\langle \mathcal{F}_{\text{BC}} \rangle$ as subroutines and securely realizes $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$.

Lemma 5.14. *Protocol* OMCP *is as secure as* $\langle \mathcal{F}_{\text{CP}}^{1:\text{M}} \rangle$ *with respect to* $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$*, under Assumption A3.*

PROOF: The proof follows from the security of the commitment scheme C.[17] Below we describe a simple transvisor $\mathcal{T} = \mathcal{T}^{\langle \mathcal{F}_{\text{CP}}^{1:\text{M}} \rangle \to \text{OMCP}}$, and will then argue that the simulation provided by $\mathcal{T}$ is good.

Construction 5.9: $\mathcal{T} = \mathcal{T}^{\langle \mathcal{F}_{\text{CP}}^{1:\text{M}} \rangle \to \text{OMCP}}$

$\mathcal{T}$ simulates all the honest parties' programs (as $\tilde{\wp}_C$ or $\tilde{\wp}_{P_i}$) and also all the instances of the functionalities $\mathcal{F}_{\text{ZK}}$ and $\mathcal{F}_{\text{BC}}$ (as $\tilde{\wp}_{\mathcal{F}_{\text{ZK}}}$ and $\tilde{\wp}_{\mathcal{F}_{\text{BC}}}$). Further it receives the messages that $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ sends to $\mathcal{S}$, and also controls the message delivery from $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ to all the parties in the session. $\mathcal{T}$ lets $\mathcal{A}$ control the message delivery from $\tilde{\wp}_{\mathcal{F}_{\text{ZK}}}$ and $\tilde{\wp}_{\mathcal{F}_{\text{BC}}}$ to $\tilde{\wp}_C$ or $\tilde{\wp}_{P_i}$, which determines if and when $\tilde{\wp}_{P_i}$ produces local outputs. $\mathcal{T}$ lets $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ send the commit message to a party P_i when the simulated program $\tilde{\wp}_{P_i}$ locally outputs the message acceptable . Similarly, when $\tilde{\wp}_{P_i}$ locally accepts a proof, $\mathcal{T}$ lets $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ send the proven message to P_i.
For the simulated honest verifiers $\tilde{\wp}_{P_i}$ simply run exactly according to the protocol specification. To describe the rest of the simulation, we consider two cases: when C is honest, and when C is corrupt.

C **Honest:** In the simulation $\tilde{\wp}_C$ runs exactly as specified by the protocol, with the following differences.

- In Step C1, $\tilde{\wp}_C$ sets $c = \mathsf{C}(0^k; r)$ where 0^k is the all zero string.
- In Step C3 and Step P2, $\tilde{\wp}_C$ does not send any message to the simulated $\mathcal{F}_{\text{ZK}}$ functionality, $\tilde{\wp}_{\mathcal{F}_{\text{ZK}}}$. Instead $\tilde{\wp}_{\mathcal{F}_{\text{ZK}}}$ simply sends the proven messages to the other parties.

C **Corrupt:** In this case the messages received by $\tilde{\wp}_{\mathcal{F}_{\text{ZK}}}$ from the corrupted sender C are used to "extract" the messages which then $\mathcal{T}$ (acting as the sender in the ideal protocol $\langle \mathcal{F}_{\text{CP}}^{1:\text{M}} \rangle$) sends to $\mathcal{F}_{\text{CP}}^{1:\text{M}}$. The extraction is straightforward: the messages from $\mathcal{A}$ in Step C3 and Step P2 contain all w and x values that $\mathcal{T}$ can send to $\mathcal{F}_{\text{CP}}^{1:\text{M}}$. Note that C sends x to different sessions of $\mathcal{F}_{\text{ZK}}$ (with different verifiers P_i), and as such these values could be different. However, since C is perfectly binding and C uses $\mathcal{F}_{\text{BC}}$ to broadcast its messages (rather than send separate messages to each party), there is only one value of x that the different sessions of $\mathcal{F}_{\text{ZK}}$ would accept (corresponding to a given c that was broadcast). Thus the the extracted value is indeed well-defined.
As soon as a value is extracted it is sent to $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ (but, as mentioned earlier, the message from $\mathcal{F}_{\text{CP}}^{1:\text{M}}$ is allowed to reach an honest party P_i only when $\tilde{\wp}_{P_i}$ produces the corresponding local output).

Conditioned on C being corrupt, it is readily seen that the simulation is perfect. Below we argue that

$$|\,\mathbf{Pr}\left[\text{EXEC}_{\text{OMCP},\mathcal{Z},\mathcal{A}} = 0 \wedge C \notin \mathfrak{X}\right] - \mathbf{Pr}\left[\text{EXEC}_{\langle \mathcal{F}_{\text{CP}}^{1:\text{M}} \rangle,\mathcal{Z},\mathcal{S}} = 0 \wedge C \notin \mathfrak{X}\right]| \tag{5.10}$$

is negligible. For the sake of contradiction, suppose otherwise. To reduce this to a contradiction to the hiding property of C, we construct m hybrid situations, where $m = \text{poly}(k)$ is a bound on the number of items in $\overline{w}$.[18]

Construction 5.10: Hybrid executions corresponding to m uses of C.

In the i-th hybrid situation, the modified transvisor $\mathcal{T}_i$ has access to the input to C from $\mathcal{Z}$ for the first i commitments. $\mathcal{T}_i$ carries out the first i commitments as prescribed the protocol using these input values (instead of committing to 0^k). Otherwise $\mathcal{T}_i$ is identical to $\mathcal{T}$.

Above, for $i = 0$, $\mathcal{T}_i$ is the same as $\mathcal{T}$, and for $i = m$, this hybrid situation is a perfect simulation of the execution $\text{EXEC}_{\text{OMCP},\mathcal{Z},\mathcal{A}}$. So, if (5.10) is non-negligible, then there is some value of i for which the corresponding difference between the $i-1^{\text{st}}$ and i^{th} hybrids above is non-negligible. In that case we can construct a machine M^{Ψ^*} to break the hiding property of C as follows:

Construction 5.11: M^{Ψ^*} to break security of C.

M^{Ψ^*} emulates the Network corresponding to the i^{th} hybrid described above. If C is corrupted M^{Ψ^*} aborts; otherwise it continues with the emulation. But when it is time for it to generate the i^{th} commitment as $c = \mathsf{C}(w; r)$ it sends out the pair of messages $(w, 0^k)$ to the experiment defining the security of C. The experiment will randomly pick one of the two messages and replies with c to M^{Ψ^*}, where c is a commitment to that message, M^{Ψ^*} will use c as the i^{th} commitment in its emulation. Finally it outputs what $\mathcal{Z}$ outputs.

Above, conditioned on the experiment picking $c = \mathsf{C}(w; r)$ the situation is identical to that in the i^{th} hybrid, where as picking $c = \mathsf{C}(0^k; r)$ is identical to the $i - 1^{\text{st}}$ hybrid. Thus the advantage $\mathcal{Z}$ has in distinguishing between the two hybrids translates directly into an advantage for M^{Ψ^*} in the experiment defining the security of C. Thus if (5.10) is non-negligible we reach a contradiction.

Thus, we conclude that $|\mathbf{Pr}\left[\text{EXEC}_{\text{OMCP},\mathcal{Z},\mathcal{A}} = 0\right] - \mathbf{Pr}\left[\text{EXEC}_{\langle\mathcal{F}_{\text{CP}}^{1:\text{M}}\rangle,\mathcal{Z},\mathcal{S}} = 0\right]|$ is indeed negligible, proving the theorem. □

We have collected the results regarding all the new protocols as Lemma 5.1. In terms of Table 5.4, Lemma 5.1 captures all the rows in after the first three. Indeed the final protocol π_{11} can be written as $\pi_3^{\text{OMCP-REAL}/\mathcal{F}_{\text{CP}}^{1:\text{M}}}$. Now we can give a proof for it by putting together the above results.

PROOF OF LEMMA 5.1: It follows from Lemma 5.14 by substituting the ideal functionalities by the protocols BC, ZK, COM, BZK, BCOM, ENC (see Section 5.8) and AMD

(see Section 5.9) one after the other. The result of these substitutions is a new protocol OMCP-REAL which is realistic (i.e., does not use any ideal functionalities). That this protocol is as secure as $\langle \mathcal{F}_{\text{CP}}^{1:\text{M}} \rangle$ with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$ follows from applying the composition theorem (Theorem 4.1) for each substitution (making use of the fact that ZK is as secure as $\langle \mathcal{F}_{\text{ZK}} \rangle$ etc.), and then using (for a constant number of times) the transitivity of the "as secure as" relation (Lemma 3.1).

To finish the proof of Lemma 5.1 we need to observe that this protocol is also a *useful realization* (Definition 3.2) of $\mathcal{F}_{\text{CP}}^{1:\text{M}}$. That is, we need to verify that the "as secure as" relation derived above holds with respect to $(\mathcal{Z}^{\Psi}, \mathring{\mathcal{A}}, \mathring{\mathcal{S}})$ as well. This follows because the protocols in all the substitutions used above are also useful. To see that is indeed the case for the protocols we showed secure in the previous sections, note that for these protocols the same simulators we described for $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$ can be used for $(\mathcal{Z}^{\Psi}, \mathring{\mathcal{A}}, \mathring{\mathcal{S}})$.[19] □

5.8 Encryption

The encryption functionality $\mathcal{F}_{\text{ENC}}$ is simple: $\wp_{\mathcal{F}_{\text{ENC}}}$ establishes a session with a sender $\wp_{P_1}$ and a receiver $\wp_{P_2}$. Then, when $\wp_{P_1}$ sends a message msg to $\wp_{\mathcal{F}_{\text{ENC}}}$, the functionality $\wp_{\mathcal{F}_{\text{ENC}}}$ sends the length of the message $|\mathsf{msg}|$ to the adversary and the message itself to $\wp_{P_2}$. (Recall that by the convention specified in Section 2.3.3, the adversary (or any other party) will not have access to msg. However, it decides if and when msg becomes available to $\wp_{P_2}$.) We allow the same instance of $\wp_{\mathcal{F}_{\text{ENC}}}$ to communicate multiple messages from $\wp_{P_1}$ to $\wp_{P_2}$.

Canetti [Can01] observes that any public key encryption scheme secure against adaptive chosen cipher text attacks (also called a CCA2-secure encryption scheme) can be converted into a protocol ENC which securely realizes $\mathcal{F}_{\text{ENC}}$ with respect to $(\mathcal{Z}, \mathcal{A}_{\mathsf{static}}, \mathcal{S}_{\mathsf{static}})$. The protocol simply involves using the CCA2-secure encryption scheme, but for each message using a freshly generated public-key/private-key pair. It is well-known that the underlying encryption schemes can be constructed based on trapdoor permutations[20] [DDN00, Sah99, Lin03c].

Lemma 5.15. *[Can01] Assuming the existence of a collection of trapdoor permutations, there is a protocol* ENC *which securely realizes* $\langle \mathcal{F}_{\text{ENC}} \rangle$ *with respect to* $(\mathcal{Z}, \mathcal{A}_{\mathsf{static}}, \mathcal{S}_{\mathsf{static}})$.

Note that we do not require ENC to be secure with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$, but only with respect to $(\mathcal{Z}, \mathcal{A}_{\mathsf{static}}, \mathcal{S}_{\mathsf{static}})$. This is sufficient because since $\mathcal{F}_{\text{ENC}}$ is a "low level" component of our construction. So in OMCP we could first substitute all "higher level" ideal subroutines with corresponding protocols to create a protocol which uses $\langle \mathcal{F}_{\text{ENC}} \rangle$ as the only subroutine (other than $\langle \mathcal{F}_{\text{AMD}} \rangle$); then we substitute $\langle \mathcal{F} \rangle$ by the protocol ENC. Using the extended UC theorem (Theorem 4.2) we can still guarantee security of the protocol thus obtained, with respect to $(\mathcal{Z}, \mathcal{A}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$.

However, using the trapdoor permutations from Assumption A3, it is no more difficult to make ENC secure with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$. Using the trapdoor permutations in Assumption A3, the CCA-2 secure encryptions of [DDN00, Sah99, Lin03c] become secure against PPT machines M^{Ψ^*} with access to Ψ^*. (This can be easily verified, because the proofs are all black-box proofs which relativize with respect to Ψ^*.) Employing such an encryption scheme in the protocol ENC we obtain the following lemma.

Lemma 5.16. *Under Assumption A3, there is an encryption scheme such that if protocol* ENC *uses this scheme, then it securely realizes* $\langle \mathcal{F}_{\text{ENC}} \rangle$ *with respect to* $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$.

5.9 Authenticated Message Delivery

In all our protocols we assume that messages sent by parties to each other are authenticated. That is, the adversary cannot insert forged messages into the Network without being detected. The basic idea behind implementing such an authentication mechanism is to use digital signatures. However there are a few subtleties involved in formulating an authentication mechanism and capturing the nature of security this offers, when there are no trusted setups (like a public-key infrastructure). The subtleties stem from the fact that there is no way to tie real-world identities to identities advertised in messages received by a party. Such identification of virtual and real world identities would have allowed the parties to have a consistent view of which parties they are communicating with. However, there is a simple solution: we use the verification keys for the signature scheme as the identities. This still does not let one associate real-life identities with the virtual identities, but lets one have a consistent and unforgeable identity (or more than one identity) in the Network. We present a functionality $\mathcal{F}_{\text{AMD}}$ capturing the security that can be obtained using a signature-based scheme.

It is easy to see that the protocol AMD securely realizes $\mathcal{F}_{\text{AMD}}$ if the signature scheme used is (existentially) unforgeable against chosen text attacks. Such signature schemes can be constructed using one-way functions [Rom90], or using collision-resistant hash functions [Dam87, Gol04].

Lemma 5.17. *Assuming the existence of a signature scheme unforgeable against chosen text attacks, protocol* AMD *securely realizes* $\langle \mathcal{F}_{\text{AMD}} \rangle$ *with respect to* $(\mathcal{Z}, \mathcal{A}_{\mathsf{static}}, \mathcal{S}_{\mathsf{static}})$.

Note that, as was the case with the encryption protocol ENC (Section 5.8), we need not require AMD to be secure with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$, but only with respect to $(\mathcal{Z}, \mathcal{A}_{\mathsf{static}}, \mathcal{S}_{\mathsf{static}})$, because $\mathcal{F}_{\text{AMD}}$ is used at the "lowest level" of our construction. Using the extended UC theorem (Theorem 4.2) this will still let us guarantee the security of the final protocol (obtained after substituting $\langle \mathcal{F}_{\text{AMD}} \rangle$ by AMD) with respect to $(\mathcal{Z}, \mathcal{A}_{\mathsf{static}}, \mathcal{S}^{\Psi}_{\mathsf{static}})$.

But in fact, using either of Assumption A2 or Assumption A3, the constructions of signature schemes mentioned above can be made unforgeable even for adversaries which have access to Ψ^*. Hence, we obtain

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

The parties are denoted by $P_1, P_2, \ldots$, and adversary by $\mathcal{S}$.
IDENTITY ESTABLISHMENT PHASE: $\wp_{\mathcal{F}_{\text{AMD}}}$ maintains a list $\overline{\mu}$ of all the identities established so far, and a list $\mathfrak{X}$ of all the identities used by the adversary so far.

E1 To obtain a new identity $\wp_{P_i}$ sends id-request to $\wp_{\mathcal{F}_{\text{AMD}}}$.

E2 $\wp_{\mathcal{F}_{\text{AMD}}}$ picks an identity string μ according to a (high-entropy) distribution over $\mathcal{I}_k$.[21] If $\mu \in \overline{\mu}$ or $\mu \in \mathfrak{X}$ then $\wp_{\mathcal{F}_{\text{AMD}}}$ sends failed to $\wp_{P_i}$; else it sends μ to $\wp_{P_i}$ and adds it to $\overline{\mu}$.

E3 $\wp_{P_i}$ outputs μ and internally records it as well (or if it received failed , it outputs that).

MESSAGE DELIVERY:

D1 $\wp_{P_i}$ receives input (To: $\langle pid; \mu' \rangle$, msg) which signifies that the message msg needs to be delivered to a program with ID pid belonging to a party with ID μ'.

D2 $\wp_{P_i}$ sends (To: $\langle pid; \mu' \rangle$, msg) to $\wp_{\mathcal{F}_{\text{AMD}}}$.

D3 If $\wp_{P_i}$ established an ID μ with $\wp_{\mathcal{F}_{\text{AMD}}}$, then $\wp_{\mathcal{F}_{\text{AMD}}}$ sends the message ($msgid$, From:μ, To: $\langle pid; \mu' \rangle$, msg) to $\mathcal{S}$, where $msgid$ is a k-bit message ID string chosen uniformly at random conditioned on being different from all previous message IDs. (If $\wp_{P_i}$ has not established an ID, it must be corrupt. Then $\wp_{\mathcal{F}_{\text{AMD}}}$ ignores this message.)

D4 If $\mathcal{S}$ sends a message (release, $msgid$), and if there is a party $\wp_{P_j}$ which has established an ID μ', then $\wp_{\mathcal{F}_{\text{AMD}}}$ sends the message (From:μ, To:pid, msg) to $\wp_{P_j}$. ($\mathcal{S}$ may send release commands multiple times with the same $msgid$ and each time the message is delivered to $\wp_{P_j}$.)

D5 If $\mathcal{S}$ sends a message (From:μ, To: $\langle pid; \mu' \rangle$, msg) where $\mu \in \mathfrak{X}$ then $\wp_{\mathcal{F}_{\text{AMD}}}$ sends the message (From:μ, To: $\langle pid; \mu' \rangle$, msg) to $\wp_{P_j}$.

D6 On receiving a message (From:μ, To:pid, msg), $\wp_{P_j}$ sends (From:μ, msg) to the program with ID pid (if such a program exists in the party to which $\wp_{P_j}$ belongs).

Figure 5.13: The Authenticated Message Delivery Functionality

Lemma 5.18. *Under Assumption A2 or Assumption A3, there is a signature scheme such that if protocol* AMD *uses this scheme, then it securely realizes $\langle \mathcal{F}_{\text{AMD}} \rangle$ with respect to $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\text{static}}, \mathcal{S}^{\Psi}_{\text{static}})$.*

5.10 Remarks

We make a few remarks on our construction and assumptions.

5.10.1 Reducing the Assumptions

Complexity Leveraging to Reduce Assumptions. By choosing parameters appropriately, at least one of our assumptions can be reduced to a more standard one. Specifically,

Protocol AMD

The parties are denoted by $P_1, P_2, \ldots$, and adversary by $\mathcal{A}$.
IDENTITY ESTABLISHMENT PHASE:

E1 To establish a new identity, $\wp_{P_i}$ randomly generates a pair (SK, VK) of signing and verification keys. It internally stores this pair and outputs $\mu = VK$ as the identity.

MESSAGE DELIVERY:

D1 $\wp_{P_i}$ receives input (To: $\langle pid; \mu' \rangle$, msg) which signifies that the message msg needs to be delivered to a program with ID pid belonging to a party with ID μ'.

D2 $\wp_{P_i}$ signs the tuple (From:μ, To: $\langle pid; \mu' \rangle$, msg) using the signing key SK, to obtain a signature sign .

D3 $\wp_{P_i}$ sends (From:μ, To: $\langle pid; \mu' \rangle$, msg, sign, μ) to $\mathcal{A}$.

D4 If $\wp_{P_j}$ with identity μ' receives a message (From:μ, To: $\langle pid; \mu' \rangle$, msg, sign, μ) from $\mathcal{A}$, then it verifies sign using μ as verification key. If the signature verifies then it sends sends (From:μ, msg) to the program with ID pid (if such a program exists in the party to which $\wp_{P_j}$ belongs).

Figure 5.14: The Authenticated Message Delivery Protocol

Assumption A3, which assumes trapdoor permutations secure against adversaries with sampling access to $\mathcal{D}_r^\mu$ can be replaced with an assumption of trapdoor permutations secure against super-polynomial adversaries.

Consider choosing the domain of the trapdoor permutation $\{0,1\}^n$ such that the input size of the hash function $k = n^\epsilon$, for some constant $0 < \epsilon < 1$. Then we can safely replace Assumption A3 by the assumption that the trapdoor permutations are secure against circuits of size 2^{n^ϵ}. This implies Assumption A3 (given Assumptions A1 and A2) because a circuit of size $2^k = 2^{n^\epsilon}$ can represent the distributions $\mathcal{D}_r^\mu$ for all (μ, r). Note that this is only to change the assumption to a more standard one (trapdoor permutations secure against sub-exponential circuits), and does not change the angel used.

Reducing Assumptions using Enhanced Angel. Recall that in all our assumptions, the adversary (collision finder, distinguisher or attacker on trapdoor permutation) was given access to the angel Ψ which provided collisions in $\mathcal{H}$. However, we can reduce the assumption by using an enhanced stateful angel. The modified angel Ψ' does not provide (z, x, y) all together as Ψ does; instead it simply gives z and stores (z, x, y) internally. Later the entity which queried the angel can make a *one-time* query (z, b) where b is a single bit; if $b = 0$ Ψ' answers with x, else it answers with y. After answering, Ψ' marks the internally stored tuple (z, x, y) as expired; future queries on z are not entertained. It is not hard to see that Ψ can through out be replaced by Ψ' in our protocols and assumptions. Since Ψ subsumes Ψ', the new assumptions obtained this way are weaker.

5.10.2 On Overcoming the Impossibility Results.

It is interesting to observe how this work manages to evade the impossibility results from [Can01, CF01, CKL03, Lin03b] (while still retaining composability). First, let us briefly recall the result showing that under the UC-framework, commitment functionality cannot be securely realized in the plain model (impossibility for other functionalities are similar in spirit). Suppose, for contradiction, there is indeed such a protocol between the sender C and receiver R. The proof proceeds by considering two "real world" situations A and B. In situation A, the adversary corrupts C and directs it to act transparently between the environment and R. The environment will run an honest commitment protocol (on behalf of C), and so the receiver will accept the commitment (and later a reveal). Since the protocol is secure, there exists a simulator $\mathcal{S}_A$ which can effect the same commitment and reveal in the "ideal world." In other words $\mathcal{S}_A$ can *extract* the committed bit from the protocol messages (so that it can send it to the ideal commitment functionality). Now consider situation B, where the receiver R is corrupted. The contradiction is achieved by considering an adversary $\mathcal{A}_B$ which directs R to act honestly, but sends all the messages also to an internal copy of $\mathcal{S}_A$. Now $\mathcal{S}_A$ is essentially in the same position as in situation A and can extract the committed bit, from the honest sender's commitment. However this violates the security of the protocol, leading to the contradiction.

We note that just allowing the adversary (real and ideal) access to more computational resources does not by itself stop the above proof from going through. $\mathcal{A}_B$ can still run $\mathcal{S}_A$ internally and violate the protocol's security, as it has the same computational powers as $\mathcal{S}_A$. So we would like to make sure that $\mathcal{A}_B$ cannot run $\mathcal{S}_A$, presumably because $\mathcal{S}_A$ has more computational powers than $\mathcal{A}_B$. But on the other hand, for the UC theorem to hold, the environment (and hence the adversary) should be able to internally run the simulators. In other words, the composition is known to hold only when the protocol is secure in environments which can be as powerful as the simulators. Indeed, in proving our UC theorem (Theorems 4.1 and 4.2, we use the fact that environment can subsume the simulators. So it would seem that we cannot prevent $\mathcal{A}_B$ from being able to run $\mathcal{S}_A$.

However, we do manage to get out of this apparent deadlock as we allow the power of the environment/simulator to *depend on the state of the Network*. In particular, Ψ bases its answers to queries on the set of corrupted parties. Note that above, in situations A and B, the set of corrupted parties are different. This lets us make sure that $\mathcal{A}_B$ in situation B cannot run $\mathcal{S}_A$ (which expects to be in situation B), because Ψ behaves differently in the two situations. This prevents the proof from going through. Indeed, as our results show, including the angel in the model prevents not just the proof, but also the impossibility.

5.11 Conclusion

In this chapter we have built a multi-party computation protocol without any trusted setups, for any functionality $\mathcal{F}$ which is provably secure (against static adversaries) accord-

ing to the Los Angeles Network-aware security notion, using the angel Ψ. In building this protocol, we used the new concept of an angel crucially, and overcame impossibility results which set in if the angel is not allowed. The protocol and its analysis employ tools from previous constructions in the literature, as well as new techniques.

However the protocol above, typical of this line of construction, is very complex and inefficient, as is evident from Table 5.4. In the next chapter we seek to prove security for vastly simpler protocols for (limited forms of) multi-party computation.

Notes

[1]We make a few comments on defining natural functionalities.

- Though we allow the adversary to provide input to a functionality, a protocol's useful behavior is defined with respect to the "nice" class of adversaries whose inputs are trivial. (See endnote 17 of Chapter 3.)
- An even more restrictive definition of natural functionalities — which allows secure realizability in the case of adaptive corruptions as well — restricts $\wp_{\mathcal{F}}$ to use only the inputs from the participants *and be deterministic* as well. This is not a serious handicap, since as part of the functionality specification one could define the random tape of $\wp_{\mathcal{F}}$ as the bitwise-xor of tapes provided by each party. Then as long as at least one party is honest, we can assume that $\wp_{\mathcal{F}}$ sampled its random tape. Alternately, as suggested in [CLOS02], one could require that the functionality sends all its randomness to the adversary as soon as all the parties in the session are corrupted.
- In [CLOS02], natural functionalities were termed "well-formed."
- We can securely realize *some* unnatural functionalities. Since it is sometimes convenient to specify intermediate functionalities as unnatural functionalities, we do not rule them out in the model.

[2]See Section 5.9.

[3]More formally, the assumption must be stated asymptotically: for every *infinite sequence* of pairs (μ, r) of increasing security parameter the assumption gives a corresponding *sequence* of distributions $\mathcal{D}_r^{\mu}$, with the stated indistinguishability relation (note that M stands for a circuit family here). This is necessary if we want to use the asymptotic statement, that the distributions are *indistinguishable*. But for the sake of readability, we avoid cumbersome notation and use the short hand "for every μ and r there is a distribution $\mathcal{D}_r^{\mu}$."

[4]See Endnote 3. Here again, to use the asymptotic statement — that the probability is *negligible* — the assumption must be interpreted as stated for all *infinite sequences* μ, and all PPT circuit *families*.

[5]It is not necessary that the range (and domain) of the permutation f be $\{0,1\}^k$. Any range (of size exponential in k) suffices. Indeed, the standard candidates for trapdoor permutations have more complex ranges. When the range is different from $\{0,1\}^k$, it becomes important that there is a way to efficiently sample an element in the range such that even when given the randomness used for this sampling, it remains hard to invert the element. In the literature the name *enhanced* trapdoor permutations is often used to specify this property. See [Gol04, Appendix C.1] for a discussion. In this thesis, by trapdoor permutations we always mean enhanced trapdoor permutations.

[6]We remark that we could consider $\mathcal{H}$ as a pair of functions $(\mathcal{H}_0, \mathcal{H}_1)$ where $\mathcal{H}_b$ takes one less bit of input than $\mathcal{H}$, and sets $\mathcal{H}_b(x) = \mathcal{H}(x, b)$. Then our assumptions on $\mathcal{H}$ are more akin to that on a clawfree collection than on a general collision-resistant hash function. However considering the non-standard nature of the assumptions, we refer to $\mathcal{H}$ by the generic name of hash function rather than by a name suggesting specific properties.

[7] Results in [BS05] do not directly state them in the Los Angeles Network-aware security framework, and nor is any angel specified. However it can be seen that an angel as suggested in Endnote 5 of Chapter 3 can be used to state the results of [BS05] in our framework.

[8]More formally, $\wp_C$ sends c to $\wp_R$ using the session $\Sigma_{\langle \mathcal{F}_{\text{ENC}} \rangle}$. That is $\wp_C$ passes c to the (dummy) program $\wp_{\langle \mathcal{F}_{\text{ENC}} \rangle, C}$, which passes it to $\wp_{\mathcal{F}_{\text{ENC}}}$ which in turn delivers it to the program $\wp_{\langle \mathcal{F}_{\text{ENC}} \rangle, R}$, from which $\wp_R$ receives the message. In these descriptions we shall often omit the reference to the dummy programs.

[9]BCOM is not necessarily a secure protocol for commitment functionality because it does not provide a way for a transvisor to *extract* the values committed to by a corrupted sender. In particular, depending on the specifics of the hash function $\mathcal{H}$, BCOM may be a perfectly hiding commitment, with no hope for a transvisor to extract the committed values.

[10]If $\mathcal{Z}$ and $\mathcal{A}$ are uniform, then $M_0^{\Psi^*}$ and $M_1^{\Psi^*}$ can also be uniform.

[11]In each of the expressions in this equation, the event $\mathsf{input}_{\wp_C} = 0$ refers to the particular execution in that expression. But in all four instances, $\mathbf{Pr}\left[\mathsf{input}_{\wp_C}\right]$ is the same because the execution (or simulation) is identical till beyond the point $\mathsf{input}_{\wp_C}$ is fixed.

While comparing two executions, we shall often refer to events as common to both of them. Implicit here is a *coupling* of the two executions: both the executions can be described by the same process until they diverge at some point (after the event in question); execution instances which share the same event in this prefix can be coupled together.

[12]For different values of the security parameter k, it could be either $M_0^{\Psi^*}$ or $M_1^{\Psi^*}$ which has an advantage comparable to the distinguishing probability $\mathbf{Pr}\left[\text{EXEC}_{\langle\mathcal{F}_{\widetilde{\text{COM}}}\rangle,\mathcal{Z},\mathcal{S}} = 0\right] - \mathbf{Pr}\left[\text{EXEC}_{\text{BCOM},\mathcal{Z},\mathcal{A}} = 0\right]$. However, if the latter is non-negligible, for some polynomial polythere are *infinitely many* values of k for which it is larger than $1/\text{poly}(k)$. Hence the advantage of at least one of $M_0^{\Psi^*}$ and $M_1^{\Psi^*}$ will be similarly large for infinitely many values of k.

[13]The functionality $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ also runs a simulation $\tilde{\wp}_V$, but $\mathcal{T}$ does not have access to any of the messages generated by this simulation.

[14]The simulation for the case when both parties are honest can be simplified if all the messages in the protocol are encrypted (as in BCOM, for instance). However, this is unnecessary in this protocol. Further, the simulator used in this case is the same as that for the case even when R is corrupt (and C is honest). So the proof of security is not overly complicated by avoiding encryption.

[15] Here for notational convenience, we restrict ourselves to two parties, C and R, which can be corrupted. However, the set of corrupted identities $\mathfrak{X}$ may be larger than this. To be precise, when we write, say $\mathfrak{X} = \{C\}$, what we imply is $R \notin \mathfrak{X}$.

[16]By the convention in Endnote 15 we indeed do have

$$\begin{aligned}\mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} = 0\right] = \mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} = 0 \wedge \mathfrak{X} = \{C,R\}\right] + \mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} = 0 \wedge \mathfrak{X} = \{\}\right] \\ + \mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} = 0 \wedge \mathfrak{X} = \{C\}\right] + \mathbf{Pr}\left[\text{EXEC}_{\text{COM},\mathcal{Z},\mathcal{A}} = 0 \wedge \mathfrak{X} = \{R\}\right]\end{aligned}$$

and similarly for $\mathbf{Pr}\left[\text{EXEC}_{\langle\mathcal{F}_{\text{COM}}\rangle,\mathcal{Z},\mathcal{S}} = 0\right]$.

[17] Note that C by itself is not an equivocable commitment scheme. Nevertheless, since the protocol never directly opens a value committed using C (instead, using $\mathcal{F}_{\text{ZK}}$ to convince the verifier in the reveal stage), it is possible to simulate commitment to an unknown input (when prover is honest) by committing to an arbitrary value.

[18] Though in the definition of $\mathcal{F}_{\text{CP}}^{\text{1:M}}$ we have not explicitly required a polynomial bound on the number of commitments made in one session of $\langle\mathcal{F}_{\text{CP}}^{\text{1:M}}\rangle$, since $\mathcal{Z} \in \mathcal{Z}^{\Gamma}$ is PPT, for each $\mathcal{Z}$ the number of commitments is indeed a polynomial in k (bounded by say the running time of $\mathcal{Z}$).

[19]Even if we extend the definition of $\mathring{\mathcal{A}}$ and $\mathring{\mathcal{S}}$ to allow communication with the environment (see Endnote 16 of Chapter 3), the simulators we described for general adversaries suffice. This is because these simulators behave nicely (i.e., belong to $\mathring{\mathcal{S}}$) when the adversary is in $\mathring{\mathcal{A}}$. Indeed, if the adversaries in $\mathring{\mathcal{A}}$ do not communicate with the environment, then the simulation is simpler. In that case, if any simulator can carry out the simulation, then a "dummy simulator" — which does not do anything except manage the delivery of messages from the functionality to the programs — suffices.

[20]See Endnote 5.

[21]When a party establishes an identity, the identity string μ is not chosen adversarially, but according to some distribution over $\mathcal{I}_k$, by $\wp_{\mathcal{F}_{\text{AMD}}}$.

Chapter 6

Monitored Functionalities

6.1 Introduction

The comprehensive guarantees of Network-aware security tend to require complex protocols. A natural question is if we can develop relaxed notions of Network-aware security which will help us prove some level of security for simpler protocols, at least for certain limited applications, which will nevertheless apply to a general Network setting.[1]

In this chapter we introduce a new framework of *monitored security*. It builds on the Los Angeles Network-aware security framework. The motivation for this definition stems from the simplicity of the protocols for the *semi-functionalities* $\mathcal{F}_{\widetilde{\text{COM}}}$ and $\mathcal{F}_{\widetilde{\text{ZK}}}$ introduced in Section 5.5. At the root of this simplicity is the fact that these functionalities enforce correctness and secrecy, but do not require the parties to *know* their inputs. Instead it is only guaranteed that some such input exists (this is the correctness guarantee). Further, only the secrecy guarantee is provided using the simulation paradigm. Our new security definition uses this pattern of separating correctness and secrecy guarantees.

To capture the correctness guarantee in an easy to present ideal interface, we introduce the notion of a *monitor*. A monitor exists in the ideal world, alongside an ideal functionality; it is a computationally unbounded virtual entity which watches over the ideal world execution and raises an alarm if some specific condition is violated.[2]

We demonstrate the usefulness of this framework by securely realizing a restricted version of two-party computation, called client server computation. In building a protocol for this functionality, we realize a useful tool called the monitored commit-and-prove semi-functionality. It uses protocols for $\mathcal{F}_{\widetilde{\text{COM}}}$ and $\mathcal{F}_{\widetilde{\text{ZK}}}$ from Chapter 5, whose security can also be cast in the framework of monitored functionality. (Recall that their correctness properties (binding and soundness, respectively) where guaranteed separately in Lemma 5.6 and Lemma 5.10.)

Limitations of Monitored Functionalities. There are some serious limitations to our current results for monitored Functionalities. It is not clear if the approach here can

directly yield protocols for the most general kind of functionalities. Firstly, our 2-party protocol is for a very special kind of multi-party computations only, which we term the server-client computation. (In a server-client computation, the client receives as output some function of its input and the server's input. But the server receives as output, the client's input.)

But a more serious limitation lies with the nature of security guarantee provided. Along with correctness and secrecy guarantees, one would like to have a guarantee that the server's input to the function is independent of the client's input. For this guarantee we will need to impose the condition that the client never *uses* its input previously.[3]

Despite the limitations, this new framework is a step in the direction of formalizing relaxed notions of security (relaxed, but still accounting for a general environment), which will help prove security guarantees for simpler and more efficient protocols.

Monitored Commitment: $\mathcal{MF}_{\widetilde{\text{COM}}}$

Semi-Functionality $\mathcal{F}_{\widetilde{\text{COM}}}$**:**

COMMIT PHASE

$\wp_C \leftrightarrow \wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$: Arbitrary protocol

$\wp_{\mathcal{F}_{\widetilde{\text{COM}}}} \rightarrow \wp_R, \mathcal{S}$: commit

REVEAL PHASE

$\wp_C \leftrightarrow \wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$: Arbitrary Protocol

$\wp_{\mathcal{F}_{\widetilde{\text{COM}}}} \rightarrow \wp_R, \mathcal{S}$: (reveal, b)where b is determined arbitrarily by $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$

Completeness: If C is honest, $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ never aborts.
Monitor: When $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ sends the message commit , the monitor must record one value b^* as the committed value. If C is honest this value must be the intended value for commitment. Later if $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ sends (reveal, b) but $b \neq b^*$ then the monitor notifies the receiver by raising an alarm.

Figure 6.1: Monitored Commitment Functionality $\mathcal{MF}_{\widetilde{\text{COM}}}$

6.2 Semi-Functionalities and Monitors.

A 2-party ideal functionality formulated in the Los Angeles Network-aware security model would typically interact with both the parties in an ideal way. For instance the ideal commitment functionality $\mathcal{F}_{\text{COM}}$ involves receiving a value from the *sender* secretly, and notifying the *receiver* (and adversary) of the receipt, and later on receiving a command to reveal from the sender, sending the original value to the receiver. This func-

Monitored Zero Knowledge Proof: $\mathcal{MF}_{\widetilde{\text{ZK}}}$

Parameterized by a polynomial time relation $R : \{0,1\}^{\ell_1} \times \{0,1\}^{\ell_2} \to \{0,1\}$. Common input to $\wp_P$ and $\wp_V$: $x \in \{0,1\}^{\ell_1}$.
Semi-Functionality:

$$\wp_P \leftrightarrow \wp_{\mathcal{F}_{\widetilde{\text{ZK}}}} : \text{Arbitrary protocol}$$
$$\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}} \to \wp_V, \mathcal{S} : (\mathsf{proven}, x)$$

Completeness: If P is honest, $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ sends the proven message to $\wp_V$.
Monitor: If $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ sends the message (proven, x) to the verifier, *and* there exists no $y \in \{0,1\}^{\ell_2}$ such that $R(x,y) = 1$ then the monitor notifies the verifier by raising an alarm.

Figure 6.2: Monitored Zero Knowledge Proof Functionality: $\mathcal{MF}_{\widetilde{\text{ZK}}}$

Monitored Commit and Prove: $\mathcal{MF}_{\widetilde{\text{CP}}}$

Parameterized by a polynomial time relation $R : \{0,1\}^{\ell_1} \times \{0,1\}^{\ell_2} \times \{0,1\}^{\ell_3} \to \{0,1\}$.
Semi-Functionality:

COMMIT PHASE:

$$\wp_C \leftrightarrow \wp_{\mathcal{F}_{\widetilde{\text{CP}}}} : \text{Arbitrary protocol}$$
$$\wp_{\mathcal{F}_{\widetilde{\text{CP}}}} \to \wp_R, \mathcal{S} : \mathsf{commit}$$

PROOF PHASE (CAN BE MULTIPLE TIMES):

$$\wp_C \to \wp_{\mathcal{F}_{\widetilde{\text{CP}}}} \to \wp_R, \mathcal{S} : x$$
$$\wp_C \leftrightarrow \wp_{\mathcal{F}_{\widetilde{\text{CP}}}} : \text{Arbitrary Protocol}$$
$$\wp_{\mathcal{F}_{\widetilde{\text{CP}}}} \to \wp_R, \mathcal{S} : (\mathsf{proven}, x)$$

Completeness: Interacting with an honest prover $\wp_C$, $\wp_{\mathcal{F}_{\widetilde{\text{CP}}}}$ never aborts.
Monitor: At the end of the commit phase, the monitor internally records a value w. If the $\wp_{\mathcal{F}_{\widetilde{\text{CP}}}}$ sends (proven, x) later in the proof phase then the monitor checks if there is a value y such that $R(w,x,y) = 1$. If not it raises an alarm.

Figure 6.3: Monitored Commit and Prove Functionality: $\mathcal{MF}_{\widetilde{\text{CP}}}$

tionality makes sure that the sender is bound to a value on committing (this is the "correctness guarantee") and that the value remains secret ("secrecy guarantee"). Further it ensures that the sender *knows* the value it committed to (because it had to explicitly send the value to the functionality). In defining a *semi-functionality* we would remove this last requirement, and further free the correctness requirements from the ideal functionality, somehow enforcing that requirement separately.

Client-Server Computation: $\mathcal{MF}_{\widetilde{\text{CSC}}}$

Parameterized by a function F.
Semi-Functionality:

$$\begin{aligned}
\wp_Q &\leftrightarrow \wp_{\mathcal{F}_{\text{CSC}}} : \text{Arbitrary Protocol} \\
\wp_{\mathcal{F}_{\text{CSC}}} &\rightarrow \wp_K, \mathcal{S} : \mathsf{commit} \\
\wp_K \rightarrow \wp_{\mathcal{F}_{\text{CSC}}} &\rightarrow \wp_Q, \mathcal{S} : x_K \\
\wp_Q &\leftrightarrow \wp_{\mathcal{F}_{\text{CSC}}} : \text{Arbitrary Protocol} \\
\wp_{\mathcal{F}_{\text{CSC}}} &\rightarrow \wp_K, \mathcal{S} : z \text{ where } z \text{ is determined arbitrarily by } \wp_{\mathcal{F}_{\text{CSC}}}
\end{aligned}$$

Completeness: Interacting with an honest server $\wp_Q$ with input x_Q, if the client $\wp_K$ sends it input x_K, $\wp_{\mathcal{F}_{\text{CSC}}}$ sends x_K to $\wp_Q$ and $z = F(x_Q, x_K)$ to $\wp_K$.
Monitor: At the end of the first step, the monitor internally records a value x^*. If $\wp_{\mathcal{F}_{\text{CSC}}}$ sends x_K to $\wp_Q$ and z to $\wp_K$, the monitor checks if $z = F(x^*, x_K)$. If not it raises an alarm.

Figure 6.4: Client-Server Computation Functionality: $\mathcal{MF}_{\widetilde{\text{CSC}}}$

A monitored functionality (e.g., $\mathcal{MF}_{\widetilde{\text{COM}}}$ described in Figure 6.1) consists of a semi-functionality ($\mathcal{F}_{\widetilde{\text{COM}}}$ in Figure 6.1) and some conditions on the semi-functionality. The semi-functionality is syntactically just a functionality, but it is not "ideal" enough. It is typically defined based on an arbitrary protocol. For instance the specification of $\mathcal{F}_{\widetilde{\text{COM}}}$ consists of arbitrary interaction between the server and $\mathcal{F}_{\widetilde{\text{COM}}}$ (which is unspecified in Figure 6.1, but will be later specified in such a way that binding property can be argued separately). Note that the arbitrary protocol is carried out *between the semi-functionality and a party, and not between the two parties*. This is why the semi-functionality guarantees secrecy – in the case of $\mathcal{F}_{\widetilde{\text{COM}}}$, the only message that the functionality sends to the receiver and the adversary before the reveal phase is the string commit . To complete the specification of the ideal functionality we need to also give a guarantee that the semi-functionality is *functional* (i.e., it can be used by the server to make commitments) and *correct* (i.e., it is binding on the server). These requirements are specified separately as properties that the semi-functionality needs to satisfy. (Note that we are not considering the requirements on a protocol yet; these are requirements on the functionality.) It is all these three requirements together that make up the specification of the ideal commitment functionality. We shall call such a collection of requirements a *monitored functionality*.

Ideal world of Monitored Functionality. The monitored functionality is proposed as an ideal functionality, which captures all the security properties of a given task. In Figures 6.1–6.4 we show the four monitored functionalities that we shall consider. The first two are in fact reformulations of the semi-functionalities from Section 5.5.

In Figures 6.1–6.4 the semi-functionalities are not fully specified, but allows arbitrary interaction between one of the parties (sender, prover or server) and the semi-

functionalities. Once a protocol is chosen, the semi-functionality will be specialized to suit that protocol. That is, the semi-functionality will carry out the client's part in the protocol. Note that the view of the server is unchanged if we replace the interaction with the semi-functionality by the protocol with the client. The important thing here is that irrespective of the protocol that is used, these semi-functionalities are designed to capture the secrecy requirements of the respective tasks. For instance, in commitment, the only messages sent to the client are "commit " and "(reveal, b)." Indeed, in all four semi-functionalities the messages reaching the client and the ideal world adversary are exactly those messages that the corresponding (fully) ideal functionalities ($\mathcal{F}_{\text{COM}}$, $\mathcal{F}_{\text{ZK}}$ etc.) would specify. The name semi-functionality is to emphasize that they provide only the secrecy guarantee, and correctness needs to be ensured separately. But otherwise there is nothing "semi" about them – technically these are full-fledged functionalities.

Next, we draw the reader's attention to the way the correctness requirement is specified. For convenience and concreteness, we employ the notion of *monitors*. A monitor is a conceptual device used to specify the security requirements of a functionality. If the security requirement is violated we want the monitor to alert the parties by "raising an alarm." Each session of the functionality has its own monitor. A monitor is a (computationally unbounded) function which can inspect the entire Network including all parties and functionalities (except any other monitors) and maintain its own internal state. This is in contrast to the PPT functionalities. There is only one way a monitor can affect the Network, namely, by raising an alarm.

Securely Realizing a Monitored Functionality. Having defined monitored functionalities, we would like to have real world protocols which are as secure as using these functionalities. That is, if we substitute the monitored functionality (i.e., the semi-functionality and monitor) by a protocol, no environment should be able to detect the difference (we are allowed also to replace the real world adversary $\mathcal{A}$, by an ideal world adversary $\mathcal{S}$). This involves two things: first the protocol should securely realize the semi-functionality (in the sense defined in Chapter 3). But in addition, it should be able to mimic being monitored. But clearly there are no monitors in the real world. So we require that even in the ideal world having the monitor should not be detectable to the environment. Note that this is a requirement on the functionality, and not on the protocol. However, it depends on the protocol in that the functionality is fully specified depending on the protocol.

Definition 6.1. *We say a protocol π is* monitorably as secure as *a monitored functionality $\mathcal{MF}$ with respect to $(\mathcal{Z}^\Gamma, \mathcal{A}^\Gamma, \mathcal{S}^\Gamma)$ if*

1. *π is as secure as the functionality $\mathcal{F}$ with respect to $(\mathcal{Z}^\Gamma, \mathcal{A}^\Gamma, \mathcal{S}^\Gamma)$, and*
2. *there exists a monitor satisfying the requirements specified in the monitored functionality, such that in a Network with environment $\mathcal{Z} \in \mathcal{Z}^\Gamma$ and adversary $\mathcal{S} \in \mathcal{S}^\Gamma$, the probability that the monitor raises an alarm is negligible. Note that there may be other protocols, functionalities and monitors in the Network.*

Note that the second condition needs to be met not just for a Network consisting of a single session of the protocol, but for the most general setting in which the protocol will be deployed. This is so because we do not have a composition theorem for (computationally unbounded) monitors. (i.e., a monitor may behave entirely differently when, in some part of the Network, a protocol substitution is carried out).[4] Note that there may be other monitors in the Network. But the monitors are independent of each other and the only way a monitor interferes with the Network is by raising an alarm. Hence other monitors can be ignored for analyzing the monitor of a particular session.

The results from Section 5.5 can be restated as follows.

Lemma 6.1. *Protocols* BCOM, BMCOM *and* BZK *respectively are monitorably as secure as functionalities* $\mathcal{F}_{\widetilde{\text{COM}}}$, $\mathcal{F}_{\widetilde{\text{MCOM}}}$ *and* $\mathcal{F}_{\widetilde{\text{ZK}}}$ *with respect to* $(\mathcal{Z}^{\Psi}, \mathcal{A}^{\Psi}_{\mathsf{static}}, \mathcal{S}^{\Psi})$.

PROOF: First part of the definition of being monitorably secure, namely that the protocols are as secure as the respective semi-functionalities, was shown in Lemmas 5.5, 5.8 and 5.9. The existence of the monitors directly follows from the correctness results: Lemmas 5.6, 5.7 and 5.10. □

6.3 Monitored Commit-and-Prove

Recall that the monitor for $\mathcal{MF}_{\widetilde{\text{COM}}}$ was required to record a bit b^* at the end of commitment phase as the committed but. Loosely speaking, we defined this bit as the most likely bit to be revealed. However in trying to extend this to obtain a monitor for $\mathcal{MF}_{\widetilde{\text{CP}}}$, the commit-and-prove monitored functionality, we face a problem: no bit (or string as the case may be) is ever *revealed* in this case. Nevertheless we shall see that a monitor can indeed record a committed string when $\mathcal{F}_{\widetilde{\text{CP}}}$ is based on a simple protocol BZK.

The commit phase in BZK is the same as in $\mathcal{F}_{\widetilde{\text{MCOM}}}$ (Section 5.5.2), but we describe it directly now. $\wp_V$ sends a random string r and $\wp_P$ sends $c = \overline{\mathcal{H}}(\mu_P, r, r', w)$, where r' is a random string privately chosen by $\wp_P$ and w is the string committed to. $\overline{\mathcal{H}} : \mathcal{I}_k \times \{0,1\}^{nk_1} \times \{0,1\}^{nk_2} \times \{0,1\}^n \to \{0,1\}^{n\ell}$ is a multi-bit version of $\mathcal{H}$:

$$\overline{\mathcal{H}}(\mu, (r_1 \cdots r_n), (r'_1 \cdots r'_n), (w_1 \cdots w_n)) = (\mathcal{H}(\mu, r_1, r'_1, w_1), \cdots, \mathcal{H}(\mu, r_n, r'_n, w_n)).$$

In the proof phase, there is a relation R regarding which the proof is given. Informally, $\wp_P$ wants to send a string x and then prove that there exists y such that $R(x, w, y) = 1$, where w is the string it committed to. However, $\wp_P$ does not reveal w; instead it proves that there exists r' such that $c = \overline{\mathcal{H}}(\mu_P, r, r', w)$. Both the proofs are carried out using the functionality $\mathcal{F}_{\widetilde{\text{ZK}}}$. More formally, first, $\wp_P$ and $\wp_V$ reduce the problem "$\exists w, y, r'$ such that $R'(x, r, c, w, y, r') = 1$" to a 3-coloring problem instance $G_{\mu_P}(x, r, c)$, where $R'(x, r, c, w, y, r') = 1$ if and only if $R(x, w, y) = 1$ and $\overline{\mathcal{H}}(\mu_P, r, r', w) = c$. This reduction is carried out in such a way that given a 3-coloring in G_{μ_P}, it is possible to recover

(w, y, r') as above. Then $\wp_P$ uses the semi-functionality $\mathcal{F}_{\widetilde{\text{ZK}}}$ to prove to $\wp_V$ that G_{μ_P} is 3-colorable. The protocol is given in Figure 6.5, where the above step is denoted by the following shorthand: $\wp_P \leftrightarrow \mathcal{F}_{\widetilde{\text{ZK}}} \rightarrow \wp_V : \text{ZKP}_R(x; r, c)$.

Commit and Prove Protocol: BZK

The parties are a prover P and verifier V. Input to P are $w \in \{0,1\}^n$ and (one or more) $x \in \{0,1\}^n$. k is the security parameter; k, k_1, k_2 are polynomially related to each other.
COMMIT PHASE:

$$\wp_V \rightarrow \wp_P : r \leftarrow \{0,1\}^{nk_1}$$
$$\wp_P \rightarrow \wp_V : c = \overline{\mathcal{H}}(\mu_P, r, r', w) \text{ where } r' \leftarrow \{0,1\}^{nk_2}$$

PROOF PHASE (CAN BE MULTIPLE TIMES):

$$\wp_P \rightarrow \wp_V : x$$
$$\wp_P \leftrightarrow \wp_{\mathcal{F}_{\widetilde{\text{ZK}}}} \rightarrow \wp_V : \text{ZKP}_R(x; c)$$
$$\wp_V : \text{Accept if accepted in the above protocol}$$

Figure 6.5: Protocol for the monitored functionality for Commit and Prove

We shall prove the following:

Lemma 6.2. *Protocol* BZK *is monitorably as secure as* $\mathcal{MF}_{\widetilde{\text{CP}}}$ *with respect to* $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi, \mathcal{S}^\Psi = \mathcal{A}^\Psi)$*, under Assumption A1 and Assumption A2.*

PROOF: *1.* BZK *is as secure as* $\mathcal{F}_{\widetilde{\text{CP}}}$.

We describe the transvisor $\mathcal{T} = \mathcal{T}^{\langle \mathcal{F}_{\widetilde{\text{CP}}} \rangle \rightarrow \text{BZK}}$.

Construction 6.1: Transvisor $\mathcal{T} = \mathcal{T}^{\langle \mathcal{F}_{\widetilde{\text{CP}}} \rangle \rightarrow \text{BZK}}$

Both P, V corrupt. If $\mathcal{A}$ corrupts both participants, $\mathcal{T}$ acts transparently, and the simulation is trivially perfect.

P corrupt, V honest. Just as in the case of $\mathcal{F}_{\widetilde{\text{COM}}}$ and $\mathcal{F}_{\widetilde{\text{ZK}}}$, a simple transvisor in which $\tilde{\wp}_V$ merely acts as a front to $\wp_{\mathcal{F}_{\widetilde{\text{CP}}}}$ yields a perfect simulation.

P honest. When the prover is not corrupted, the only protocol messages $\mathcal{A}$ can see are the initial commitment phase messages r and c and the $(\mathsf{proven}, G(x, r, c))$ message from $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ for each proof phase. The only non-trivial task for the simulator

is, when V is not corrupt, to produce the commitment text c, without having access to w. For this $\tilde{\wp}_P$ sets $\hat{c} = \overline{\mathcal{H}}(\mu_P, r, r', w')$ where $w' = 0^n$ and $r' \leftarrow \{0,1\}^{k_2}$. Then $\hat{c}$ is sent to $\wp_V$ instead of c.

Suppose in some Network, $\mathcal{Z}$ could distinguish between the actual execution of BZK and the simulation: i.e., $\text{EXEC}_{\text{BZK},\mathcal{Z},\mathcal{A}}$ and $\text{EXEC}_{\langle \mathcal{F}_{\widetilde{\text{CP}}} \rangle,\mathcal{Z},\mathcal{S}}$ differed non-negligibly. Then we shall build a distinguisher M^{Ψ^*} to violate Assumption A1. Note that Assumption A1 implies that the distributions of commitments to 0 and to 1 are indistinguishable to a PPT distinguisher (with access to Ψ^*) M^{Ψ^*}: that is, for all $\mu \in \mathcal{I}_k$, $r \in \{0,1\}^{k_1}$

$$\{z | x \leftarrow \{0,1\}^{k_2}, z = \mathcal{H}(\mu, r, x, 0)\} \approx \{z | y \leftarrow \{0,1\}^{k_2}, z = \mathcal{H}(\mu, r, y, 1)\},$$

because both the distributions are indistinguishable from $\{z | (x, y, z) \leftarrow \mathcal{D}_r^\mu\}$. We build M^{Ψ^*} so that it can distinguish between the above two distributions.

Construction 6.2: M^{Ψ^*} to distinguish between $\mathcal{H}(\mu, r, x, 0)$ and $\mathcal{H}(\mu, r, x, 1)$.

M^{Ψ^*} receives as input $z = \mathcal{H}(\mu, r, x, b)$, and tries to distinguish between $b = 0$ and $b = 1$ as follows. M^{Ψ^*} simulates the Network execution $\text{EXEC}_{\text{BZK},\mathcal{Z},\mathcal{A}}$ internally. If either P or V is corrupt, output a random bit and bail out. Otherwise, carry out the execution, but with the following modification: instead of sending out c as prescribed, $\wp_P$ sends out c' as constructed below.

- If $\mathcal{Z}$ provides an input to $\wp_P$, call it w. Set

$$c = (\mathcal{H}(\mu, r_1, r'_1, w_1), \cdots, \mathcal{H}(\mu, r_n, r'_n, w_n))$$

$$d = (\mathcal{H}(\mu, r_1, r'_1, 0), \cdots, \mathcal{H}(\mu, r_n, r'_n, 0)).$$

- If w_i, the i-th bit of w, is 0, then let

$$c^i = (c_1, \ldots, c_{i-1}, d_i, \ldots, d_n)$$

  where $c_j = \mathcal{H}(\mu, r_j, r'_j, w_j)$ and $d_j = \mathcal{H}(\mu, r_j, r'_j, 0)$. That is, c^i is constructed by taking the first $i-1$ components of c and remaining components from d (which is independent of w).

- If $w_i = 1$, the let

$$c^i = (c_1, \ldots, c_{i-1}, z, d_{i+1}, \ldots, d_n,$$

  That is, c^i is the same as defined for the case $w_i = 0$, but the i-th component is replaced by z.

- Pick a random index $i \leftarrow \{1, \ldots, n\}$, and let $c' = c^i$.

Then M^{Ψ^*} outputs what $\mathcal{Z}$ outputs in this simulated Network.

We claim that M^{Ψ^*} has a non-negligible advantage in distinguishing between $z = \mathcal{H}(\mu, r, x, 0)$ and $z = \mathcal{H}(\mu, r, x, 1)$, if $\text{EXEC}_{\text{BZK},\mathcal{Z},\mathcal{A}}$ and $\text{EXEC}^{\langle \mathcal{F}_{\widetilde{\text{CP}}} \rangle, \mathcal{Z}, \mathcal{S}}$ differed non-negligibly. To see this, first we consider a sequence of hybrid executions bridging these two, obtained by modifying $\tilde{\wp}_P$ in $\text{EXEC}^{\langle \mathcal{F}_{\widetilde{\text{CP}}} \rangle, \mathcal{Z}, \mathcal{S}}$ as follows: $\tilde{\wp}_P$ is given access to w that $\mathcal{Z}$ passes on to $\wp_P$ as input; instead of 0^n to construct $\hat{c}$, $\tilde{\wp}_P$ uses w^i obtained by zeroing out all but the first i bits of w. Let EXEC_i denote the output of $\mathcal{Z}$ in the i-th hybrid execution. Then, we complete the argument by observing that

$$
\begin{aligned}
&|\mathbf{Pr}\left[\text{EXEC}_{\text{BZK},\mathcal{Z},\mathcal{A}} = 0\right] - \mathbf{Pr}\left[\text{EXEC}^{\langle \mathcal{F}_{\widetilde{\text{CP}}} \rangle, \mathcal{Z}, \mathcal{S}} = 0\right]| \\
&\qquad = |\mathbf{Pr}\left[\text{EXEC}_n = 0\right] - \mathbf{Pr}\left[\text{EXEC}_0 = 0\right]| \\
&\qquad = |\sum_{i=1}^{n} (\mathbf{Pr}\left[\text{EXEC}_i = 0\right] - \mathbf{Pr}\left[\text{EXEC}_{i-1} = 0\right])| \\
&\qquad = n|\,\mathbf{Pr}_{z=\mathcal{H}(\mu,r,x,1)}\left[M^{\Psi^*}(z) = 0\right] - \mathbf{Pr}_{z=\mathcal{H}(\mu,r,x,0)}\left[M^{\Psi^*}(z) = 0\right]|
\end{aligned}
$$

where the last equality holds because when $z = \mathcal{H}(\mu, r, x, 0)$, conditioned on M^{Ψ^*} picking the index i (which happens with probability $1/n$), $M^{\Psi^*}(z)$ is identical to EXEC_{i-1}, and hence

$$\mathbf{Pr}_{z=\mathcal{H}(\mu,r,x,0)}\left[M^{\Psi^*}(z) = 0\right] = 1/n \sum_{i=1}^{n} \mathbf{Pr}\left[\text{EXEC}_{i-1} = 0\right].$$

Similarly

$$\mathbf{Pr}_{z=\mathcal{H}(\mu,r,x,1)}\left[M^{\Psi^*}(z) = 0\right] = 1/n \sum_{i=1}^{n} \mathbf{Pr}\left[\text{EXEC}_{i} = 0\right].$$

2. *There is a monitor satisfying the requirements specified by $\mathcal{MF}_{\widetilde{\text{CP}}}$, such that the probability of the monitor raising an alarm (before the environment halts) is negligible.*

We restrict ourselves to the case when $\mathcal{MF}_{\widetilde{\text{CP}}}$ allows only one proof phase per session. It is possible to extend it to multiple proofs, but the details become lengthy and tedious.

First we describe how a value w^* is recorded by $\mathcal{M}$. For this we shall first describe an "extractor" machine $M^{\Psi_{\setminus P}}$.

Construction 6.3: Extractor $M^{\Psi_{\setminus P}}$.

$M^{\Psi_{\setminus P}}$ simulates the entire Network (except any monitors) internally, starting at the point where the session of interest running our Commit-and-Prove protocol starts (this start state is given to $M^{\Psi_{\setminus P}}$ as input). It uses access to $\Psi_{\setminus P}$ to simulate calls to Ψ (note that P is honest in this Network, and hence $\Psi_{\setminus P}$ subsumes Ψ). $M^{\Psi_{\setminus P}}$ is a PPT (oracle) machine because $\mathcal{Z}$ has a $\text{poly}(k)$ time bound, and other (non-angel) entities — which are all invoked directly or indirectly by $\mathcal{Z}$— are bounded by polynomials in k and their input size. $M^{\Psi_{\setminus P}}$ runs the Network until the proof

phase of the session started, and the prover makes the commitment step. At this point $M^{\Psi\setminus P}$ "forks": it clones the Network and runs the two copies independent of each other. If in both the copies the proof is accepted by the verifier, $M^{\Psi\setminus P}$ checks if the n-bit queries made by the verifier in $\text{ZKP}_R(x; r, c)$ are identical or not. If they are not identical this lets $M^{\Psi\setminus P}$ extract a 3-coloring for G (assuming the monitors for the $\mathcal{F}_{\widetilde{\text{COM}}}$s do not raise any alarm). Then $M^{\Psi\setminus P}$ derives a witness (w, r', y) from this 3-coloring, and outputs it. Else $M^{\Psi\setminus P}$ outputs $\perp$.

Now we use $M^{\Psi\setminus P}$ to describe the monitor $\mathcal{M}$.

Construction 6.4: Monitor $\mathcal{M} = \mathcal{M}_{\mathcal{F}_{\widetilde{\text{CP}}}}$.

When $\mathcal{F}_{\widetilde{\text{CP}}}$ sends commit to $\wp_V$, for each w $\mathcal{M}$ checks the probability of $M^{\Psi\setminus P}$ outputting w, and records the one with the highest such probability, say w^*. Later if $\mathcal{F}_{\widetilde{\text{CP}}}$ sends (proven, x) for some x such that for no y $R(w^*, x, y)$ holds, then it raises an alarm. Also, for purposes of analysis, when the prover executes the commitment protocol (semi-functionality) as part of the zero-knowledge proof protocols, $\mathcal{M}$ starts the monitors for $\mathcal{F}_{\widetilde{\text{COM}}}$ as sub-monitors. The monitors will also be run when the extractor $M^{\Psi\setminus P}$ runs. If any of these sub-monitors raises an alarm, then too $\mathcal{M}$ will raise an alarm.

Clearly, by design, $\mathcal{M}$ satisfies the requirements of the functionality. We go on to prove that the probability that $\mathcal{M}$ raises an alarm (which event we denote by alarm) is negligible. In the rest of the proof, we condition on the event that none of the sub-monitors for $\mathcal{F}_{\widetilde{\text{MCOM}}}$ raise an alarm. Since we have already shown that this is an event of negligible probability (and since only polynomially many such sub-monitors are run), this will not change our conclusions.

Now, consider the point at which $M^{\Psi\setminus P}$ forks the Network. Let p_w be the probability that $M^{\Psi\setminus P}$ outputs w starting at (conditioned on) this point. Let q be the probability that $\wp_V$ accepts the proof $\text{ZKP}_R(x; c)$, but $\nexists(y, r')R'(w^*, x, r, r', c, y) = 1$. Note that $\mathbf{Pr}[\text{alarm}] = \mathbf{E}[q]$, where the expectation is over the distribution on the state of the Network at the point at which $M^{\Psi\setminus P}$ forks.

Since we assume that the sub-monitors do not raise alarm, $M^{\Psi\setminus P}$ outputs some w if the two copies it runs both accept the proof, and in the second copy the verifier sends a query different from the one in the first copy. So, $\sum_{w\neq w^*} p_w \geq q(q - 2^{-n})$. Then,

$$\begin{aligned}
\mathbf{Pr}\left[M_*^{\Psi\setminus P} \text{ outputs } w \neq w^*\right] &\geq \mathbf{E}\left[q(q - 2^{-n}\right] \\
&\geq \mathbf{E}[q]^2 - 2^{-n}\,\mathbf{E}[q] \\
&\geq \frac{1}{2}\,\mathbf{E}[q]^2 \qquad\qquad \text{if } \mathbf{E}[q] \geq 2^{-n+1}.
\end{aligned}$$

If the assumption in the last line above does not hold, we would be done, because $\mathbf{Pr}[\mathsf{alarm}] = \mathbf{E}[q]$. So we make that assumption and proceed.

Now we shall demonstrate a (non-uniform) PPT machine $M_*^{\Psi\setminus P}$ which accepts $r \leftarrow \{0,1\}^k$ and outputs (x, y) such that $\mathcal{H}(\mu_P, r, x, 0) = \mathcal{H}(\mu_P, r, y, 1)$, with a probability polynomially related to the probability of the monitor raising an alarm.

Construction 6.5: $M_*^{\Psi\setminus P}$ to find (x, y) such that $\mathcal{H}(\mu_P, r, x, 0) = \mathcal{H}(\mu_P, r, y, 1)$.

$M_*^{\Psi\setminus P}(r)$ starts $M^{\Psi\setminus P}$ and runs the commit phase by sending r. It forks $M^{\Psi\setminus P}$ after the commitment from $\wp_P$ arrives. Then it runs the two independent copies of $M^{\Psi\setminus P}$ (which involves forking the Network again), and checks if they output different values (w_1, r_1') and (w_2, r_2'), with $w_1 \neq w_2$. If so, $M_*^{\Psi\setminus P}$ derives a collision to the hash function from some bit at which w_1 and w_2 differ, and outputs the corresponding portions of r_1', r_2'.

We say that $M_*^{\Psi\setminus P}$ succeeds if it gets (w_1, w_2), such that $w_1 \neq w_2$ from two runs of $M^{\Psi\setminus P}$. Then,

$$\begin{aligned}
\mathbf{Pr}\left[M_*^{\Psi\setminus P} \text{ succeeds}\right] &\geq \sum_{w'} p_{w'} \sum_{w\neq w'} p_w \\
&\geq \sum_{w'} p_{w'} \sum_{w\neq w^*} p_w && \text{because for all } w',\ p_{w^*} \geq p_{w'} \\
&= (\sum_{w'} p_{w'})(\sum_{w\neq w^*} p_w) \geq (\sum_{w\neq w^*} p_w)^2 \\
&\geq \frac{1}{4}\mathbf{E}[q]^4 = \frac{1}{4}\mathbf{Pr}[\mathsf{alarm}]^4.
\end{aligned}$$

Putting it all together we have that $\mathbf{Pr}[\mathsf{alarm}] \leq (4\,\mathbf{Pr}\left[M_*^{\Psi\setminus P} \text{ finds a collision}\right])^{\frac{1}{4}}$, which is negligible by assumption on $\mathcal{H}$. □

6.4 Client-Server Computation

As we have seen above, theoretically interesting cryptographic tools like commitment and zero-knowledge proofs can be securely realized in this framework, relatively efficiently (compared to the protocols in Chapter 5). The reason for this is that our security requirements are much more relaxed. However this raises the question if these weakened versions of the above tools are useful to achieve security for practically interesting tasks. In this section we make some progress towards making this framework usable for multi-party computation problems.

We restrict ourselves to 2-party computations of a very specific kind, as described in Figure 6.4. Note that the client does not keep any secrets from the server Q. But the server

must *commit* to its inputs (and the monitor shall record the committed input) before the client sends its inputs. First, we shall give a protocol for this monitored functionality, before discussing some extensions possible and some limitations.

Client-Server Computation Protocol: CSC

The protocol is parameterized by a function F.

$$\wp_Q \leftrightarrow \wp_{\mathcal{F}_{\widetilde{\text{CP}}}} \rightarrow \wp_K : \text{COMMIT-PHASE}(x_Q)$$
$$\wp_K \rightarrow \wp_Q : x_K$$
$$\wp_Q \rightarrow \wp_K : z = F(x_Q, x_K)$$
$$\wp_Q \leftrightarrow \wp_{\mathcal{F}_{\widetilde{\text{CP}}}} \rightarrow \wp_K : \text{PROOF-PHASE}(z == F(\text{COMMITTED-VALUE}, x_K))$$

Figure 6.6: Protocol for the monitored functionality for Client-Server Computation

Theorem 6.1. *The protocol* CSC *is monitorably as secure as* $\mathcal{MF}_{\widetilde{\text{CSC}}}$ *with respect to* $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\mathsf{static}}, \mathcal{S}^\Psi_{\mathsf{static}})$.

PROOF: The proof directly follows from the security of $\mathcal{MF}_{\widetilde{\text{CP}}}$.
1. There is a monitor satisfying the requirements specified by $\mathcal{MF}_{\widetilde{\text{CSC}}}$*, such that the probability of the monitor raising an alarm is negligible.*

Construction 6.6: Monitor $\mathcal{M} = \mathcal{M}_{\mathcal{F}_{\text{CSC}}}$.

$\mathcal{M}$ internally starts the monitor $\mathcal{M}_{\mathcal{F}_{\widetilde{\text{CP}}}}$ (described in Construction 6.4); If the protocol proceeds beyond the first step, $\mathcal{M}_{\mathcal{F}_{\widetilde{\text{CP}}}}$ would record a value x^* internally as the committed value. $\mathcal{M}$ will copy that value and record it as the input of Q. Later if the monitor for $\mathcal{MF}_{\widetilde{\text{CP}}}$ raises an alarm, $\mathcal{M}$ will raise an alarm.

If $\mathcal{F}_{\text{CSC}}$ sends the value z to K, then $\mathcal{F}_{\widetilde{\text{CP}}}$ must have returned $(\mathsf{proven}, z = F(x^*, x_K))$. So if the monitor for $\mathcal{MF}_{\widetilde{\text{CP}}}$ does not raise an alarm, it means indeed $z = F(x^*, x_K)$ and $\mathcal{M}$ need not raise any alarm either. Thus $\mathcal{M}$ does satisfy the requirements specified by $\mathcal{MF}_{\widetilde{\text{CSC}}}$. Further the probability that $\mathcal{M}$ raises an alarm is the same as that the $\mathcal{M}_{\mathcal{F}_{\widetilde{\text{CP}}}}$ raises an alarm. By earlier analysis, this is indeed negligible.
2. CSC *is as secure as* $\mathcal{F}_{\text{CSC}}$ *with respect to* $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\mathsf{static}}, \mathcal{S}^\Psi_{\mathsf{static}})$.

We describe a PPT transvisor $\mathcal{T} = \mathcal{T}^{\langle \mathcal{F}_{\text{CSC}} \rangle \rightarrow \text{CSC}}$ (with access to Ψ).

Construction 6.7: Transvisor $\mathcal{T} = \mathcal{T}^{\langle \mathcal{F}_{\text{CSC}} \rangle \rightarrow \text{CSC}}$

Both Q, K corrupt. If $\mathcal{A}$ corrupts both participants, $\mathcal{T}$ acts transparently, and the simulation is trivially perfect.

Q corrupt, K honest. As in the case of all the monitored functionalities introduced in this work, thanks to the way the semi-functionality is designed, a simple transvisor in which $\tilde{\wp}_K$ merely acts as a front to $\wp_{\mathcal{F}_{\text{CSC}}}$ yields a perfect simulation.

Q honest. Note that in this case the view of $\mathcal{A}$ consists of the following: the message commit from $\wp_{\widetilde{\mathcal{F}_{\text{CP}}}}$, the value x_K, the value $z = F(x_Q, x_K)$ and the message $(\mathsf{proven}, z == F(\text{COMMITTED-VALUE}, x_K))$ from $\wp_{\widetilde{\mathcal{F}_{\text{CP}}}}$. On the other hand $\mathcal{T}$ receives x_K and $z == F(\text{COMMITTED-VALUE}, x_K)$ from $\mathcal{F}_{\text{CSC}}$, which is enough to carry out a perfect simulation.

It readily follows that the simulation is perfect. □

We record our final result for client-server computation as the following corollary.

Corollary 6.1. *There is a protocol which does not use any ideal functionalities, and which is monitorably as secure as $\mathcal{MF}_{\widetilde{\text{CSC}}}$ with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\mathsf{static}}, \mathcal{S}^\Psi_{\mathsf{static}})$, under assumptions A1, A2 and A3.*

PROOF: Theorem 6.1 directly implies this corollary, except that the protocol CSC uses ideal functionalities. Using the composition theorem 4.1 to compose CSC with protocols BZK, BZK, BMCOM, ENC and AMD (employing Lemmas 5.9, 5.8, 5.5, 5.17) we obtain a protocol which is secure as CSC with respect to $(\mathcal{Z}^\Psi, \mathcal{A}^\Psi_{\mathsf{static}}, \mathcal{S}^\Psi_{\mathsf{static}})$, and does not use any ideal functionalities. By transitivity of the "as secure as" relation (Lemma 3.1) we see that this protocol satisfies the first condition of being monitorably as secure as $\mathcal{MF}_{\widetilde{\text{CSC}}}$. □

6.5 Independence of Inputs and Locked States

6.5.1 Problem with $\mathcal{MF}_{\widetilde{\text{CSC}}}$

There are some serious limitations to the functionality $\mathcal{MF}_{\widetilde{\text{CSC}}}$. Clearly, the set of functions that can be computed are limited (namely, only client-server computations). But more seriously, the guarantee given by the monitor is not satisfactory. In particular, there is no guarantee of *"independence" of inputs*. Though the monitor records a value for the server's input prior to the client sending out its input, the value recorded is allowed to be *dependent on the entire Network*, and in particular on the input of the client![5]

6.5.2 The Solution: Restricting the Monitors

Below we suggest a way to address this problem. We show that if the clients keep their private inputs totally unused until the point of commitment by the server (but may use them immediately afterwards), then the monitor can be required to record a value independent of their private inputs. As it turns out, the protocols are not altered, but some

restrictions are imposed on the monitor, and some parts of the proof become significantly more involved.

Locked State. We introduce a notion that some part of the state of the Network can be kept "locked." This part, which we shall call the *locked state*, cannot be used in the Network (until it is unlocked). The requirement on the monitor is that it does not have access to the part of the Network state if it is locked at the point the monitor is required to record a value; it will have to record a value based on the rest of the Network, which we shall call the *open state*.

Technically, the locked state corresponding to a protocol execution is defined at the beginning of that execution: it is the maximal part of the Network state, not including any of the adversary's state, such that the *distribution* of the rest of the Network state at the recording point is *independent* of it. Note that the independence requirement implies in particular that the probability of unlocking the state before the commitment phase is over is zero (unless the locked state is completely predictable *a priori* from the open state).

We do allow the locked state to evolve, as long as the independence is maintained (in particular, no information should pass between the locked state and the open state). Further, for full generality, we allow the locked state to be randomized: i.e., its value is a random variable. However, we shall require that this random variable is efficiently sampleable (which is implied by the assumption that the non-adversarial part of the Network is PPT). In particular all the "future" randomness, i.e., randomness which is sampled after the monitor finishes recording, can be considered part of the locked state.[6]

Use of Locked States. As indicated earlier, the reason we allow the notion of a locked state in our framework has to do with the meaningfulness of the two-party computation scenario. With the modification sketched above in place, we can allow the client to keep its input locked, and then even the monitor does not get to see it, before recording the other party's input.

However, note that to keep an input locked, it can never be used in the Network at all.[7] This is because the monitor is computationally unbounded. Note that this is related to the problem of malleability: if it was used in the Network previously, somehow that can be mauled and used to make a commitment related to it. However, interestingly we do avoid the problem of malleability while *opening* a commitment: the locked state is allowed to be unlocked *before the commitment is opened.*

6.5.3 Restricted Monitors

We now show that in the monitored functionalities we use, the monitors can be required not to inspect the locked state of the Network. Surprisingly, this complicates the construction of the monitor and the proofs considerably.

Lemma 6.3. *For any polynomials (in the security parameter k) τ and Π, under Assumption A2, there is a monitor satisfying the requirements specified by $\mathcal{MF}_{\widetilde{\text{COM}}}$ which does not inspect the locked state of the Network, such that the probability of the monitor raising an alarm within time τ is less than $1/\Pi$.*

PROOF OVERVIEW: When $\mathcal{F}_{\widetilde{\text{COM}}}$ sends the commit message the monitor $\mathfrak{M}_\Pi$ must record a bit b^* internally. First, we sketch how $\mathfrak{M}_\Pi$ does this. As before, the basic idea is for the monitor to look ahead in the Network, and record the more likely bit that the sender will ever reveal; if the sender can reveal to both bits with significant probability, a reduction can be used to obtain a circuit for finding collisions in the hash function. But note that here $\mathfrak{M}_\Pi$ does not know the value of the locked state, and so it cannot calculate the bit as above. However, we can show that for no two values for the locked state, can the sender feasibly reveal the commitment in different ways. Intuitively then, the monitor can use an arbitrary value for the locked state and use it to carry out the calculation. However, there are a couple of problems with this. Firstly, revealing can depend not only on the open state of the Network at the end of commitment, but also on the locked state, as it might be unlocked after the commit phase is over. In particular, for certain values of the locked state (and open state), the sender might never complete the reveal phase. So using a single value of the locked state will not suffice. The second problem is that while $\mathfrak{M}_\Pi$ is computationally unbounded, the reduction to finding collision should use a polynomial sized circuit. This circuit will need to be given the value(s) of the locked state with which it will emulate the Network. Further, the circuit will obtain as input the random challenge in the commitment. Thus, the value(s) of the locked state that it obtains should be defined prior to seeing the random challenge.

Nevertheless, we show how to define *polynomially many values for the locked state of the Network, based only on the open state of the Network*, and obtain a bit b^* using just these values. To show that the probability of $\mathfrak{M}_\Pi$ raising an alarm within time τ is less than $1/\Pi$, we show that otherwise we can give a polynomial sized circuit (with the above mentioned values of the locked states built-in) which can find a collision in our hash function for a random challenge with significant possibility.

The full proof is given below. ◁

PROOF: When $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ sends the commit message the monitor $\mathfrak{M}_\Pi$ must record a bit b^* internally, by inspecting just the open state of the Network, so that it will not have to raise an alarm with in time τ, except with probability at most $1/\Pi$. First, we describe how $\mathfrak{M}_\Pi$ decides on b^*, and then argue why it satisfies the requirement.

We use the following notation: the open state of the Network (defined for the particular instance of $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$ that we are considering), at the point commit message is sent by $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$, is denoted by the random variable $\mathcal{L}$ and a generic value of this random variable will be denoted by λ. The value of the locked state of the Network at that point is denoted

by the variable δ^*. Since we allow the locked state to be randomized, by the "value" of the locked state, we mean a distribution. Recall that the monitor does not know δ^*. But it can consider all possible values (distributions) of the locked state (this is well-defined, since the distributions should be sampleable by a PPT circuit, and the monitor can construct all such circuits). We shall denote a generic value of the locked state (represented by the efficient sampler which produces this distribution) by δ.

All our probabilities are conditioned on any initial (open) state of the Network. Let p_λ denote the probability that $\mathcal{L}$ takes value λ. Note that by definition of the locked states, p_λ is the same independent of δ^*. In other words, if we replace δ^* by some other value δ, the probability p_λ does not change for any λ. Denote by $p_{\text{rev:0}}^{(\lambda,\delta)}$ the probability, conditioned on δ being the locked state and λ being the open state at the point of commitment, that $\wp_{\widetilde{\mathcal{F}_{\text{COM}}}}$ later sends 0 as the revealed bit to the receiver (within time τ from the point of commitment, and before the receiver is corrupted). Similarly define $p_{\text{rev:1}}^{(\lambda,\delta)}$. Let $p_{\text{rev}}^{(\lambda,\delta)} = p_{\text{rev:0}}^{(\lambda,\delta)} + p_{\text{rev:1}}^{(\lambda,\delta)}$ denote the probability that $\wp_{\widetilde{\mathcal{F}_{\text{COM}}}}$ sends any revealed bit at all to the receiver (within time τ, and before receiver is corrupted), conditioned on δ and λ being the locked and open states.

First, $\mathfrak{M}_\Pi$ chooses a set Δ of values for locked state and uses them to "cover" a set Λ of values for open states, as follows. Start with an empty set Δ, and also an empty set Λ. Then, it checks if there exists a locked state value $\delta \notin \Delta$ such that

$$\sum_{\lambda \notin \Lambda} p_\lambda p_{\text{rev}}^{(\lambda,\delta)} \geq 1/\varpi \tag{6.1}$$

where ϖ is a polynomial that we shall shortly relate to Π, If such an δ exists, it adds it to Δ. Further δ will be used to cover all (uncovered) values λ for which

$$p_{\text{rev}}^{(\lambda,\delta)} \geq 1/(2\varpi). \tag{6.2}$$

Then all such λ are added to Λ. Note that then we will have

$$\sum_{\lambda \text{ covered by } \delta} p_\lambda \geq 1/(2\varpi). \tag{6.3}$$

The whole procedure is repeated, until there are no more δ satisfying (6.1). From (6.3), each new δ added to Δ covers an additional $1/(2\varpi)$ fraction (weight) of λ. So the above procedure terminates after adding at most 2ϖ elements to Δ. Also, we note that for every δ, we will have

$$\sum_{\lambda \notin \Lambda} p_\lambda p_{\text{rev}}^{(\lambda,\delta)} \leq 1/\varpi. \tag{6.4}$$

(This is true for $\delta \notin \Delta$ as otherwise it would have been added. This is true for $\delta \in \Delta$ as well, because for every $\lambda \notin \Lambda$ we have $p_{\text{rev}}^{(\lambda,\delta)} \leq 1/(2\varpi)$.)

$\mathfrak{M}_\Pi$ uses the above set Δ to decide what bit to record, depending only on the open state at the point of commitment. If this state is λ, $\mathfrak{M}_\Pi$ checks if it is covered by any $\delta \in \Delta$. If not, it records an arbitrary bit. But if it is covered by some $\delta \in \Delta$, then if $p^{(\lambda,\delta)}_{\text{rev:0}} \geq p^{(\lambda,\delta)}_{\text{rev:1}}$ it records the bit 0; else it records 1. We shall denote the bit so defined, by b^*_λ.

Now we need to prove that the probability that $\wp_{\widetilde{\mathcal{F}_{\text{COM}}}}$ reveals a bit other than b^*_λ is negligible, no matter what the actual locked state δ^* is. The high level plan is to show that if for some locked state value δ^*, if this is not the case, then we can demonstrate a PPT circuit $M^{\Psi\setminus R}$, which can find collisions in our hash function. $M^{\Psi\setminus R}$ will emulate the Network until the commitment point, but using an externally received random string as the challenge sent by $\wp_{\widetilde{\mathcal{F}_{\text{COM}}}}$ in the commitment phase. Then it will fork into different execution threads using δ^* as well as $\delta \in \Delta$ as the locked state values. It will be the case that there will be some $\delta \in \Delta$ which covers a significant fraction of Λ such that the execution thread with δ^* has a significant probability of making $\wp_{\widetilde{\mathcal{F}_{\text{COM}}}}$ reveal $b \neq b^*_\lambda$, *and* the execution thread with δ has a significant probability of making $\wp_{\widetilde{\mathcal{F}_{\text{COM}}}}$ reveal b^*_λ. This will allow $M^{\Psi\setminus R}$ to have a significant probability in outputting a collision, corresponding to the random challenge it received externally. Details follow.

Construction 6.8: PPT machine $M^{\Psi\setminus R}$ for finding collisions in $\mathcal{H}$.

$M^{\Psi\setminus R}$ receives (as non-uniform input) the initial open state of the Network, and the set Δ of at most 2ϖ values for the locked state. It simulates (the open state of) the Network (if the receiver is not corrupted[8]), with the following modifications:

- If the Network reaches the commitment phase, then instead of independently choosing a random challenge $r \leftarrow \{0,1\}^{k_1}$ on behalf of $\wp_{\widetilde{\mathcal{F}_{\text{COM}}}}$, it receives an r from outside and passes it on as the challenge from $\wp_{\widetilde{\mathcal{F}_{\text{COM}}}}$.
- Then if it receives a commitment c from the sender, it makes multiple copies of the Network to be run independently from that point onwards. $M^{\Psi\setminus R}$ makes one copy for each $\delta \in \Delta$ and one for the actual value of the locked state δ^*. (Note that though $\mathfrak{M}_\Pi$ cannot see δ^*, we can define $M^{\Psi\setminus R}$ which depends on δ^*.) $M^{\Psi\setminus R}$ runs these copies as independent threads (terminating a thread if the receiver is corrupted in it).
- If in any two such copies, c is revealed differently, then $M^{\Psi\setminus R}$ copies the reveal messages obtained by $\wp_{\widetilde{\mathcal{F}_{\text{COM}}}}$, which contain a collision to the hash function, using the random challenge r, and outputs it.

We would like to show that the probability with which $M^{\Psi\setminus R}$ succeeds in outputting a collision is polynomially related to that of the probability of $\mathfrak{M}_\Pi$ raising an alarm (if the latter is more than $1/\Pi$). The former probability is lower bounded by $\sum_\lambda p_\lambda p^{(\lambda,\delta)}_{\text{rev:}b^*_\lambda} p^{(\lambda,\delta^*)}_{\text{rev:}\bar{b}_\lambda}$ for all $\delta \in \Delta$, where $\bar{b}_\lambda = 1 - b^*_\lambda$.

Suppose the probability of $\mathfrak{M}_\Pi$ raising the alarm is at least $2/\varpi$ (we shall set $\varpi = 2\Pi$). Then, $\sum_\lambda p_\lambda p^{(\lambda,\delta^*)}_{\text{rev}:\bar{b}_\lambda} \geq 2/\varpi$. We shall argue that there exists $\delta \in \Delta$ such that the above sum puts a significant weight on λ covered by δ. Specifically, we claim

$$\sum_{\lambda \text{ covered by } \delta} p_\lambda p^{(\lambda,\delta^*)}_{\text{rev}:\bar{b}_\lambda} \geq 1/(2\varpi^2). \tag{6.5}$$

To see this, note that using (6.4), and that $p^{(\lambda,\delta^*)}_{\text{rev}:\bar{b}_\lambda} \leq p^{(\lambda,\delta^*)}_{\text{rev}}$, we get $\sum_{\lambda\in\Lambda} p_\lambda p^{(\lambda,\delta^*)}_{\text{rev}:\bar{b}_\lambda} \geq 2/\varpi - 1/\varpi = 1/\varpi$. Now, all $\lambda \in \Lambda$ are covered by at most 2ϖ $\delta \in \Delta$. So indeed there exists an $\delta \in \Delta$ such that $\sum_{\lambda \text{ covered by } \delta} p_\lambda p^{(\lambda,\delta^*)}_{\text{rev}:\bar{b}_\lambda} \geq (1/\varpi)/(2\varpi)$.

Now we use the earlier lowerbound on the collision probability and conclude that

$$\begin{aligned}
\mathbf{Pr}\left[M^{\Psi\setminus R} \text{ outputs a collision}\right] &\geq \sum_\lambda p_\lambda p^{(\lambda,\delta)}_{\text{rev}:b^*_\lambda} p^{(\lambda,\delta^*)}_{\text{rev}:\bar{b}_\lambda} \\
&\geq \sum_{\lambda \text{ covered by } \delta} p_\lambda p^{(\lambda,\delta)}_{\text{rev}:b^*_\lambda} p^{(\lambda,\delta^*)}_{\text{rev}:\bar{b}_\lambda} \\
&\geq \sum_{\lambda \text{ covered by } \delta} p_\lambda (\frac{1}{2} p^{(\lambda,\delta)}_{\text{rev}}) p^{(\lambda,\delta^*)}_{\text{rev}:\bar{b}_\lambda} \qquad \text{by choice of } b^*_\lambda \\
&\geq \sum_{\lambda \text{ covered by } \delta} p_\lambda \frac{1}{4\varpi} p^{(\lambda,\delta^*)}_{\text{rev}:\bar{b}_\lambda} \qquad \text{by (6.2)} \\
&\geq \frac{1}{8\varpi^3} \qquad \text{by (6.5).}
\end{aligned}$$

This concludes the proof as we have shown that if $\mathfrak{M}_\Pi$ raises alarm with probability more than $2/\varpi = 1/\Pi$ probability, then $M^{\Psi\setminus R}$ succeeds with probability at least $1/(8\varpi^3)$, which is a contradiction. □

The construction of the monitor for $\mathcal{F}_{\widetilde{\text{CP}}}$ is also changed in a similar fashion. However, since the monitor in this case is defined based on an extractor, and the extractor itself will need to be modified to take polynomially many values of the locked state, the proof is much more involved.

Lemma 6.4. *For any polynomial (in the security parameter k) Π, under Assumption A2, there is a monitor satisfying the requirements specified by $\mathcal{MF}_{\widetilde{\text{CP}}}$ which does not inspect the locked state of the Network, such that the probability of the monitor raising an alarm is less than $1/\Pi$.*

PROOF: When $\wp_{\mathcal{F}_{\widetilde{\text{CP}}}}$ sends the commit message the monitor $\mathfrak{M}_\Pi$ must record a value w^* internally, by inspecting just the open state of the Network, so that it will not have to raise an alarm except with probability at most $1/\Pi$. First, we describe how $\mathfrak{M}_\Pi$ decides on w^*, and then argue why it satisfies the requirement.

We use the symbols λ, δ and δ^* similar to that in the proof for $\mathcal{F}_{\widetilde{\text{COM}}}$. Here again they correspond to the states of the Network at the point $\mathcal{F}_{\widetilde{\text{CP}}}$ sends the commit message to the verifier. The probability p_λ is also defined as there: it is the probability that the open

state of the Network is λ at point of commitment. Beyond this point, the proof differs from that of $\mathcal{F}_{\widetilde{\text{COM}}}$.

Note that $\mathcal{F}_{\widetilde{\text{CP}}}$ allows multiple proof sessions based on the same commitment. We shall use i to denote the proof number (i.e., the first proof will have $i = 1$ and so on). Let τ be an upperbound on the number of proofs (which is a polynomial depending on the environment). The monitor could use any proof to derive the witness w^*. However, depending on different values for the locked state, different proofs may have different probabilities of being accepted. So it will not be sufficient to consider just one value of i. As was done with locked state values in the case of $\mathcal{F}_{\widetilde{\text{COM}}}$, this will be dealt with by considering multiple values of i. In fact, the monitor will form a set Δ to "cover" *pairs* of the form (λ, i) for most values of λ and i.

Unlike in the case of $\mathcal{F}_{\widetilde{\text{COM}}}$, since the witness to be recorded is not necessarily revealed later, the witness recorded w^*_λ is defined in terms of an extractor $\mathcal{E}$. The extractor $\mathcal{E}$ when started with parameters (λ, δ, i) will emulate the Network, starting from the states (λ, δ) (which correspond to the open and locked states of the Network at the point $\mathcal{F}_{\widetilde{\text{CP}}}$ accepts the commitment). For purpose of analysis, we assume that the monitors for $\mathcal{MF}_{\widetilde{\text{COM}}}$ are also started for the instances of $\mathcal{MF}_{\widetilde{\text{COM}}}$ used in the proof phase. Further we shall condition on the event that none of these monitors raise an alarm. This will not make any significant difference to the conclusions below since only polynomially many such monitors are invoked (even taking into account the possible cloning of the Network by $\mathcal{E}$), and the probability that any of them raises an alarm is negligible (by considering an appropriate monitor).

Construction 6.9: Extractor $\mathcal{E}(\lambda, \delta, i)$.

Suppose the prover enters the i^{th} proof with $\mathcal{F}_{\widetilde{\text{CP}}}$, with a common input x. If the prover makes the commitment step in this proof, then at that point $\mathcal{E}$ clones the entire Network and runs two independent threads of execution. If in both threads the proofs are eventually accepted by $\mathcal{F}_{\widetilde{\text{CP}}}$, $\mathcal{E}$ checks if the t-bit queries made by $\mathcal{F}_{\widetilde{\text{CP}}}$ in $\text{ZKP}_R(x; r, c)$ are identical or not (where $t = \omega(\log k)$). If they are not identical this lets $\mathcal{E}$ extract a Hamiltonian cycle for the graph G (assuming the monitors for the $\mathcal{F}_{\widetilde{\text{COM}}}$s do not raise any alarm). Then $\mathcal{E}$ derives a witness (w, r', y) from this Hamiltonian cycle, and outputs it. Else it outputs $\perp$.

We shall denote the probability that $\mathcal{E}(\lambda, \delta, i)$ outputs a witness of the form $(w, \cdot, \cdot)$ by $p^{(\lambda,\delta,i)}_{\text{ext:}w}$. Also the probability that $\mathcal{E}(\lambda, \delta, i)$ outputs any witness at all will be denoted by $p^{(\lambda,\delta,i)}_{\text{ext}}$. Note that $p^{(\lambda,\delta,i)}_{\text{ext}} = \sum_w p^{(\lambda,\delta,i)}_{\text{ext:}w}$.

Now we are ready to describe how $\mathfrak{M}_\Pi$ chooses the value w^*_λ. It chooses a set Δ of δ and uses them to "cover" a set Λ of values for open states, as was done by the monitor for $\mathcal{MF}_{\widetilde{\text{COM}}}$. The only differences are that Λ consists of pairs (λ, i), rather than just the open state values λ, and instead of looking at $p^{(\lambda,\delta)}_{\text{rev}}$, now the monitor looks at $p^{(\lambda,\delta,i)}_{\text{ext}}$.[9] That is,

at each step of building Δ and Λ, we add a new δ to Δ if

$$\sum_{(\lambda,i)\notin\Lambda} p_\lambda p_{\text{ext}}^{(\lambda,\delta,i)} \geq 1/\varpi$$

and then use it to *cover* all (λ, i) for which

$$p_{\text{ext}}^{(\lambda,\delta,i)} \geq 1/(2\varpi\tau).$$

The covered pair (λ, i) is added to Λ.

For $(\lambda, i) \in \Lambda$, define $\delta_{(\lambda,i)}$ as the value in Δ which covers it. For $(\lambda, i) \notin \Lambda$, let $\delta_{(\lambda,i)}$ be an arbitrary value in Δ (say the lexicographically smallest value). Then we denote $p_{\text{ext:}w}^{(\lambda,\delta_{(\lambda,i)},i)}$ by $p_{\text{ext:}w}^{(\lambda,i)|\Delta}$, and $p_{\text{ext}}^{(\lambda,\delta_{(\lambda,i)},i)}$ by $p_{\text{ext}}^{(\lambda,i)|\Delta}$.

Then, similar to before, the following properties will hold.

$$\sum_{(\lambda,i) \text{ covered by } \delta} p_\lambda \geq 1/(2\varpi) \qquad \forall\delta \tag{6.6}$$

$$|\Delta| \leq 2\tau\varpi \tag{6.7}$$

$$\sum_{(\lambda,i)\notin\Lambda} p_\lambda p_{\text{ext}}^{(\lambda,\delta,i)} \leq 1/\varpi \qquad \forall\delta \tag{6.8}$$

$$p_{\text{ext}}^{(\lambda,i)|\Delta} \geq 1/(2\varpi) \qquad \text{if } (\lambda, i) \in \Lambda. \tag{6.9}$$

Note that in equation (6.7), we have an extra factor of τ now, because the each λ can be covered up to τ times (for each value of i).

Construction 6.10: Monitor $\mathfrak{M}_\Pi$.

Given λ, if there is some i such that $(\lambda, i) \in \Lambda$, then define w_λ^* as that w which maximizes $p_{\text{ext:}w}^{(\lambda,i)|\Delta}$. Note that for the same value of λ, we may have $(\lambda, i) \in \Lambda$ for multiple values of i; maximization is over all such pairs. If for no i do we have $(\lambda, i) \in \Lambda$, then define w_λ^* as the all-zero string. If the open state of the Network is λ at the point of commitment, then $\mathfrak{M}_\Pi$ records w_λ^* as the committed value.
Note that Λ may be exponentially large in k, but $\mathfrak{M}_\Pi$ is computationally unlimited. Also note that Λ is defined independent of the locked state of the Network δ^*. Thus indeed $\mathfrak{M}_\Pi$ can calculate w_λ^* at the point of commitment.
Then if for any proof in the session if $\wp_{\mathcal{F}_{\overline{\text{CP}}}}$ accepts a statement such that w_λ^* is not part of a valid witness for the statement, then $\mathfrak{M}_\Pi$ raises an alarm.

Our aim is to show that the probability $\mathfrak{M}_\Pi$ raises an alarm is less than $1/\Pi$.

Roughly, the covered values of λ (i.e., λ such that $(\lambda, i) \in \Lambda$ for some i) are such that with a good probability the extractor will extract some w starting with the locked state δ. However, this does not prevent the extractor from extracting different values of w with

significant probabilities. Intuitively however, extracting different values of w will let one find a collision in the hash function, and hence the weight of such λ should be small. We formalize this below.

Let $\Lambda_{\mathsf{bad}} \subseteq \Lambda$ contain all $(\lambda, i) \in \Lambda$ such that $p^{(\lambda,i)|\Delta}_{\text{ext}:w^*_\lambda} \leq \frac{1}{2} p^{(\lambda,i)|\Delta}_{\text{ext}}$. Then

Claim.

$$p^{(\lambda,i)|\Delta}_{\text{ext}:w^*_\lambda} \geq 1/(4\varpi) \qquad \textit{for } (\lambda, i) \in \Lambda \backslash \Lambda_{\mathsf{bad}} \tag{6.10}$$

$$\sum_{(\lambda,i)\in\Lambda_{\mathsf{bad}}} p_\lambda p^{(\lambda,i)|\Delta}_{\text{ext}} \leq \nu_k \tag{6.11}$$

where ν_k stands for some negligible function in k.

Equation (6.10) follows from the definition of Λ_{bad} and equation (6.9). To prove equation (6.11) first we observe that if $p^{(\lambda,i)|\Delta}_{\text{ext}:w^*_\lambda} \leq \frac{1}{2} p^{(\lambda,i)|\Delta}_{\text{ext}}$, then we can split the values w into two sizeable disjoint sets: W 1 and W 2 such that for $i = 1, 2$ we have $\sum_{w\in Wi} p^{(\lambda,i)|\Delta}_{\text{ext}:w^*_\lambda} \geq \frac{1}{4} p^{(\lambda,i)|\Delta}_{\text{ext}}$. Then, we can build a PPT machine $M^{\Psi_{\backslash V}}$ (with oracle access to $\Psi_{\backslash V}$) to find collisions in $\mathcal{H}$ (for ID V) with probability polynomially related to $\sum_{(\lambda,i)\in\Lambda_{\mathsf{bad}}} p_\lambda p^{(\lambda,i)|\Delta}_{\text{ext}}$, as follows.

Construction 6.11:

$M^{\Psi_{\backslash V}}$ to find collisions in $\mathcal{H}$. $M^{\Psi_{\backslash V}}$ simulates the entire Network (recall that the monitor operates only when V is honest, and hence $\Psi_{\backslash V}$ subsumes Ψ), with the following modifications:

- Instead of sending random r to $\wp_P$, it will accept r as an input and send that to $\wp_C$ as the first message in the commitment phase.
- On reaching an open state λ at the point after the commitment phase, for each $i = 1, \ldots, \tau$ and $\delta \in \Delta$ $M^{\Psi_{\backslash V}}$ runs two copies of the extractor $\mathcal{E}(\lambda, \delta, i)$.
- If for any $\delta \in \Delta$, the two executions of $\mathcal{E}(\lambda, \delta, i)$ extract different values w_1 and w_2, then $M^{\Psi_{\backslash V}}$ uses them to derive a collision.

The probability of $M^{\Psi_{\backslash V}}$ outputting a collision is bounded below by

$$\begin{aligned}\sum_{(\lambda,i)\in\Lambda_{\mathsf{bad}}} p_\lambda \left(\mathbf{Pr}\left[w_1 \in W1\right] \mathbf{Pr}\left[w_2 \in W2\right]\right) &\geq \sum_{(\lambda,i)\in\Lambda_{\mathsf{bad}}} p_\lambda \left(\frac{1}{4} p^{(\lambda,i)|\Delta}_{\text{ext}}\right)^2 \\ &\geq \frac{1}{16}\left(\sum_{(\lambda,i)\in\Lambda_{\mathsf{bad}}} p_\lambda p^{(\lambda,i)|\Delta}_{\text{ext}}\right)^2 .\end{aligned}$$

Since the left hand side is negligible, so must the right hand side, proving equation (6.11).

Now we need to show that, no matter what the actual value δ^* of the locked state of the Network is, the probability that $\mathfrak{M}_\Pi$ raises an alarm is smaller than $1/\Pi$. For this again we construct a collision finder $N^{\Psi_{\setminus V}}$ (somewhat resembling the above construction of $M^{\Psi_{\setminus V}}$, but using δ^* for one of the runs on which the extractor is applied).

Construction 6.12: $N^{\Psi_{\setminus V}}$ to find collisions in $\mathcal{H}$.

$N^{\Psi_{\setminus V}}$ simulates the entire Network (recall that the monitor operates only when V is honest, and hence $\Psi_{\setminus V}$ subsumes Ψ), with the following modifications:

- Instead of sending random r to $\wp_P$, it will accept r as an input and send that to $\wp_C$ as the first message in the commitment phase.
- On reaching an open state λ at the point after the commitment phase, for each $i = 1, \ldots, \tau$ and $\delta \in \Delta$ $N^{\Psi_{\setminus V}}$ runs a copy of the extractor $\mathcal{E}(\lambda, \delta, i)$. Further, it runs a copy of the extractor with the real locked state δ^*, $\mathcal{E}(\lambda, \delta^*, i)$.
- If $\mathcal{E}(\lambda, \delta^*, i)$ outputs w_1 and for any $\delta \in \Delta$, the execution of $\mathcal{E}(\lambda, \delta, i)$ extracts $w_2 \neq w_1$, then $N^{\Psi_{\setminus V}}$ uses (w_1, w_2) to derive a collision.

For every i we have

$$\mathbf{Pr}\left[N^{\Psi_{\setminus V}} \text{ succeeds}\right] \geq \sum_{\lambda} \left(p_\lambda p^{(\lambda,i)|\Delta}_{\text{ext:}w^*_\lambda} \sum_{w \neq w^*_\lambda} p^{(\lambda,\delta^*,i)}_{\text{ext:}w} \right).$$

This is because if in the "real run" (i.e., with the locked state δ^*) the extractor returns $w \neq w^*_\lambda$ and in the run with $\delta = \delta_{(\lambda,i)}$ if the extractor returns w^*_λ, then $N^{\Psi_{\setminus V}}$ succeeds in outputting a collision. Note that the right hand summation is over mutually exclusive events (one event for each value of λ). Now summing up the above for $i = 1, \ldots, \tau$ we get

$$\begin{aligned}
\tau\,\mathbf{Pr}\left[N^{\Psi_{\setminus V}} \text{ succeeds}\right] &\geq \sum_{\lambda,i} \left(p_\lambda p^{(\lambda,i)|\Delta}_{\text{ext:}w^*_\lambda} \sum_{w \neq w^*_\lambda} p^{(\lambda,\delta^*,i)}_{\text{ext:}w} \right) \\
&\geq \sum_{(\lambda,i)\in\Lambda} \left(p_\lambda p^{(\lambda,i)|\Delta}_{\text{ext:}w^*_\lambda} \sum_{w \neq w^*_\lambda} p^{(\lambda,\delta^*,i)}_{\text{ext:}w} \right) \\
&\geq \sum_{(\lambda,i)\in\Lambda\setminus\Lambda_{\mathsf{bad}}} \left(p_\lambda p^{(\lambda,i)|\Delta}_{\text{ext:}w^*_\lambda} \sum_{w \neq w^*_\lambda} p^{(\lambda,\delta^*,i)}_{\text{ext:}w} \right).
\end{aligned}$$

Now applying equation (6.10) (and dividing by τ), we obtain

$$\begin{aligned}\mathbf{Pr}\left[N^{\Psi\setminus V} \text{ succeeds}\right] &\geq \frac{1}{4\varpi\tau}\sum_{(\lambda,i)\in\Lambda\setminus\Lambda_{\mathsf{bad}}}\left(p_\lambda\sum_{w\neq w^*_\lambda}p^{(\lambda,\delta^*,i)}_{\text{ext:}w}\right)\\ &= \frac{1}{4\varpi\tau}\sum_{(\lambda,i)\in\Lambda}\left(p_\lambda\sum_{w\neq w^*_\lambda}p^{(\lambda,\delta^*,i)}_{\text{ext:}w}\right)-\sum_{(\lambda,i)\in\Lambda_{\mathsf{bad}}}\left(p_\lambda\sum_{w\neq w^*_\lambda}p^{(\lambda,\delta^*,i)}_{\text{ext:}w}\right).\end{aligned}$$

Then, applying equation (6.11), we obtain

$$\mathbf{Pr}\left[N^{\Psi\setminus V} \text{ succeeds}\right] \geq \frac{1}{4\varpi\tau}\sum_{(\lambda,i)\in\Lambda}\left(p_\lambda\sum_{w\neq w^*_\lambda}p^{(\lambda,\delta^*,i)}_{\text{ext:}w}\right)-\nu_k. \tag{6.12}$$

Now we shall relate $\mathbf{Pr}\left[\mathsf{alarm}\right]$ to $\sum_{(\lambda,i)\in\Lambda}\left(p_\lambda\sum_{w\neq w^*_\lambda}p^{(\lambda,\delta^*,i)}_{\text{ext:}w}\right)$. Recall that alarm occurs when $\wp_{\mathcal{F}_{\widetilde{\text{CP}}}}$ accepts a statement for which w^*_λ is not part of a valid witness, where λ is the open state of the Network at the point of commitment (in the commitment phase). Let alarm_i denote the event that the alarm is raised for the i^{th} proof. Then $\mathbf{Pr}\left[\mathsf{alarm}\right] \leq \sum_{i=1}^{\tau}\mathbf{Pr}\left[\mathsf{alarm}_i\right]$. Let α_i denote the full state of the Network at the point at which the extractor would have forked (i.e., the point at which $\wp_{\mathcal{F}_{\widetilde{\text{COM}}}}$— run internally by $\wp_{\mathcal{F}_{\widetilde{\text{ZK}}}}$ in turn run by $\wp_{\mathcal{F}_{\widetilde{\text{CP}}}}$— accepts the commitment for the zero knowledge proof). Let q_{α_i} denote the probability that conditioned on α_i, a proof of a statement for which w^*_λ is not part of a valid witness is accepted by $\wp_{\mathcal{F}_{\widetilde{\text{CP}}}}$. Then

$$\mathbf{Pr}\left[\mathsf{alarm}_i\right] = \sum_{\alpha_i}p_{\alpha_i}q_{\alpha_i}.$$

On the other hand,

$$\sum_{\lambda,i}p_\lambda\sum_{w\neq w^*_\lambda}p^{(\lambda,\delta^*,i)}_{\text{ext:}w} \geq \sum_i\sum_{\alpha_i}p_{\alpha_i}q_{\alpha_i}(q_{\alpha_i}-\nu_k) \geq 1/\tau\left(\sum_i\sum_{\alpha_i}p_{\alpha_i}q_{\alpha_i}\right)^2-\nu_k$$

where ν_k stands for a negligible function in k. Further,

$$\mathbf{Pr}\left[\mathsf{alarm}\right] \leq \sum_{i=1}^{\tau}\mathbf{Pr}\left[\mathsf{alarm}_i\right].$$

Combining the above inequalities we get

$$\sum_{\lambda,i}p_\lambda\sum_{w\neq w^*_\lambda}p^{(\lambda,\delta^*,i)}_{\text{ext:}w} \geq 1/\tau\,\mathbf{Pr}\left[\mathsf{alarm}\right]^2-\nu_k. \tag{6.13}$$

It remains to relate this to the sum on the left hand restricted to $(\lambda, i) \in \Lambda$.

$$\begin{aligned}\sum_{(\lambda,i)\in\Lambda} p_\lambda \sum_{w\neq w^*_\lambda} p^{(\lambda,\delta^*,i)}_{\text{ext:}w} &= \sum_{\lambda,i} p_\lambda \sum_{w\neq w^*_\lambda} p^{(\lambda,\delta^*,i)}_{\text{ext:}w} - \sum_{(\lambda,i)\notin\Lambda} p_\lambda \sum_{w\neq w^*_\lambda} p^{(\lambda,\delta^*,i)}_{\text{ext:}w} \\ &\geq \sum_{\lambda,i} p_\lambda \sum_{w\neq w^*_\lambda} p^{(\lambda,\delta^*,i)}_{\text{ext:}w} - \sum_{(\lambda,i)\notin\Lambda} p_\lambda p^{(\lambda,\delta,i)}_{\text{ext}}.\end{aligned}$$

Then appealing to equation (6.8),

$$\sum_{(\lambda,i)\in\Lambda} p_\lambda \sum_{w\neq w^*_\lambda} p^{(\lambda,\delta^*,i)}_{\text{ext:}w} \geq \sum_{\lambda,i} p_\lambda \sum_{w\neq w^*_\lambda} p^{(\lambda,\delta^*,i)}_{\text{ext:}w} - 1/\varpi. \tag{6.14}$$

Combining equations (6.12), (6.13) and (6.14)

$$\tau\, \mathbf{Pr}\left[N^{\Psi\setminus V} \text{ succeeds}\right] \geq \frac{1}{4\varpi}\left(\frac{1}{\tau}\,\mathbf{Pr}\left[\mathsf{alarm}\right]^2 - \nu_k - 1/\varpi\right) - \nu_k.$$

Since $\mathbf{Pr}\left[N^{\Psi\setminus V} \text{ succeeds}\right]$ is negligible by Assumption A2, we conclude that

$$\mathbf{Pr}\left[\mathsf{alarm}\right] \leq \sqrt{\tau/\varpi} + \nu_k.$$

By choosing $\varpi = 2\tau\Pi^2$ we obtain that the probability $\mathfrak{M}_\Pi$ raises an alarm, $\mathbf{Pr}\left[\mathsf{alarm}\right]$ is less than $1/\Pi$, as needed to be proved. □

Given that the monitors for $\mathcal{F}_{\widetilde{\text{COM}}}$ and $\mathcal{F}_{\widetilde{\text{CP}}}$ do not require to inspect the locked states, it easily follows that the monitors for the $\mathcal{F}_{\widetilde{\text{ZK}}}$ and $\mathcal{F}_{\text{CSC}}$ need not inspect the locked states either. (Recall that these monitors simply run the other monitors internally.) Thus we get the following result.

Lemma 6.5. *For any polynomial (in the security parameter k) Π, under Assumption A2, there is a monitor satisfying the requirements specified by $\mathcal{MF}_{\widetilde{\text{CSC}}}$ which does not inspect the locked state of the Network, such that the probability of the monitor raising an alarm is less than $1/\Pi$.*

This result is useful when the client K can keep its input locked until after the server Q commits to its input. (Note that the protocol CSC requires K to unlock its input *after* the commitment by Q.) Then the monitor is guaranteed to record the server's input independent of the client's.

6.6 Conclusion

In this chapter we presented an alternate notion of Los Angeles security, employing the new concept of monitors. Monitored functionalities capture a level of security which is weaker than Los Angeles security, but is nevertheless of intuitive value. The advantage of this notion is demonstrated by the fact that protocols like CSC which (after substituting

in all ideal subroutines) are extremely simple can be shown secure under this notion. However its usefulness is somewhat limited: in the case of client-server computation, the client must keep its input locked until after the server commits to its input, to have the guarantee that the server's input is independent of client's. The way we have formulated the framework, weaker guarantees of independence are not available. In the next chapter, we point out a few interesting directions in which this exploration could be furthered.

Notes

[1]Stand-alone security notions can also be seen as relaxations of Network-aware security, which admit much simpler protocols. However, when these protocols are deployed in a Network setting, the stand-alone security guarantees simply do not have any implications. Our goal is to develop relaxed *Network-aware* security definitions which, though weaker, are nevertheless for the Network setting.

[2]A monitor is different from an angel in a crucial manner: a monitor can see the entire Network, including inputs to the honest parties, where as an angel does not see anything that the adversary cannot. Further, the monitor does not communicate with any of the other entities in the Network, except by raising an alarm (which, it will be required, should happen only with negligible probability). In contrast an angel is available for counsel to the environment, the adversary and the transvisors.

[3]It is always the case that, in general, if a party uses its input in one session, then there may be no independence in a later protocol. However, Network-aware security can still provide a relative security guarantee which is useful if we know that the information leaked by a previous use of the input can be tolerated. In the context of monitored functionalities and locked states as presented here, we do not provide any guarantee of independence when an input has been used previously.

[4] However, typically the entire Network except for the session being analyzed and the adversary can be considered part of the environment. The monitors we demonstrate are robust against redefining the Network in this way.

[5]The monitor's recorded value is independent of as yet unsampled randomness in the Network. So if the client's input is only a freshly sampled random value, as is the case in a ZK proof or coin-tossing protocol, this issue does not arise.

[6]Incidentally, in the use of semi-functionalities in Chapter 5, the only locked state is future randomness. However this is an especially simple special case, taken care of by the proofs there. As it turns out generalizing to other locked states complicates our arguments considerably.

[7]In other words, the inputs are for *one time* use only. After that if it is used as a client input in a server-client computation protocol, there is no guarantee that the server's input will be independent of that input. This is a significant limitation. However note that a client's input for a "server-client" computation, with a corrupt server is the *last time* it can be used secretly, as the computation gives the client's input to the server.

It is an interesting problem to relax this information theoretic locking constraint to a computational equivalent.

[8]These arguments extend to the adaptive case. There the simulation is carried out *until* the receiver is corrupted.

[9]The reason for considering (λ, i) is that while relating to the alarm probability, we need to consider the possibility that the alarm may occur for any i.

Chapter 7

Conclusion

In this thesis we have overcome a proven impossibility, by an appropriate change in the theoretical abstraction involved. Our new notions of security are aimed at making the theory of cryptography more practically relevant, as much as at answering pressing problems posed in the literature. While the protocol in Chapter 5 for secure multi-party computation without trusted setups is our central result, the definition of security using angels is the critical enabling element in this work. The assumptions we used in Chapters 5 and 6 are also crucially used in the constructions.

There are several directions in which this research must be carried forward. We list a few of these below.

Complexity Theoretic Assumptions. Our novel complexity assumptions is a strength as well as weakness of our work. On the one hand we would like to know if more standard assumptions can yield similar results. This question has been in part answered by the recent work of Barak and Sahai [BS05], who avoid the "non-malleable" nature of our assumptions, though their protocols are significantly more involved. Another question is whether our assumptions can be reduced to other assumptions which may be easier to study or evaluate. Malkin et. al. [MMY05] use specific number theoretic assumptions (which are still non-standard) to instantiate hash functions satisfying our assumptions.

The other possible direction is to investigate the *implications* of our assumptions. In particular, we would like to know whether these or similar assumptions can yield improved protocols, or whether they have implications in cryptography beyond the construction of protocols for general multi-party computation.

In either direction, it is an intriguing problem to understand our assumptions further and see how they relate to more well-studied assumptions in complexity theory.

Tradeoffs Against Efficiency. Monitored security notion of Chapter 6 promises a viable efficiency-security tradeoff: by replacing certain security requirements by their moni-

tored equivalents, it may be possible to construct vastly simpler protocols. This tradeoff has more room for investigation, both in terms of constructing new functionalities and protocols within the monitored security framework, and in terms of devising even newer frameworks for providing weaker security guarantees.

Another tradeoff is between efficiency and the strength of complexity assumptions. Indeed, by requiring that it is not only hard to find a collision for the hash function $\mathcal{H}$, but in fact it is hard to even find a point in the range of the hash function which has multiple pre-images, one could convert the basic protocol BCOM itself into a secure protocol for $\mathcal{F}_{\text{COM}}$ (using an appropriate angel). However requiring that it is hard to find such points in the range, along with Assumption A1 is a very strong assumption. It is a challenge to assess this tradeoff as it requires a good understanding of the strength of the assumption.

Extensions. Several extensions are possible to our protocols, some of which are relatively simple. On the other hand certain other extensions are open questions.

Our results for general multi-party computation are all restricted to static adversaries. The proofs do not generalize to the setting of adaptive adversary: protocols COM and OMCP use perfectly binding commitments which prevents them from being secure against an adaptive attack. It is not clear if simple modifications of the our protocol can yield security against adaptive attacks. However Barak and Sahai [BS05] do provide such security guarantee for their protocol.

In the case of monitored functionalities, the extension to adaptive adversaries is easier, because the basic protocols BCOM and BZK do not use binding commitments, and are secure against adaptive adversaries; encryption protocol ENC can also be made secure against adaptive adversaries by using non-committing encryptions [CFGN96, DN00] (with appropriate complexity theoretic assumptions) as pointed out in [Can01, CLOS02]. The challenge, on the other hand, is to extend the applicability of the framework to more general multi-party computation problems (retaining the relative simplicity of the protocols). While it is not hard to extend the 2-party client-server computation to a multi-client-single-server setting, it seems problematic to extend it to a general setting with no server, and retain the independence guarantees as well as the advantages of using monitors. Intriguing possibilities include considering computational restrictions on monitors (but still giving them more power than the players) as well as formulating less than perfect independence guarantees for inputs.

Restricted Models. Studying simplified models is often useful in understanding fundamental issues and in developing tools which may be extended to more complicated settings. One direction of simplification would be to impose restrictions on the nature of the Network, on the pattern of corruptions allowed in the Network or on the kind of functionalities of interest. Indeed, models preceding the Network-aware security model can be considered to be the result of such restrictions. However it may be useful to consider newer models which are closer to the Network-aware security model in what we

now know are some crucial features (like requiring a transvisor-based simulation, and not a rewinding-based one). For instance [PS05] introduces a "fixed-roles" model called the server-client model (which we omit from this thesis), which otherwise retains all the features of the model used for Los Angeles Network-aware security. Interestingly, the resulting simplifications in the assumptions on the hash function $\mathcal{H}$ make it possible to realize such a hash function using standard (super-polynomial) assumptions [MMY05].

From Theory to Practice. From a practical point of view the ultimate goal of theoretical cryptography is to able to provide meaningful security guarantees for practically deployed schemes. Some of the challenges involved are of a fundamental nature and well within the realm of theoretical cryptography. We already mentioned the issue of efficiency. Another issue is regarding the use of randomness. In theory we assume that all parties have access to *perfect* randomness: secret, unbiased random bits which are independent of each other and of the rest of the Network. However this is somewhat problematic (even without going into the philosophical questions of modeling physical sources using probability distributions), because in practice, the sources available are seldom perfectly random. A more plausible guarantee is that they have *a high entropy rate*. However now we know [DOPS04] that such a guarantee is not enough. It remains an interesting question to model imperfect random sources theoretically and understand how they can be safely used in cryptography. Another challenge is to contain the side-channels by which the environment communicates with the adversary. Note that the security of the cryptographic protocols does not make any guarantee regarding the behavior of the environment. A more detailed and less pessimistic model of environment, adversary and communication will be be helpful in studying these side-channels, and also may prove useful in dealing with new (to theory) problems like denial of service attacks. Indeed, there is a lot of room for the trade-off between generality and usefulness.

While Network-aware security notions are fast evolving and provide a fertile ground for exploration and innovation, attempts at becoming relevant to practice will probably provide the most challenging and rewarding directions to pursue this exploration. This thesis — by providing flexible definitions of security, demonstrating secure protocols with no trusted setups and simpler protocols with a security-efficiency tradeoff, and devising concrete complexity assumptions to capture strong assumptions on hash functions used in practice — takes a step along that direction.

Notation Quick Reference

k Security parameter. *pg. 22.*

$\mathfrak{T}^{\Gamma}$ Class of PPT transvisors with access to the angel Γ. *pg. 38.*

$\mathcal{A}$ The Adversary in the Network. *pg. 17.*

$\mathcal{A}^{\Gamma}$ Class of PPT adversaries with access to Γ. May have restrictions in corruption patterns. *pg. 24.*

$\mathring{\mathcal{A}}$ Class of adversaries which do not corrupt any party or communicate with the environment, and deliver all messages immediately. *pg. 24.*

$\mathcal{A}^{\Gamma}_{\mathsf{SH}}$,$\mathcal{A}^{\Gamma}_{\mathsf{SH\text{-}static}}$ Class of semi-honest (honest-but-curious) adversaries. *pg. 24.*

$\mathcal{A}^{\Gamma}_{\mathsf{static}}$ Class of PPT adversaries with access to Γ, restricted to static corruption. *pg. 24.*

$\mathcal{S}^{\Gamma}$,$\mathcal{S}^{\Gamma}_{\mathsf{static}}$,$\mathcal{S}^{\Gamma}_{\mathsf{SH}}$,$\mathcal{S}^{\Gamma}_{\mathsf{SH\text{-}static}}$ Classes of simulators. *pg. 24.*

$\mathcal{Z}$ The Environment in the Network. *pg. 17.*

$\mathcal{Z}^{\Gamma}$ Class of PPT environments with access to Γ, invoking a single session of a protocol. *pg. 24.*

$\mathcal{Z}^{\Gamma}_{*}$ Class of PPT environments with access to Γ, invoking multiple sessions of protocols from a protocol collection. *pg. 24.*

$\mathcal{Z}$, $\mathcal{A}_{\mathsf{static}}$, $\mathcal{A}_{\mathsf{SH\text{-}static}}$, $\mathcal{S}_{\mathsf{static}}$, $\mathcal{S}_{\mathsf{SH\text{-}static}}$ Classes of environments, adversaries and simulators without access to any angel. *pg. 24.*

π,ρ, σ, ϕ Typical symbols used to denote unnamed protocols. *pg. 18.*

$\Sigma_{\pi,sid}$ A session of protocol π with session ID sid. When irrelevant or clear from the context, π and/or sid may be dropped from the symbol. *pg. 18.*

$\wp_{\Sigma,pid}$ A program instance associated with the protocol session Σ, executed by an honest party in the Network. pid identifies this program within the session Σ. When the session and program ID are clear or irrelevant, we simply write $\wp$. *pg. 18.*

$\mathcal{F}$ Typical symbol for an ideal functionality. *pg. 19.*

$\wp_{\mathcal{F}}$ An instance of the functionality $\mathcal{F}$. *pg. 19.*

μ Typical variable used to indicate the identity string of a party. *pg. 56.*

$\mathcal{I}_k$ Set of identities used for the parties. (See Section 5.9.) *pg. 56.*

pid Typical symbol to denote a program instance ID value. *pg. 18.*

sid Typical symbol to denote a session ID value. *pg. 18.*

Γ Typical symbol for an angel. *pg. 19.*

$\mathfrak{X}$ Set of corrupted parties in the Network. *pg. 57.*

Ψ The angel used in protocols of Chapters 5 and 6. *pg. 57.*

Ψ^* An oracle like Ψ, but does not filter queries based on $\mathfrak{X}$. *pg. 57.*

$\Psi_{\backslash\mu}$ An oracle like Ψ^*, but filters out queries for ID μ. *pg. 57.*

$\mathtt{comb}(\mathcal{E}_1, \ldots, \mathcal{E}_\ell)$ Combination of the entities $\mathcal{E}_1, \ldots, \mathcal{E}_\ell$ in the Network. *pg. 25.*

$\text{EXEC}_{\pi,\mathcal{Z},\mathcal{A}}$ Random variable for the bit output by $\mathcal{Z}$ in a Network with (at most) one session of the protocol π and adversary $\mathcal{A}$. *pg. 35.*

$\langle\mathcal{F}\rangle$ A dummy protocol for communicating with the ideal functionality $\mathcal{F}$. The dummy protocol $\langle\mathcal{F}\rangle$ typically involves just relaying to $\wp_{\mathcal{F}}$ all the inputs received, and outputting the messages received from $\wp_{\mathcal{F}}$. *pg. 36.*

$\mathcal{T}^{\rho\to\pi}$ Transvisor which transforms the view of the adversary from that of interacting with a session of ρ to that of interacting with a session of π. *pg. 38.*

$\pi^{\phi/\sigma}$ Protocol π with subroutine σ substituted by ϕ. (Mnemonic: π using ϕ *instead of* σ.) *pg. 44.*

$\mathsf{B}(\cdot)$ Hardcore predicate associated with the collection of trapdoor permutations T. *pg. 57.*

C A perfectly binding commitment scheme. *pg. 58.*

$\mathcal{D}_r^\mu$ A distribution over collisions in $\mathcal{H}(\mu, r, \cdot, \cdot)$. *pg. 56.*

$\mathcal{H}$ A non-self-reducible collision-resistant hash function. *pg. 56.*

T A collection of trapdoor permutations. T denotes the algorithm used to sample permutations and trapdoors — (f, f^{-1}) — from the collection. *pg. 57.*

Bibliography

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *Proc. 42nd FOCS*, pages 106–115. IEEE, 2001. Preliminary full version available on `http://www.math.ias.edu/~boaz`.

[BBP04] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In *Eurocrypt '04*, pages 171–188, 2004.

[Bea91a] Donald Beaver. Foundations of secure interactive computing. In *Crypto '91*, pages 377–391, 1991. LNCS No. 576.

[Bea91b] Donald Beaver. Secure multi-party protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 4:75–122, 1991.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th STOC*, pages 1–10. ACM, 1988.

[BL02] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. Cryptology ePrint Archive, Report 2002/043, 2002. Extended abstract appeared in STOC' 02.

[Blu82] Manuel Blum. Coin flipping by phone. In *Proc. 24th IEEE Computer Conference (CompCon)*, pages 133–137, 1982. See also *SIGACT News*, Vol. 15, No. 1, 1983.

[BPW04] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2004.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the First Annual Conference on Computer and Communications Security*. ACM, November 1993.

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition using super-polynomial simulation, 2005. To appear in FOCS' 05.

[Can95] Ran Canetti. *Studies in Secure Multi-party Computation and Applications*. PhD thesis, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 1995.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Electronic Colloquium on Computational Complexity (ECCC) TR01-016, 2001. Previous version "A unified framework for analyzing security of protocols" availabe at the ECCC archive TR01-016. Extended abstract in FOCS 2001.

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Revised version of [Can01].

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th STOC*, pages 11–19. ACM, 1988.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. Report 2001/055, Cryptology ePrint Archive, July 2001. Extended abstract appeared in CRYPTO 2001.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648. ACM, 1996.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Proc. 30th STOC*, pages 209–218. ACM, 1998.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proc. 26th FOCS*, pages 383–395. IEEE, 1985.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *Eurocrypt '02*, pages 337–351, 2002. LNCS No. 2332.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *Eurocrypt '03*, 2003. LNCS No. 2656.

[CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires $\tilde{\omega}(\log n)$ rounds. Record 2001/051, Cryptology ePrint Archive, June 2001. Extended abstract appeared in STOC' 01.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party computation. In *Proc. 34th STOC*, pages 494–503. ACM, 2002.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In *Crypto '03*, pages 265–281, 2003.

[Dam87] Ivan Damgård. Collision free hash functions and public key signature schemes. In *Eurocrypt '87*, pages 203–216, 1987. LNCS No. 304.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437 (electronic), 2000. Preliminary version in STOC 1991.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[DM00] Yevgeniy Dodis and Silvio Micali. Parallel reducibility for information-theoretically secure computation. In *CRYPTO*, pages 74–92, 2000. LNCS No. 1880.

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *CRYPTO*, pages 432–450, 2000. LNCS No. 1880.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero knowledge. In *Proc. 30th STOC*, pages 409–418. ACM, 1998.

[DOPS04] Yevgeniy Dodis, Shien Jin Ong, Manoj Prabhakaran, and Amit Sahai. On the (im)possibility of cryptography with imperfect randomness. In *FOCS*, pages 196–205. IEEE, 2004.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, February 1996. Preliminary version appeared in ICALP' 90.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proc. 21st STOC*, pages 25–32. ACM, 1989.

[GL90] Shafi Goldwasser and Leonid Levin. Fair computation of general functions in presence of immoral majority. In *Crypto '90*, pages 77–93, 1990. LNCS No. 537.

[GL02] Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In *DISC*, volume 2508 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2002.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984. Preliminary version appeared in STOC' 82.

[GMPY05] Juan Garay, Philip MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. Unpublished manuscript (Improvement on an earlier work which is available from `http://eprint.iacr.org/2004/009`), 2005.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proc.* 17*th STOC*, pages 291–304. ACM, 1985.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play ANY mental game. In ACM, editor, *Proc.* 19*th STOC*, pages 218–229. ACM, 1987. See [Gol04, Chap. 7] for more details.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, July 1991. Preliminary version in FOCS' 86.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001. Earlier version available on `http://www.wisdom.weizmann.ac.il/~oded/frag.html`.

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

[GRR98] Rosario Gennaro, Michael Rabin, and Tal Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In *Proc.* 17*th ACM PODC*, pages 101–111. ACM, 1998.

[GT03] Shafi Goldwasser and Yael Tauman. On the (in)security of the fiat-shamir paradigm. Cryptology ePrint Archive, Report 2003/034, 2003. Extended abstract appeared in FOCS' 03.

[KLP05] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In *Proc. 37th STOC*. ACM, 2005.

[KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in poly-logarithm rounds. In *Proc. 33th STOC*, pages 560–569. ACM, 2001. Preliminary full version published as cryptology ePrint report 2000/013.

[KPR98] Joe Kilian, Erez Petrank, and Charles Rackoff. Lower bounds for zero knowledge on the Internet. In *Proc. 39th FOCS*, pages 484–492. IEEE, 1998.

[Lin03a] Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *Proc. 35th STOC*, pages 683–692. ACM, 2003.

[Lin03b] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *Proc. 44th FOCS*. IEEE, 2003.

[Lin03c] Yehuda Lindell. A simpler construction of cca2-secure public-key encryption under general assumptions. In *Eurocrypt '03*, pages 241–254, 2003. LNCS No. 2656.

[Lyn96] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[MMY05] Tal Malkin, Ryan Moriarty, and Nikolai Yakovenko. Generalized environmental security from number theoretic assumptions. Unpublished Manuscript, 2005.

[MR91] Silvio Micali and Phillip Rogaway. Secure computation. In *Crypto '91*, pages 392–404, 1991. LNCS No. 576.

[Oka91] Tatsuaki Okamoto. An extension of zero-knowledge proofs and its applications. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT*, volume 739 of *Lecture Notes in Computer Science*, pages 368–381. Springer, 1991.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *Eurocrypt '03*, 2003. LNCS No. 2656.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *Proc. 36th STOC*, pages 232–241. ACM, 2004.

[PR03] Rafael Pass and Alon Rosen. Bounded-concurrent secure two-party computation in a constant number of rounds. In *Proc. 44th FOCS*. IEEE, 2003.

[PR05] Rafael Pass and Alon Rosen. New and improved constructions of non-malleable cryptographic protocols. In *Proc. 37th STOC*. ACM, 2005.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *Proc. 43rd FOCS*. IEEE, 2002.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: Achieving universal composability without trusted setup. Cryptology ePrint Archive, Report 2004/139, 2004. Extended abstract in Proc. 36th STOC, pages 242–251, ACM, 2004.

[PS05] Manoj Prabhakaran and Amit Sahai. Relaxing environmental security: Monitored functionalities and client-server computation. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 104–127. Springer, 2005.

[PSW00] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Secure reactive systems. Technical Report RZ 3206 (#93252), IBM Research, 2000. Available from `http://domino.research.ibm.com/library/cyberdig.nsf`. A shorter version "Cryptographic Security of Reactive Systems" in Workshop on Secure Architectures and Information Flow, Royal Holloway, Univ. of London, Dec. 1999.

[PW00] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *ACM Conference on Computer and Communications Security*, pages 245–254, 2000.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–, 2001. A longer version appeared in the Cryptology ePrint Archive, Report 2000/066.

[Rab81] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[RBO89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st STOC*, pages 73–85. ACM, 1989.

[RK99] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. In *Eurocrypt '99*, 1999. LNCS No. 1592.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proc. 22nd STOC*, pages 387–394. ACM, 1990.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proc. 40th FOCS*, pages 543–553. IEEE, 1999.

[Sha49] Claude Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, October 1949.

[SL94] Roberto Segala and Nancy A. Lynch. Probabilistic simulations for probabilistic processes. In Bengt Jonsson and Joachim Parrow, editors, *CONCUR*, volume 836 of *Lecture Notes in Computer Science*, pages 481–496. Springer, 1994.

[Yao82a] Andrew Chi-Chih Yao. Protocols for secure computation. In *Proc. 23rd FOCS*, pages 160–164. IEEE, 1982.

[Yao82b] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions. In *Proc. 23rd FOCS*, pages 80–91. IEEE, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proc. 27th FOCS*, pages 162–167. IEEE, 1986.