Enhanced Security Models for Network Protocols

by

Shabsi Walfish

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
January 2008

Yevgeniy Dodis

בס״ד

To my family, with love.

Acknowledgments

First and foremost, I thank my advisor Yevgeniy Dodis, who contributed to both this work, and my personal development as a researcher, in innumerable ways. It was Yevgeniy who first drew me into the study of cryptography with his irrepressible (and apparently contagious) enthusiasm for the subject. Throughout my time in the graduate program, brainstorming together with Yevgeniy was undoubtedly my favorite research activity. I greatly benefited from his incredible talent for seeing the relationships between concepts and the way they fit into the "big picture" of cryptography. He is a gifted teacher, and I feel most fortunate and honored to be his student.

I also thank several other contributors to this work, with whom I had the privilege to collaborate: Ran Canetti, Rafael Pass, and Victor Shoup. Ran closely guided the development of the new security models presented in Chapter 1, and taught me a great deal about the philosophy of security models (and the proper use of nomenclature). Rafael first encouraged me to study the problem of deniability, and I credit him with providing some of the early insights leading to the development of the deniable protocols presented in Chapter 4. I owe particular thanks to Victor for his significant contributions to the efficient protocols of Chapter 5, and, more generally, for teaching me by example how to be more precise in my reasoning. Victor is truly a world-class scientist, and I learned many important lessons from his incisive observations about probability theory, number theory, and formal proof techniques. Additionally, I thank Jonathan Katz and Adam Smith for their contributions to the impossibility results for deniable authentication appearing in Chapter 3. There are also many other researchers whom I thank for their various insightful comments and suggestions regarding this work, including Johan Håstad, Yehuda Lindell, Philip MacKenzie, and Silvio Micali.

My years in graduate school were greatly enhanced by interactions with my fellow students at New York University, including Carl Bosley, Nelly Fazio, Michael J. Freedman, Kristiyan Haralambiev, Antonio Nicolosi, Prashant Puniya, and Daniel Wichs. I thank them all for being encouraging and helpful compatriots.

Acknowledgments

Above all, I am deeply thankful to my family and friends for their support, understanding, and encouragement. I am especially grateful for the enduring support and assistance of my wonderful parents, Marvin and Sheila, who taught me the most important things in life. I am also very appreciative of my daughter, Tziporah, who (mercifully) slept quietly through the night whenever I was up late working, and always greeted me with a smile in the morning. Most importantly, I thank my beloved wife Meredith for all of her love, patience, and understanding as I completed the writing of this thesis.

Abstract

Modeling security for protocols running in the complex network environment of the Internet can be a daunting task. Ideally, a security model for the Internet should provide the following guarantee: a protocol that "securely" implements a particular task specification will retain all the same security properties as the specification itself, even when an arbitrary set of protocols runs concurrently on the same network. This guarantee must hold even when other protocols are maliciously designed to interact badly with the analyzed protocol, and even when the analyzed protocol is composed with other protocols. The popular Universal Composability (UC) security framework aims to provide this guarantee.

Unfortunately, such strong security guarantees come with a price: they are impossible to achieve without the use of some *trusted setup*. Typically, this trusted setup is global in nature, and takes the form of a Public Key Infrastructure (PKI) and/or a Common Reference String (CRS). However, the current approach to modeling security in the presence of such setups falls short of providing expected security guarantees. A quintessential example of this phenomenon is the *deniability* concern: there exist natural protocols that meet the strongest known security notions (including UC) while failing to provide the same deniability guarantees that their task specifications imply they should provide.

We introduce the *Generalized Universal Composability* (GUC) framework to extend the UC security notion and enable the re-establishment of its original intuitive security guarantees even for protocols that use global trusted setups. In particular, GUC enables us to guarantee that secure protocols will provide the same level of deniability as the task specification they implement. To demonstrate the usefulness of the GUC framework, we first apply it to the analysis and construction of *deniable authentication* protocols. Building upon such deniable authentication protocols, we then prove a general feasibility result showing how to construct protocols satisfying our security notion for a large class of two-party and multi-party tasks (assuming the availability of some reasonable trusted setup). Finally, we highlight the practical applicability of GUC by constructing efficient protocols that securely instantiate two common cryptographic tasks: *commitments* and *zero-knowledge proofs*.

Table of Contents

List of Figures

Introduction

The Evolution of Security Models

Prior to Claude Shannon's seminal paper [81] in 1949, which introduced the concept of provably secure cryptography, information security was little more than a continual arms race between the designers of security mechanisms and those who sought to circumvent security. Increasingly complex security schemes were constantly being designed and redesigned to combat novel attacks of increasing sophistication. Such an approach to security is ultimately untenable – when a new attack is discovered, it may remain possible to exploit the vulnerability of existing systems for extended periods of time while the system designers scramble to develop and deploy a new system capable of thwarting the attack. Shannon's notion of provable security represented the first attempt to escape from this constant cycle of attack discovery and system redesign – a provably secure design will always remain secure, irrespective of new techniques that may be discovered. However, as Shannon's work immediately illustrated, designing *practical* security systems that are provably secure against *any* conceivable attack is, simply, an impossible task.

Rather than abandon the notion of provable security altogether, modern cryptographers have leveraged a fundamental observation: "real world" adversaries are typically subject to practical limitations (for example, due to the basic laws of physics), so the class of attacks that a secure system actually needs to withstand may be somewhat restricted. Therefore, in recent years cryptographers have focused much of their efforts on defining *security models* that reflect various (assumed) limitations on attackers' capabilities. The most basic and successful "modern" security model was implicitly introduced by Diffie and Hellman's ground-breaking paper on public key cryptography [40] in 1976, and was later established formally in [54]. The adversarial model of [40] was the first to employ the notion of *computationally intractable problems* – problems that are assumed to be too difficult to solve in any reasonable amount of time with modern computing resources (even though their solutions do exist). Such computational intractability can be leveraged for the purposes of designing secure systems by first assuming some particular practical limit on the computational resources available to an attacker. This approach is commonly referred to as "computational security", since security is only guaranteed against com-

putationally bounded adversaries (in contrast to Shannon's perfectly impenetrable "information theoretic" security model). Thus, rather than attempting to construct systems secure against all possible attacks, for the purposes of computational security it is sufficient to construct systems secure against all "Probabilistic Polynomial Time" (PPT) bounded adversaries. By realistically restricting the computational abilities of the adversary in this fashion, [40] demonstrated the possibility of much more practical provably secure systems.

While the notion of computational security allows cryptographers to avoid the impractical requirements imposed by information theoretic security, it also comes with drawbacks. For instance, we do not know of any useful[1] problems which can be proven to be computationally intractable, and, furthermore, it seems highly unlikely that such proofs can be found.[2] Thus, we are forced to rely on specific intractability assumptions when implementing secure systems in this model. The extent to which this is a drawback largely depends on the supporting evidence behind the assumptions being made, and, therefore, an attempt is made to rely only on the most reasonable and thoroughly vetted assumptions. Another difficulty arises when considering issues of efficiency in implementations. Choosing specific parameters for the "size" of the "intractable" problem (upon which the efficiency of the cryptosystem depends) also fixes the quantity of computational resources required for an adversary to solve it. Since it is difficult to know precisely what computational resources will become available to the attacker over the course of time, it is possible (or even likely) that solving a specific instance of a problem which seems to be computationally intractable at the present time will become feasible in the (near) future. Finally, particular technological breakthroughs (such as quantum computers) could potentially violate intractability assumptions (most notably including the "hardness of factoring assumption"), rendering all systems relying on these assumptions insecure.

Thus, modern security models represent a tradeoff between "heuristic" design principles (where the cryptosystem designer merely attempts to thwart all known attacks) and truly provable security. This tradeoff allows us to implement practical systems that are provably secure, but at the expense of making some specific assumptions about our adversary which may not always hold

[1] In this context, a functional problem may be considered "useful" if its solution can be checked efficiently.

[2] Such a proof would conclusively show that "$P \neq NP$", which is widely assumed to be essentially unprovable.

true. (Still, such assumptions represent a significant improvement over the heuristic approach, which implicitly assumes that the attacker will not invent *any* new method of attack.) Ultimately, our goal is to make our assumptions about the limitations of the adversary as realistic as possible. Indeed, it is important to focus on *realism* when designing any security model. If the security requirements of the model are too strong (such as the information theoretic modeling of Shannon), it fails to admit many useful schemes that have practical security guarantees. On the other hand, if the requirements of the model are too weak, it might admit some schemes which are insecure in practice. Bearing this in mind, we are in a position to better appreciate the formative influences at work and proceed to discuss the next steps in the evolution of security models.

Following the advent of computational security notions, the issue of designing interactive *protocols* to solve general security and privacy related problems drove the development of new security models for networked environments, such as local area networks and the Internet. To facilitate the process of modeling security requirements in network environments, these newer models are often built around a security "framework" that formally models certain common features of a particular communication network environment. These frameworks help to avoid overly-complicated security definitions for simple tasks by separating the modeling of network interactions from the modeling of the task itself. Furthermore, since many cryptographic tasks are applicable to multiple varieties of network environments, it makes intuitive sense to separate the security definition for the task from the security model for the network.

The most basic "network security framework" models a single dedicated channel between participating parties, with only one instance of the protocol running over the channel. This model was used implicitly in early works on cryptographic protocols, such as "zero-knowledge proof" protocols introduced by Goldwasser et al. in [55]. Perhaps most importantly, it was shown by Yao in [85] and Goldreich et al. in [51] that it is possible to solve virtually *any* multi-party protocol problem in this model (*i.e.*, securely compute any reasonable[3] function of the parties' inputs, without revealing any information beyond the output of the specified function) by using special "secure protocol compilers". Henceforth we refer to such protocol problems as general *multi-party computation* problems.

[3]Here, "reasonable" essentially means computable in PPT.

Once a particular framework has been selected to model the behavior of the network itself, we still require a means of defining security concerns specific to a given multi-party computation problem. One highly elegant and fundamental technique for defining security of a particular task is to employ the *trusted party paradigm*, first introduced in [51]. Intuitively, we will say that a protocol securely "realizes" a given computational task if running the protocol amounts to "emulating" an *ideal protocol* where all parties secretly hand their inputs to an imaginary "trusted party" who locally computes the desired outputs and secretly hands them back to the parties. One potential advantage of this paradigm is its strong "composability" property: if a protocol π truly emulates a certain trusted party $\mathcal{F}$, this would naturally imply that any system that makes use of the trusted party $\mathcal{F}$ should, in principle, behave the same way even when replacing the (perfectly secure) invocations of the trusted party $\mathcal{F}$ by invocations of the (practical) protocol π that realizes $\mathcal{F}$.

Several formalizations of the above intuitive ideas exist, *e.g.*, [53, 69, 9, 18, 41, 76, 19, 78, 8]. These formalizations vary in their rigor, expressibility, generality and restrictiveness, as well as security and composability guarantees. Among these, Canetti's Universal Composability (UC) framework [19] aims to achieve a very strong notion of composability that holds even within a very general network "environment". In a nutshell, the UC framework provides the following powerful security guarantee:

> *A UC-secure protocol π implementing a trusted party $\mathcal{F}$ does not affect any activity in the network environment differently than $\mathcal{F}$ does — even when there are arbitrary protocols running concurrently with π that may be maliciously constructed.*

Surely, if we can provably assert such a security guarantee for our protocols, then they are sufficiently secure even for use in the complex environment of the Internet.

Unfortunately, as a consequence of the extremely stringent security requirements of the UC framework, it was shown in a series of works [19, 23, 29, 66] that many important security tasks *cannot* be proven secure in the "plain" UC framework. In particular, most useful two-party protocols (such as commitment schemes, zero-knowledge proofs, etc.) cannot be proven to be UC-secure. Indeed, the provably UC-secure protocols in [19] require that a majority of protocol participants be honest, which is not a reasonable requirement for two-party protocols (since,

for two parties, this implies that all participating parties are honest). Therefore, while the UC framework gives us the security guarantees that we want, it would seem we have once again arrived at an impasse akin to the sweeping impossibility results implied by Shannon's work.

In order to circumvent this impasse and enable the construction of useful protocols that retain strong security and composability guarantees, various enhancements and modifications to the UC framework that enable the realization of useful two-party tasks have been proposed. Most modifications to UC are in fact *relaxing* the security requirements of the plain UC framework (which explains how it once again becomes possible to realize useful functionalities). Such relaxations can be dangerous, since they tend to admit new attacks with ramifications that must be carefully considered. For example, the "angel"-based security of Prabhakaran et al. [78] relaxes the UC security model through the use of some (fictitious) "angels" that can run super-polynomial time algorithms. Given that these "angels" can potentially break many computational security assumptions, it is not entirely clear to what extent their introduction weakens the security guarantees of the UC framework. Similarly, Barak et al. [8] also relax the model by allowing super-polynomial time simulations. Alternatively, it is possible to overcome the impossibility results by assuming an a priori bound on concurrency in the network, as in [67, 75, 74], or by employing "timing assumptions" as was done by Kalai et al. in [62]. However, in addition to weakening the security model, these alternative approaches seem to be inherently too impractical for use in large networks such as the Internet (for efficiency reasons).

On the other hand, some *enhancements* to the UC-framework make useful two-party protocols possible by additionally modeling the availability of certain real-world tools. These tools can be used to help bolster the security of honest parties, enabling them to achieve the (previously impossible) security goals of UC. Such tools may be useful both for overcoming the impossibilities of the "plain" UC model, as well as (potentially) for improving the practical efficiency of implementations.

For instance, in [30], Canetti et al. introduce a "Common Reference String" (CRS) – which must be generated by trustworthy means – into the UC framework. Assuming the additional availability of *secure channels*, it was shown in [30] that such a CRS can be used to realize any reasonable two-party or multi-party computation task with full UC security. We will return to this model shortly, but for now we note that a CRS is merely one possible example of a *trusted*

setup assumption that can be used to enable the construction of UC secure protocols for two-party tasks. Another example of such a *trusted setup* tool is the key registration model of Barak et al. [6]. Notably, both [30] and [6] took some liberties when idealizing the "real world" version of these tools for the purposes of security modeling, as we will soon see.

Developing Realistic Enhancements

The main focus of this thesis regards the development and application of highly *realistic* enhancements for the UC framework. Each of the aforementioned previous works on enhancements and relaxations of the UC framework introduced some unrealistic aspects into the modeling. Such departures from the more fundamentally realistic "plain" UC model can produce a palpable weakening of the security guarantees provided by the model. While it is clearly impossible to develop a completely realistic mathematical model of security, we will demonstrate how to make a subtle (but important) improvement to the realism of the modeling which leads to a substantial qualitative difference in the security guarantees it provides.

We begin our investigation of this phenomenon by pointing out a natural limitation of the UC framework: it was designed to model situations where secure protocol sessions are "self-contained", and, therefore, it does not provide any mechanism from protocols to *share state*. Indeed, one would normally consider it quite dangerous for a secure protocol session to either leak state information, or import state information from other protocols. However, there is an exception to this rule that occurs whenever protocols employ a one-time *global setup*. Since global operations are generally expensive, most real-world examples of global setups – such as a Common Reference String (CRS) setup or a Public Key Infrastructure (PKI) setup – are performed one time only. Multiple secure protocol sessions must then re-use the one-time setup, and this means that those secure protocols are indeed sharing state information across sessions. However, the plain UC modeling requires that any such setup used by a particular protocol session be *local* to that protocol session (*i.e.*, the setup is invoked when the protocol session begins, and is not reused by any other protocol sessions).

The approach taken by previous works in the UC framework to modeling of such one-time global setups employs the Joint-state Universal Composition (JUC) theorem of Canetti and Rabin [31]. At a high level, the JUC theorem allows multiple protocol sessions to share state

by modeling them as sub-sessions within a single protocol. That is, if a protocol π makes use of some one-time global setup $\mathcal{G}$, we model it as part of a "multi-session extension" protocol Π which parties can use to invoke multiple instances of π. Since Π is only run once (despite multiple invocations of the sub-protocol π), it will invoke a single instance of $\mathcal{G}$ that will not share state with any protocol outside of a single session of Π. Yet, this single instance of $\mathcal{G}$ is used by Π to run multiple instances of the protocol π, thereby allowing us to capture the re-usability of $\mathcal{G}$.

However, a subtle problem arises when applying the aforementioned JUC theorem to the modeling of certain global setups. Since the single instance of $\mathcal{G}$ is accessible only to Π in the modeling, we are implicitly assuming that only protocol sessions contained within π can make use of the setup. On a superficial level, this seems to imply only that honest parties should agree not to re-use information published by $\mathcal{G}$ in any protocol that was not specifically modeled within Π. In and of itself, this would not be very problematic in practice (for example, it is already standard in the cryptographic community to recommend that secret keys should only be used for a single application). Unfortunately, if we dig a little deeper, we discover a more insidious problem with the model. Since we will ultimately realize $\mathcal{G}$ via an actual trusted party on the network, who publicizes the same information that $\mathcal{G}$ does, it is entirely possible that other (arbitrary) activity in the network environment might be influenced by the presence of this information. That is, the *environment* can observe global setups in the real world, whereas the model only allows parties to access $\mathcal{G}$ via a specified protocol Π. A more realistic modeling would grant the environment access to $\mathcal{G}$ as well, enabling it to observe public values in the same way that real world parties can.

Of course, this immediately raises the question of whether or not such a subtle problem with the modeling can lead to actual vulnerabilities in secure protocols. As a case study, consider the CRS model employed by [30]. There, due to the application of JUC theorem, the CRS is assumed to be generated and published *only* to instances of a particular UC-secure realization of the "bit commitment" task. To model the "public" aspect of the information, a copy of the CRS is also assumed to be given to the real world adversary attacking the protocol session. However, the technique employed by [30] to show the existence of a simulator for any attack on the realized bit commitment protocol does something unrealistic: it allows the *simulator* to generate the CRS when attacking the ideal bit commitment functionality. Of course, in the real world, the CRS

must be generated by a trusted party. This means that, *in the real world*, it is not possible to employ the simulation technique of [30] since once can clearly distinguish the simulation from real world interactions by the use of a different CRS. In fact, as long as the CRS is generated by a trustworthy party, it can be proved[4] that *no simulation* attacking only the ideal bit commitment functionality can produce transcripts that are indistinguishable from real world transcripts of the commitment protocol proposed in [30]. This directly contradicts the intuitive basis of security in the trusted party paradigm – it is possible to learn something by attacking the real world protocol which it is *not* possible to learn by attacking the ideal functionality.

Still, it is difficult to say precisely what damage this "unsimulatable" information does to our security in practice. In particular, the environment in [30] can still somehow learn the value of the CRS by obtaining it from the adversary. This means that any attack which exploits the "unsimulatable" information must rely upon the distinguisher having access to *trustworthy* copy of the CRS (since, if one can lie about the CRS to the distinguisher, it is still possible to simulate any attack via the technique of [30]). What difference can the trustworthiness of the CRS possibly make to the distinguisher? There is at least one intuitive security property that can be violated when the distinguisher trusts the source of the CRS: *deniability*. If a party P who runs the ideal bit commitment protocol later wishes to deny having done so, it is safe for P to make such a claim since the ideal bit commitment functionality leaves absolutely no traces of interaction with P (see Chapter 4 for more details). On the other hand, a party who runs the commitment protocol of [30] *cannot* plausibly deny having done so, since it is impossible for anyone else to simulate the same protocol messages with respect to the real CRS. In a sense, we are stating a tautology – if an attack on some protocol π realizing an ideal functionality $\mathcal{F}$ cannot be indistinguishably simulated in the real world, then one cannot plausibly deny the use of π instead of $\mathcal{F}$. For the special case where $\mathcal{F}$ itself leaves behind no evidence of interaction (as is the case with the ideal functionality for bit commitments), we intuitively say that π is not "deniable", whereas $\mathcal{F}$ itself was inherently "deniable".

The problem with deniability that arises due to the modeling of [30] is, in fact, common to the model of [6] as well, and inherently arises in previously proposed practical relaxations of UC-security as well (namely, the same problem arises when using the "angel" modeling of [78]).

[4]See the impossibility result in Section 4.2.

Worse yet, all of the aforementioned works assume the presence of *ideal authenticated channels*. Ideal authenticated channels are inherently deniable, yet we do not have such channels in the real world. Instead, we must use a message authentication protocol in combination with yet another global setup (generally, a PKI). Here again, we will run into the problem of how to realistically model the global setup within the confines of the UC framework – if we cannot address this, then *all* protocols realized using authenticated channels will lose deniability, whether or not they depend on any additional global setup. Since the task of global authentication *inherently* requires some kind of global setup (in order to provide parties with some uniquely identifying information), it is of absolutely crucial importance that we solve the problem of realistically modeling global setup for any scenario where ideally authenticated channels are required.

In order to realistically model global setups, we must *generalize* the original UC framework of [19] to support protocols that share state. In particular, our generalization must allow for the modeling of protocols that share some state with other *arbitrary* protocols executing in the network environment. To this end, we introduce the Generalized Universal Composability (GUC) framework. The GUC framework will allow us to achieve the following intuitive notion, which should be compared to the intuition of the original UC framework:

> *A GUC-secure protocol π using some global setup $\mathcal{G}$, and implementing a trusted party $\mathcal{F}$, does not affect any activity in the network environment differently than $\mathcal{F}$ does — even when there are arbitrary protocols running concurrently with π that may be maliciously constructed, and even when those protocols have access to the same global setup $\mathcal{G}$.*

With the support of the new GUC framework, we can realistically model the information publicized by a global setup by giving it directly to the environment (in a trustworthy way), as well as to all parties. This means that attack simulations cannot "lie" to the environment about the information published by the global setup (as per the modeling of [30]), since the environment will have direct access to it. A direct consequence of this more realistic modeling is that the simulation used in the proof of security for a protocol (in the trusted party paradigm) can actually be run in the real world, and the result will still be indistinguishable from real protocol interactions. For the special case of "deniable" functionalities that we previously mentioned,

any GUC-secure realization of the functionality using such a realistic modeling of global setup will retain the *same* deniability guarantees as the ideal functionality. Therefore, the primary application of the new GUC model is to realize such "deniable" functionalities while still retaining the desirable deniability property. More generally speaking, the GUC model enables protocol designers to employ a more realistic modeling of global setup – and, as has historically been the case, more realistic security models ultimately lead to more secure (or more practical) protocols.

Organization

The remainder of this thesis is organized into five chapters, followed by some concluding remarks.

Chapter 1 introduces our new *Generalized Universal Composability* (GUC) framework, and proves that a strong *composition theorem*, analogous to that of the UC framework, holds in the GUC framework as well. We also introduce a simplified variant of GUC, which we call *External-subroutine Universal Composability* (EUC). The EUC framework is slightly simpler and more restrictive than GUC, and is less satisfyingly realistic on an intuitive basis, but it is considerably simpler to work with when proving security. Despite the additional restrictions and simplicity of the EUC notion, for any protocols which *can* be modeled in the EUC framework (which includes nearly all natural protocols using a global setup), we prove that EUC security is *equivalent* to GUC security.

Chapter 2 details many fundamental notions, definitions, and tools that are used throughout the remainder of the thesis. Most importantly, we introduce the various global setup models that we refer to in our impossibility results and constructions. The global setups we model include both Public Key Infrastructure (PKI) and Common Reference String (CRS) setups, in addition to a new setup notion we call "Augmented Common Reference String" (ACRS) setup. We will later show some important tasks for which a CRS setup does not suffice, but for which our newly proposed ACRS setup model does suffice.

Chapter 3 explores the notion of deniability in greater detail, and studies the problem of realizing ideal authenticated channels (which require "deniable authentication"). In particular, since authentication typically requires the use of a global PKI, we consider the ramifications of the GUC modeling on the applicability of a global PKI for authentication. Surprisingly, we show that it is *impossible* to realize ideal authenticated channels via a global PKI, unless we assume

a very limited adversary that is somewhat unrealistic. However, we also propose an alternative (and reasonable) setup assumption that extends the PKI model by employing a specialized one-time symmetric key exchange protocol with the following properties: 1) if the protocol succeeds, parties obtain a key that allows them to realize ideal authenticated channels, but 2) if the protocol aborts, the adversary can potentially prove that the parties were attempting to communicate. This realization allows us to continue using ideal authenticated channels in order to construct deniable protocols, and then we can rest assured that they remain secure in practice (provided that the setup phase has successfully completed).

Chapter 4 begins by proving that, even assuming the presence of ideal authenticated channels, the CRS model of [30] *does not* suffice to GUC-realize most useful functionalities (when the CRS is realistically modeled as a global entity). Specifically, we prove that a realization of ideal bit commitment is impossible in the CRS model (which, in turn, rules out many natural tasks that imply bit commitments). At the same time, we show that the ACRS model *does* suffice to GUC-realize most useful functionalities – specifically, we show how to GUC-realize bit commitments. Furthermore, we show how to apply the techniques of [30] to GUC-realize *any* reasonable (*i.e.*, "well-formed") two-party or multi-party task in the ACRS model.

Finally, Chapter 5 shows how construct efficient GUC-secure protocols for *string* commitments and "zero-knowledge proofs" in the ACRS model. Our constructions achieve practical levels of efficiency by employing specific number theoretic assumptions. Furthermore, we show how the network latency of our protocols can be reduced dramatically by employing the Random Oracle (RO) model [11] to obtain round-optimal protocols for these tasks. The results of this chapter conclusively demonstrate that it is possible to design very practical protocols in the new GUC framework.

We conclude by discussing future directions for the GUC framework, and security models in general.

1 • The Generalized Universal Composability (GUC) Framework

Perhaps the must crucial aspect of "secure" protocol design is the meaning we impart to the word "secure". For this reason, we begin our exploration of network security by presenting a new formal security model: the *Generalized Universal Composability* (GUC) framework. This model builds off of the Universal Composability (UC) framework developed by Canetti [20], by extending the modeling to include *shared state*. Protocols inherently share some state information across multiple sessions (or, potentially, even with sessions of entirely different protocols) whenever they make use of a *global setup*. It is extremely common for cryptographic protocols to make use of a global setup, such as a PKI or CRS. Protocols that do not require such global setup have already been well studied in the context of the UC framework – therefore, we will focus our attention on applications of the new GUC framework to the design and analysis of protocols using global setup.

Before delving into the details for our new security framework, we begin with a high-level overview of the necessary concepts.

1.1 Overview of Generalized UC Security

To provide the proper setting for the new developments of the GUC framework, we now briefly review the original UC framework of [19] (henceforth referred to as "Basic UC"). To keep our discussion at a high level of generality, we will focus on the notion of protocol "emulation", wherein the objective of a protocol π is to emulate another protocol φ. Here, typically, π is an implementation (such as the actual "real world" protocol) and φ is a specification (which directly computes the desired "ideal functionality" $\mathcal{F}$ in an "ideally secure" way, by using a trusted entity). Throughout our discussion, all entities and protocols we consider are "efficient" (*i.e.*, polynomial time bounded Interactive Turing Machines, in the sense detailed in [20]).

The Basic UC Framework. At a very high level, the intuition behind security in the basic UC framework is that any adversary $\mathcal{A}$ attacking a protocol π should learn no more information than could have been obtained via the use of a simulator $\mathcal{S}$ attacking protocol φ. (More generally, the adversary attacking π should not be able to accomplish any goal that could not have been accomplished by the simulated attack on protocol φ instead.) Furthermore, we would like this emulation guarantee to be maintained even if φ were to be used a subroutine in (*i.e.*, composed with) arbitrary other protocols that may be running concurrently in the networked environment – even after we substitute π for φ in all the instances where it is invoked. Thus, we may set forth a challenge experiment to distinguish between actual attacks on protocol π, and simulated attacks on protocol φ (referring to these protocols as the "challenge protocols"). As part of this challenge scenario, we will allow adversarial attacks to be orchestrated and monitored by a distinguishing environment $\mathcal{Z}$ that is also empowered to control the inputs supplied to the parties running the challenge protocol, as well as to observe the parties' outputs at all stages of the protocol execution. One may imagine that this environment represents all other activity in the network, including the actions of other protocol sessions that may influence inputs to the challenge protocol (and which may, in turn, be influenced by the behavior of the challenge protocol). Ultimately, at the conclusion of the challenge, the environment $\mathcal{Z}$ will be tasked to distinguish between adversarial attacks perpetrated by $\mathcal{A}$ on the challenge protocol π, and attack simulations conducted by $\mathcal{S}$ with protocol φ acting as the challenge protocol instead. If no environment can successfully distinguish these two possible scenarios, then protocol π is said to "UC-emulate" the protocol φ.

Specifying the precise capabilities of the distinguishing environment $\mathcal{Z}$ is crucial to the meaning of this security notion. We want $\mathcal{Z}$ to somehow encompass the entire range of activity in the network – which may be under adversarial control – that is external to the challenge protocol but can potentially interact with it. If the environment $\mathcal{Z}$ is to properly capture general interaction between other activity in the network and the challenge protocol, then $\mathcal{Z}$ *must* be able to choose the inputs to the challenge protocol and observe its outputs (for example, both of these capabilities are required to model the scenario where the challenge protocol is being called as a subroutine by another network protocol). Additionally, we must also grant $\mathcal{Z}$ the ability to interact with the attacker (which will be either the adversary, or a simulation), in order to model

the capability of the attacker to coordinate its attacks by exploiting concurrent network activities (which the environment is responsible for modeling). Granting precisely these capabilities to $\mathcal{Z}$, and then further allowing $\mathcal{Z}$ to invoke only a *single session* of the challenge protocol, is precisely the model underlying the notion of UC-emulation originally put forth in Canetti's basic UC framework [19]. Canetti showed that these emulation requirements are sufficient to obtain a very strong *composition theorem*, which guarantees that (potentially many) arbitrary instances of the φ that may be running in the network can be safely substituted with any protocol π that UC-emulates φ. Thus, even if we *constrain* the distinguisher $\mathcal{Z}$ to a fairly basic kind of interaction with a single adversary and a *single session* of the challenge protocol (even if we don't provide $\mathcal{Z}$ the ability to invoke other protocols at all), we can already achieve the strong and intuitive security guarantees we desired. Notably, although the challenge protocol might invoke subroutines of its own, it was not necessary to grant $\mathcal{Z}$ any capability to interact directly with any such subroutines.

To distinguish the common (but special) case where φ is a "specification" (*i.e.*, an "ideal" secure protocol) and π is a "realization" (*i.e.*, a more practical "real" protocol), we often say that the protocol φ is used in the "ideal world" whereas π is used in the "real world". We also introduce the notion of "hybrid models" to help conceptually modularize the design of protocols. A protocol π is said to be "realized in the $\mathcal{G}$-hybrid model" if π invokes the ideal functionality $\mathcal{G}$ as a subroutine (perhaps multiple times). (As we will soon see below, the notion of hybrid models greatly simplifies the discussion of UC secure protocols that require "setup".) A high-level conceptual view of UC protocol emulation in a hybrid model is shown in Figure 1.1.

Limitations of Basic UC. Buried inside the intuition behind the basic UC framework is the critical notion that the environment $\mathcal{Z}$ is capable of utilizing its input/output interface to the challenge protocol to mimic the behavior of other (arbitrary) protocol sessions that may be running in a computer network. Indeed, as per the result of [19] mentioned in our discussion above, this is, in fact, the case – at least, when considering challenge protocols that are essentially "self-contained". Such self-contained protocols, which do not make use of any "subroutines" (such as ideal functionalities) belonging to other protocol sessions, are called *subroutine respecting* protocols – and the basic UC framework is focused on modeling this kind of protocol. On the

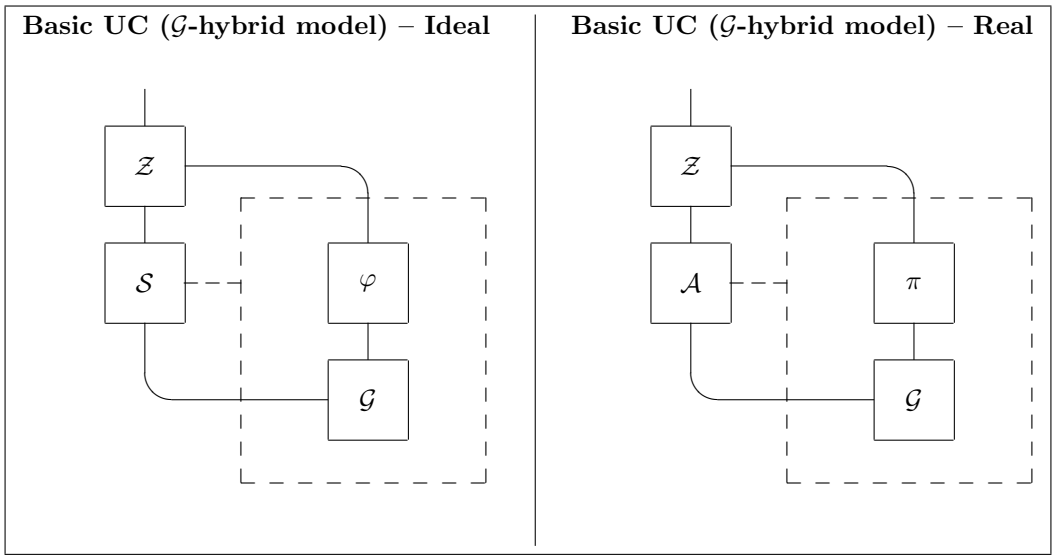

Figure 1.1: The Basic UC Experiment in the $\mathcal{G}$-hybrid model. A simulator $\mathcal{S}$ attacks a single session of protocol φ running with an ideal subroutine $\mathcal{G}$, whereas an arbitrary "real" adversary $\mathcal{A}$ attacks a session of π running with an ideal subroutine $\mathcal{G}$. The dashed box encloses protocols where $\mathcal{S}$ or $\mathcal{A}$ control the network communications, whereas the solid lines represent a direct Input/Output relationship. (In a typical scenario, φ would be the ideal protocol for a desired functionality $\mathcal{F}$, whereas π would be a practical protocol realizing $\mathcal{F}$, with $\mathcal{G}$ modeling some "setup" functionality required by π. Observe that the environment can never interact directly with $\mathcal{G}$, and thus, in this particular scenario, $\mathcal{G}$ is never invoked at all in the ideal world since we are typically interested in the case where ideal protocol for $\mathcal{F}$ does not make use of $\mathcal{G}$.)

other hand, special considerations would arise if the challenge protocol utilizes (or produces) information that is also shared by other network protocol sessions. A typical example of such shared information involves the use of a global setup such as a public Common Reference String (CRS) or a standard Public Key Infrastructure (PKI). This kind of shared state is not *directly* modeled by the basic UC framework discussed above. In fact, the composition theorem of [19] only holds when considering instances of subroutine respecting protocols (which *do not* share any state information with other protocol sessions). Unfortunately, it is impossible to produce UC secure realizations of most useful functionalities without resorting to some kind of setup. Yet, to properly comply with the requirements of the UC framework, the setup would have to be done individually and independently for every protocol session! This does not faithfully represent the common realization, where the setup is done only once (particularly since setup

often involves large overhead) and is subsequently shared by all instances. To work around this limitation previous works handled such "shared state" protocol design situations via a special proof technique, known as the Joint-state Universal Composition Theorem (JUC Theorem) [31].

Intuitively, the JUC Theorem models shared state information by creating a single "multi-session" protocol, which internally emulates multiple sessions of all the protocols in the network that must share the information. (This is done by implementing all those protocols as "sub-sessions" of the multi-session protocol, which merely maintains the shared state information and dispatches any protocol request to a corresponding instance of the true destination protocol which it runs as a subroutine). Yet, even the the JUC Theorem does not accurately model truly *global* shared state information. The JUC Theorem only allows for the construction of protocols that share state *amongst themselves*. That is, an *a priori* fixed set of protocols can be proven secure if they share state information *only* with each other. No security guarantee is provided in the event that the shared state information is also used by other protocols which the original protocols were not specifically designed to interact with. Of course, malicious entities may take advantage of this by introducing new protocols that use the shared state information if the shared state is publicly available. In particular, protocols sharing global state (*i.e.*, using global setups) which are modeled in this fashion may not resist *adaptive chosen protocol attacks*, and can therefore suffer from a lack of deniability, as we previously mentioned regarding the protocols of [23], [30], and as is discussed in further detail in Chapter 3.

The Generalized UC Framework. To summarize the preceding discussion, the environment $\mathcal{Z}$ in the basic UC experiment is unable to invoke protocols that share state in any way with the challenge protocol. This limitation is unrealistic in the case of global setup, when protocols share state information with each other (and indeed, it was shown to be impossible to realize UC-secure protocols without resort to such tactics [23, 19, 29]). To overcome this limitation, we propose the Generalized UC (GUC) framework. The GUC challenge experiment is similar to the basic UC experiment, only with an *unconstrained* environment. In particular, we will allow $\mathcal{Z}$ to actually invoke and interact with arbitrary protocols, and even multiple sessions of the challenge protocol (since such multiple activations might conceivably be useful to $\mathcal{Z}$ in its efforts to distinguish between the two possible challenge protocols). Some of the protocol

sessions invoked by $\mathcal{Z}$ may even share state information with challenge protocol sessions, and indeed, those protocol sessions might provide $\mathcal{Z}$ with information related to the challenge protocol instances that it could not have obtained otherwise. The only remaining limitation on $\mathcal{Z}$ is that we prevent it from directly observing or influencing the network communications of the challenge protocol sessions, but this is naturally the job of the adversary (which $\mathcal{Z}$ is allowed to interact with). Thus, the GUC experiment allows a very powerful distinguishing environment capable of truly capturing the behavior of arbitrary protocol interactions in the network, *even if protocols can share state information with arbitrary other protocols.* Of course, protocols that are GUC secure are also composable (this fact follows almost trivially from a greatly simplified version of the composition theorem proof of [20], the simplifications being due to the ability of the unconstrained environment to directly invoke other protocol sessions rather than needing to "simulate" them internally).

The External-subroutine UC Framework. Unfortunately, since the setting of GUC is so complex, it becomes extremely difficult to prove the security of protocols in this new framework. Essentially, the distinguishing environment $\mathcal{Z}$ is granted a great deal of freedom in its choice of attacks, and any proof of protocol emulation in the GUC framework must hold even in the presence of other arbitrary protocols running concurrently. To simplify matters, we observe that in practice protocols which are designed to share state do so only in a very limited fashion (such as via a CRS, or a PKI, etc.). In particular, we will model shared state information via the use of "shared functionalities", which are simply functionalities that may interact with more than one protocol session (such as the CRS functionality). For clarity, we will distinguish the notation for shared functionalities by adding a bar (*i.e.*, we use $\bar{\mathcal{G}}$ to denote a shared functionality). We call a protocol π that *only* shares state information via a single shared functionality $\bar{\mathcal{G}}$ a $\bar{\mathcal{G}}$*-subroutine respecting* protocol. Bearing in mind that it is generally possible to model "reasonable" protocols that share state information as $\bar{\mathcal{G}}$-subroutine respecting protocols, we can make the task of proving GUC security simpler by considering a compromise between the constrained environment of basic UC and the unconstrained environment of GUC. An $\bar{\mathcal{G}}$*-externally constrained* environment is subject to the same constraints as the environment in the basic UC framework, only it is additionally allowed to invoke a single "external" protocol (specifically, the protocol for the shared

functionality $\bar{\mathcal{G}}$). Any state information that will be shared by the challenge protocol must be shared via calls to $\bar{\mathcal{G}}$ (*i.e.*, challenge protocols are $\bar{\mathcal{G}}$-subroutine respecting), and the environment is specifically allowed to access $\bar{\mathcal{G}}$. Although $\mathcal{Z}$ is once again constrained to invoking a single instance of the challenge protocol, it is now possible for $\mathcal{Z}$ to internally mimic the behavior of multiple sessions of the challenge protocol, or other arbitrary network protocols, by making use of calls to $\bar{\mathcal{G}}$ wherever shared state information is required. Thus, we may avoid the need for the JUC Theorem (and the implementation limitations it imposes), by allowing the environment direct access to shared state information (*e.g.*, we would allow it to observe the Common Reference String when the shared functionality is the CRS functionality). We call this new security notion External-subroutine UC (EUC) security, and we say that a $\bar{\mathcal{G}}$-subroutine respecting protocol π $\bar{\mathcal{G}}$-EUC-emulates a protocol φ if π emulates φ in the basic UC sense with respect to $\bar{\mathcal{G}}$-externally constrained environments.

We show that if a protocol π $\bar{\mathcal{G}}$-EUC-emulates φ, then it also GUC-emulates φ (and vice versa, provided that π is $\bar{\mathcal{G}}$-subroutine respecting).

Theorem 1.1. *Let π be any protocol which invokes no shared functionalities other than (possibly) $\bar{\mathcal{G}}$, and is otherwise subroutine respecting (*i.e.*, π is $\bar{\mathcal{G}}$-subroutine respecting). Then protocol π GUC-emulates a protocol φ, if and only if protocol π $\bar{\mathcal{G}}$-EUC-emulates φ.*

That is, provided that π only shares state information via a single shared functionality $\bar{\mathcal{G}}$, if it merely EUC-emulates φ with respect to that functionality, then π is a full GUC-emulation of φ! As a special case, we obtain that all basic UC emulations (which may not share *any* state information) are also GUC emulations.

Corollary 1.1. *Let π be any subroutine respecting protocol. Then protocol π GUC-emulates a protocol φ, if and only if π UC-emulates φ.*

The corollary follows by letting $\bar{\mathcal{G}}$ be the null functionality, and observing that the $\bar{\mathcal{G}}$-externally constrained environment of the EUC experiment collapses to become the same environment as that of the basic UC experiment when $\bar{\mathcal{G}}$ is the null functionality. Thus, it is sufficient to prove basic UC security for protocols with no shared state, or $\bar{\mathcal{G}}$-EUC security for protocols that share state only via $\bar{\mathcal{G}}$, and we will automatically obtain the full benefits of GUC security. The proof of the theorem is given in Section 1.2.

Figure 1.2 depicts the differences in the experiments of the UC models we have just described, in the presence of a single shared functionality $\bar{\mathcal{G}}$ (of course, the GUC framework is not inherently limited to special case of only one shared functionality). In Section 1.2 we elaborate the technical details of our new models, in addition to proving the equivalence of GUC and EUC security.

We are now in a position to state a strong new composition theorem, which will directly incorporate the previous result (that proving EUC security is sufficient for GUC security). Let ρ be an arbitrary protocol (not necessarily subroutine respecting!) which invokes φ as a sub-protocol. We will write $\rho^{\pi/\varphi}$ to denote a modified version of ρ that invokes π instead of φ, wherever ρ had previously invoked φ. We prove the following general theorem in Section 1.2 below:

Theorem 1.2 (Generalized Universal Composition)**.** *Let ρ, π, φ be PPT multi-party protocols, and such that both φ and π are $\bar{\mathcal{G}}$-subroutine respecting, and π $\bar{\mathcal{G}}$-EUC-emulates φ. Then $\rho^{\pi/\varphi}$ GUC-emulates protocol ρ.*

We stress that π must merely $\bar{\mathcal{G}}$-EUC-emulate φ, but that the resulting composed protocol $\rho^{\pi/\varphi}$ fully GUC-emulates ρ, even for a protocol ρ that is not subroutine respecting.

1.2 Details of the Generalized UC Framework

We now present more formal details of our new generalized UC (GUC) framework, and discuss its relationship to basic UC Security. (Here we will refer to the formulation of UC security in [20]). We also present a simplified variant of the new notion called External-subroutine UC (EUC), and prove its equivalence. Finally, we re-assert the universal composition theorem with respect to the new notion. Many of the low-level technical details, especially those that are essentially identical to those of the basic UC framework, are omitted. A full treatment of these details can be found in [20]. In particular, we do not discuss the proper modeling of polynomial runtime restrictions, the order of activations, etc. These issues are handled as in the basic UC framework, which we now briefly review.

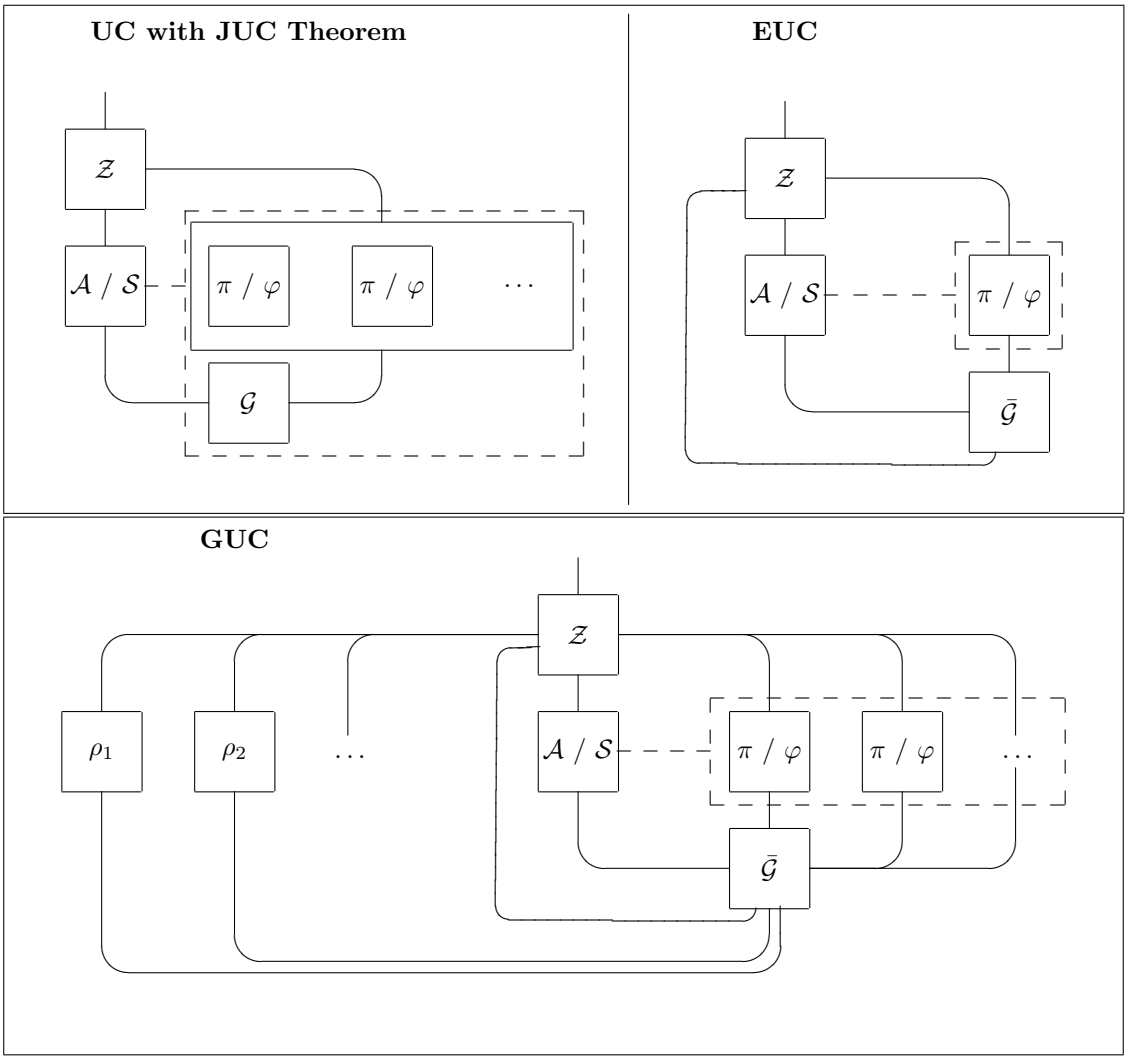

Figure 1.2: Comparison of security models. Using Basic UC with the JUC Theorem to share state, only copies of the challenge protocol (or other protocols which may be jointly designed *a priori* to share $\mathcal{G}$) are allowed to access the common subroutine $\mathcal{G}$, and $\mathcal{Z}$ may only interact with the "multi-session" version of the challenge protocol. In the EUC paradigm, only a single session of the challenge protocol is running, but the shared functionality $\bar{\mathcal{G}}$ it uses is accessible by $\mathcal{Z}$. Finally, in the GUC setting, we see the full generality of arbitrary protocols $\rho_1, \rho_2, \ldots$ running in the network, alongside multiple copies of the challenge protocol. Observe that both $\mathcal{Z}$, and any other protocols invoked by $\mathcal{Z}$ (such as ρ_1), have direct access to $\bar{\mathcal{G}}$ in the GUC setting. Intuitively, the GUC modeling seems much closer to the actual structure of networked protocol environments.

Basic UC Security

The basic UC framework is built around the notion of *UC emulation*. A protocol is a UC secure *realization* of an *ideal functionality* (which models the security goal), if it UC-emulates the ideal functionality, in the sense that executing the protocol is indistinguishable for an external environment from an interaction with a trusted party running the ideal functionality. Before reviewing the actual "UC experiment" that defines the notion of UC emulation, we first briefly review the basic model of distributed computation, using Interactive Turing Machines (ITMs). While we do not modify this model, familiarity with it is important for understanding our generalization.

Systems of ITMs. To capture the mechanics of computation and communication in computer networks, the UC framework employs an extension of the ITM model [55] (see [20] for precise details on the additional extensions). A computer program (such as for a protocol, or perhaps program of the adversary) is modeled in the form of an ITM (which is an abstract notion). An execution experiment consists of a *system* of ITMs which are instantiated and executed, with multiple instances possibly sharing the same ITM code. (More formally, a system of ITMs is governed by a *control function* which enforces the rules of interaction among ITMs as required by the protocol execution experiment. Here we will omit the full formalisms of the control function, which can be found in [20], and which require only minor modifications for our setting.)

A particular executing ITM instance running in the network is referred to as an ITI (or "ITM Instance"), and we must have a means to distinguish individual ITIs from one another even if they happen to be running identical ITM code. Therefore, in addition to the program code of the ITM they instantiate, individual ITIs are parameterized by a *party ID* (pid) and a *session ID* (sid). We require that each ITI can be uniquely identified by the *identity* pair $\mathsf{id} = (\mathsf{pid}, \mathsf{sid})$, irrespective of the code it may be running. All ITIs running with the same code and session ID are said to be a part of the same *protocol session*, and the party IDs are used to distinguish among the various ITIs participating in a particular protocol session. (By the uniqueness of ITI identities, no party is allowed to participate in more than one protocol session using the same session ID.) As a shorthand notation, we may write P to implicitly denote the pid of some particular (generally self-evident) ITI that is being run by P. We also refer to a a particular

protocol session running with ITM code π as an *instance* of protocol π.

ITMs are allowed to communicate with each other via the use of three kinds of I/O tapes: local *input tapes*, local *subroutine output* tapes, and *communication tapes*. The input and subroutine output tapes model "trusted communication", say, communication within a single physical computer. The communication tape models "untrusted communication", say, communication over an open network. Consequently, writes to the local input tapes of a particular ITI must include both the identity *and the code* of the intended target ITI, and the ITI running with the specified identity must also be running the specified code, or else an error condition occurs. Thus, input tapes may be used to invoke local "trusted" subroutines, and indeed, new ITIs must be introduced into the currently executing system by means of such invocations. That is, if a target ITI with the specified identity does not exist, it is created ("invoked"), and given the specified code. We also require that when an ITI writes to the local subroutine output tape of another ITI, it must provide its own code, and thus these tapes are useful for accepting output from such local "trusted" subroutines. Finally, all "untrusted" communications are passed via the communication tapes, which guarantee neither the code of the intended recipient ITI, nor the code of the sending ITI (but merely their identities).

The UC Protocol Execution Experiment. The UC protocol execution experiment is defined as a system of ITMs that is parameterized by three ITMs. An ITM π specifies the code of the *challenge protocol* for the experiment, an ITM $\mathcal{A}$ specifies the code of the *adversary*, and an ITM $\mathcal{Z}$ provides the code of the *environment*. The protocol execution experiment places precise conditions on the order in which ITIs are activated, and which ITIs are allowed to invoke or communicate with each other. The precise formal details of how these conditions are defined and imposed (*i.e.*, the control function and related formalisms) can be found in [20], but we shall describe some of the relevant details informally. The experiment initially launches only an ITI running $\mathcal{Z}$. In turn, $\mathcal{Z}$ is permitted to invoke only a single ITI running $\mathcal{A}$, followed by (multiple) ITIs running the "challenge protocol" π *provided that those ITIs running* π *all share the same* sid. This sid, along with the pids of all the ITIs running π, may be chosen arbitrarily by $\mathcal{Z}$.

It is stressed that the environment *may not* invoke any additional ITIs, and it is only allowed to write to the input tapes of ITIs which it has directly invoked (or to receive outputs from

those ITIs via its subroutine output tape). The environment may not interact with *any* of the communication tapes, nor the tapes of ITIs that it did not directly invoke. In summary, the environment can communicate only with the ITI running the code of the adversary $\mathcal{A}$, and ITIs participating in a single session of protocol π. We thus refer to the execution experiment as being a *constrained* one, and, in particular, the environment $\mathcal{Z}$ as being a *constrained environment*. The output of the environment $\mathcal{Z}$ in this basic UC protocol execution experiment is denoted by $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

Ideal Functionalities. We say an ITM $\mathcal{F}$ is an *ideal functionality* if its code represents a desired (interactive) function to be computed by parties or other protocols which may invoke it *as a subroutine* (and thus, in a perfectly secure way). The pid of any ITI running $\mathcal{F}$ is set to the special value $\perp$, indicating that the ITI is an ideal functionality. $\mathcal{F}$ accepts input from other ITIs that have the same sid as $\mathcal{F}$, and may write outputs to multiple ITIs as well.

Every ideal functionality $\mathcal{F}$ also induces an *ideal protocol* $\text{IDEAL}_{\mathcal{F}}$. Parties running $\text{IDEAL}_{\mathcal{F}}$ with the session ID sid act as *dummy parties*, simply forwarding their inputs to the input tape of an ITI running $\mathcal{F}$ with the same sid, and copying any subroutine output received from $\mathcal{F}$ to the subroutine output tape of the ITI which invoked the party (typically, the environment $\mathcal{Z}$).

UC Emulation and Realizations. In the basic UC framework, a protocol π is said to *UC-emulate* another protocol φ if, for any adversary $\mathcal{A}$, there exists a *simulator* $\mathcal{S}$ such that for all environments $\mathcal{Z}$ it holds that $\text{EXEC}_{\varphi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. That is, no environment behaves significantly differently in the protocol execution experiment when interacting with the challenge protocol π, under any given attack, than it does when interacting with the challenge protocol φ, under a simulation of that same attack. Intuitively, this means that the protocol π is "at least as secure as" the protocol φ, since the effect of any attack on π can also be emulated by attacking φ. A protocol π is said to *UC-realize* an ideal functionality $\mathcal{F}$ if π UC-emulates $\text{IDEAL}_{\mathcal{F}}$. Furthermore, if the protocol π is a $\mathcal{G}$-hybrid protocol, then we say that π is a UC-secure realization of $\mathcal{F}$ in the $\mathcal{G}$-hybrid model.

Generalized UC Security

We present the generalized variant of UC security. Technically, the difference is very small; however, its effect is substantial. The essential difference here from the basic UC security notion is that in our model, the environment $\mathcal{Z}$ is allowed to invoke ITIs with arbitrary code and arbitrary sids, including multiple concurrent instances of the challenge protocol π and other protocols. We stress that these ITIs are even allowed to *share state* with each other across multiple sessions (which is a significant departure from prior models). To simplify the presentation and analysis, we will still assume that $\mathcal{Z}$ invokes only a single instance of the adversary $\mathcal{A}$.[1] We call such an environment *unconstrained*, since it need not obey any constraints in regards to which protocols it may invoke.[2] To distinguish this from the basic UC experiment, we denote the output of an unconstrained environment $\mathcal{Z}$, running with an adversary $\mathcal{A}$ and a challenge protocol π in the GUC protocol execution experiment, by $\text{GEXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. GUC emulation is now defined as follows, analogously to the definition of basic UC emulation outlined above:

Definition 1.3 (GUC-Emulation)**.** Let π and φ be PPT multi-party protocols. We say that π GUC-emulates φ if, for any PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any (unconstrained) PPT environment $\mathcal{Z}$, we have:

$$\text{GEXEC}_{\varphi,\mathcal{S},\mathcal{Z}} \approx \text{GEXEC}_{\pi,\mathcal{A},\mathcal{Z}}$$

As long as the protocol in question makes sure that no instance shares any subroutines with other protocol instances, GUC security is equivalent to basic UC security. This statement, which intuitively follows from the "universal composition theorem", will be formalized later. However, we are primarily interested in protocols that *do* share some modules, or subroutines, with other

[1] Although it is conceptually interesting to consider scenarios where the environment may invoke separate adversaries to attack separate instances of the challenge protocol, particularly when there is some shared state, it can be shown that this notion is equivalent to our simplified single adversary model.

[2] More formally, the control function for the GUC protocol execution experiment allows $\mathcal{Z}$ to invoke ITIs running arbitrary ITM code, and with arbitrary (but unique) identities. In particular, the environment may invoke many ITIs with a special code $\perp$ (even with different sids), which the control function will substitute for ITIs running π. Thus, the reason for our departure with the convention of [20] (which replaces the code of all ITIs invoked by $\mathcal{Z}$ with instances of π) is to provide $\mathcal{Z}$ with the ability to invoke ITIs running arbitrary code (other than π), yet still enable it to invoke instances of the challenge protocol without having access to its code.

protocol instances. For such protocols the generalized formulation differs radically from the basic one. Specifically, we are interested in modeling "shared trusted modules", which are captured via the *shared functionality* construct.

Shared Functionalities. In addition to the notion of ideal functionalities inherited from the basic UC security setting, we coin the notion of *shared functionalities*. A shared functionality $\bar{\mathcal{G}}$ is completely analogous to an ideal functionality, except that it can accept inputs from ITIs with *arbitrary* session IDs. Thus, a shared functionality is just an ideal functionality that may communicate with more than one protocol session. In order to distinguish shared functionalities from ideal functionalities, we require the sid of a shared functionality to begin with the special symbol # (which is used exclusively in the sid of shared functionalities). As a shorthand notation, we denote the portion of the sid of a shared functionality which follows the # symbol by shsid (and thus, shared functionalities have $\mathsf{sid} = \#\|\mathsf{shsid}$). Similarly to ideal functionalities, shared functionalities also have # as their fixed pid, and thus all shared functionalities have fixed identities (and can be invoked only by specifying the code of their corresponding ITM).

Discussion. Recall that in the basic UC definition, $\mathcal{Z}$ is constrained in its ability to invoke protocols. The environment is allowed to invoke a single instance of each party who is participating in a *solitary session* of the challenge protocol, as well as a single instance of the adversary attacking the protocol. However, in basic UC, $\mathcal{Z}$ may not invoke or communicate with other ITIs of any sort (*e.g.*, to represent other parties and protocol sessions concurrently running in the network). Intuitively, it would seem that if this constraint were removed and $\mathcal{Z}$ were allowed to invoke arbitrary ITIs running arbitrary protocols (including multiple concurrent sessions of the challenge protocol itself), then $\mathcal{Z}$ would become a more powerful distinguisher (strengthening the security requirements for protocols to remain indistinguishable). In reality, as we will soon see, since basic UC security does not allow protocols to *share state*[3] with each other, any concurrent protocol executions that $\mathcal{Z}$ might wish to run can simply be simulated by $\mathcal{Z}$ internally with no need to actually invoke the ITIs. Thus, the constraint that $\mathcal{Z}$ may only invoke parties running a

[3]The typical example of protocols sharing state occurs when using the CRS model. Multiple instances of a protocol may all share the same CRS, which certainly implies a relationship between those protocol executions which is not captured by standard UC security.

single instance of the protocol it is attempting to distinguish is not a true limitation, and indeed this is where the power of UC security comes from (*e.g.*, the ability for $\mathcal{Z}$ to conduct this kind of internal simulation is the reason UC security holds even in the presence of concurrent protocol executions).

Unlike this basic UC security setting, we wish to consider definitions of security even for protocols that may share state information externally with other (concurrently executing) sessions of the same protocol, or even with other (independently designed) protocols. In such a setting, it is no longer possible for $\mathcal{Z}$ to simulate other protocol executions internally, since some of those protocols may share state with the protocol that $\mathcal{Z}$ is attempting to distinguish. Thus, the constraints placed on $\mathcal{Z}$ in the basic UC setting are of great concern for us, since they would prevent $\mathcal{Z}$ from ever seeing the effects of other protocol executions that share state externally with the protocol $\mathcal{Z}$ is attempting to distinguish. Of course, protocol executions in the real world certainly do involve such effects, as we will see in Chapter 3. We introduced the notion of Generalized UC (GUC) security in order to address this issue by properly capturing interactions between protocols that share state information externally.

External-subroutine UC Security

Since the unconstrained environment of the GUC security model is able to invoke arbitrary ITIs, as well as multiple sessions of the challenge protocol, it becomes difficult to directly prove that one protocol GUC-emulates another protocol, *i.e.*, to show that a simulated adversary $\mathcal{S}$ attacking a protocol φ behaves indistinguishably from an actual adversary $\mathcal{A}$ attacking protocol π. In particular, such analysis requires direct argumentation about systems where multiple instances of multiple protocols run concurrently. This stands in contrast to the situation with basic UC security, where it suffices to analyze a single instance of the protocol in isolation, and security in a multi-instance system follows from a general composition theorem.

We alleviate this situation in two steps. First, we formulate another notion of protocol emulation, called External-subroutine UC (EUC) emulation, which is considerably simpler, and in particular considers only a single instance of the challenge protocol. We show that this simplified notion is equivalent to the GUC notion described above. In a second step, we use the new EUC-emulation notion to re-assert the universal composition theorem with respect to GUC-emulation.

We remark that in the basic UC framework these two conceptual steps are demonstrated via the same technical theorem (namely, the UC theorem). We find that in the present framework it is clearer to separate the two issues.

Subroutine respecting protocols. Before proceeding to define External-subroutine UC emulation, we coin the following terminology. We say that an ITI M is a *subroutine* of another ITI M' if M either receives inputs on its input tape from M' (and does not explicitly ignore them), or writes outputs to the subroutine output tape of M'. Recursively, we also say that if M is a subroutine of a party (ITI) running protocol π or a *sub-party* of protocol π, then M is a *sub-party* of protocol π. By uniqueness of session identifiers, if there is an instance of protocol π running with session ID sid, all ITIs running with session ID sid are running π or are sub-parties of π.

A protocol π is said to be $\bar{\mathcal{G}}$-*subroutine respecting* if none of the sub-parties of an instance of π provides output to or receives input from any ITI that is not also party/sub-party of that instance of π, *except* for communicating with a *single instance* of the shared ITI $\bar{\mathcal{G}}$. In other words, an instance of a $\bar{\mathcal{G}}$-subroutine respecting protocol π has the property that all sub-parties of this instance of π are only allowed to communicate with parties or sub-parties of this same instance of π (they do not share themselves with other protocol instances in any way), with the sole exception that calls to a shared functionality $\bar{\mathcal{G}}$ are allowed. Using this terminology, we can now define External-subroutine UC emulation.

The External-subroutine UC Protocol Execution Experiment. Rather than allowing the environment to operate completely unconstrained as in the GUC experiment, we constrain the environment of the EUC experiment so that it may only invoke particular types of ITIs. Specifically, the environment is only allowed to invoke parties participating in a *single* instance of the challenge protocol (as in the constrained environment of basic UC), plus a *single* ITI running the code of a shared functionality $\bar{\mathcal{G}}$. In other words, the EUC experiment is the same as the basic UC experiment, except that the environment (which is otherwise constrained in the usual fashion) is also allowed to provide input to, and obtain output from, a single instance of a shared functionality. (Typically, the code of this shared functionality would be specified by the challenge protocol under consideration.) We say that such an environment is $\bar{\mathcal{G}}$-*externally*

constrained when it is allowed such extra access to a shared functionality $\bar{\mathcal{G}}$. (Note that although we consider only one shared functionality at a time for the sake of simplicity, it is also reasonable to define the notions of "subroutine respecting" and "EUC security" with respect to multiple shared functionalities.) Given a $\bar{\mathcal{G}}$-subroutine respecting protocol π, we denote the output of the environment in the $\bar{\mathcal{G}}$-EUC protocol experiment by $\text{EXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A},\mathcal{Z}}$. EUC-emulation is defined analogously to the notion of GUC-emulation:

Definition 1.4 (EUC-Emulation)**.** Let π and φ be PPT multi-party protocols, where π is $\bar{\mathcal{G}}$-subroutine respecting. We say that π EUC-emulates φ with respect to shared functionality $\bar{\mathcal{G}}$ (or, in shorthand, that π $\bar{\mathcal{G}}$-EUC-emulates φ) if for any PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any $\bar{\mathcal{G}}$-externally constrained environment $\mathcal{Z}$, we have:

$$\text{EXEC}^{\bar{\mathcal{G}}}_{\varphi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A},\mathcal{Z}}$$

$\bar{\mathcal{G}}$-EUC Secure Realization. We say that a protocol π *realizes* an ideal functionality $\mathcal{F}$ if π $\bar{\mathcal{G}}$-EUC-emulates IDEAL_F. Notice that the formalism implies that the shared functionality $\bar{\mathcal{G}}$ exists both in the model for executing π *and also in the model for executing the ideal protocol for* $\mathcal{F}$, IDEAL_F.

We remark that the notion of $\bar{\mathcal{G}}$-EUC-emulation can be naturally extended to protocols that use several different shared functionalities (instead of only one).

Equivalence of GUC to EUC and a generalized UC theorem

We show that $\bar{\mathcal{G}}$-EUC-emulation is, surprisingly, equivalent to full GUC-emulation for any $\bar{\mathcal{G}}$-subroutine respecting protocol. Perhaps unsurprisingly, the proof of the equivalence theorem incorporates most of the arguments of the universal composition theorem. In particular, the "quality" of security degrades linearly with the number of instances of π invoked by $\mathcal{Z}$ in the GUC experiment.

The formal statement of equivalence appears in Theorem 1.1 above. The proof of the theorem, which we now give, makes use of a hybrid argument (akin to that in the universal composition theorem of [20]) to show that security for the single-instance setting of EUC is sufficient to ensure security under the more strenuous multi-instance setting of GUC.

Proof of Theorem 1.1. It is trivial to show that protocols which GUC-emulate each other also $\bar{\mathcal{G}}$-EUC-emulate each other, since any simulation that is indistinguishable to unconstrained environments is certainly indistinguishable to the special case of $\bar{\mathcal{G}}$-externally constrained environments as well. However, the other direction is non-obvious. The basic idea of the proof is that a $\bar{\mathcal{G}}$-externally constrained environment can simulate the same information available to an unconstrained environment, even while operating within its constraints. The proof technique is essentially the same as the proof of composition theorem, only in this case multiple sessions must be handled directly without going through an intermediate protocol. (That is, the proof of composition theorem considers a protocol π which emulates a protocol φ, and shows that a protocol ρ which may invoke multiple copies of π emulates a protocol ρ which invokes φ instead. Here, we essentially need to demonstrate that if a single copy of π emulates φ, then multiple copies of π emulate multiple copies of φ.)

Applying the technique of [20], we observe that there are equivalent formulations of protocol emulation with respect to dummy adversaries (this must be proven separately, but the proofs are essentially identical to those for the original notion of UC emulation), and we will use those alternative formulations here to simplify the proof. Let $\mathcal{D}$ denote the fixed "dummy adversary" (which simply forwards messages to and from the environment). For the remainder of the proof, we shall refer to the "simulator $\mathcal{S}$" as the "ideal adversary", in order to avoid confusion with the "internal simulation" being conducted by the environment (roughly speaking, $\mathcal{S}$ attempts to mimic real attacks by attacking the ideal functionality, so this terminology is appropriate).

Suppose that π $\bar{\mathcal{G}}$-EUC-emulates φ. Then there exists some ideal adversary $\mathcal{S}$ that will satisfy $\text{EXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{D},\mathcal{Z}} \approx \text{EXEC}^{\bar{\mathcal{G}}}_{\varphi,\mathcal{S},\mathcal{Z}}$ for any $\bar{\mathcal{G}}$-externally constrained environment $\mathcal{Z}$. To prove our claim, we will need to show that the existence of such $\mathcal{S}$ is sufficient to construct a new ideal adversary $\tilde{\mathcal{S}}$ such that $\text{GEXEC}_{\pi,\mathcal{D},\tilde{\mathcal{Z}}} \approx \text{GEXEC}_{\varphi,\tilde{\mathcal{S}},\tilde{\mathcal{Z}}}$ holds for any unconstrained environment $\tilde{\mathcal{Z}}$.

We construct an ideal adversary $\tilde{\mathcal{S}}$ in a similar fashion to the construction of $\mathcal{A}_\pi$ in the proof of composition theorem in [20], using multiple instances of $\mathcal{S}$ that are simulated by $\mathcal{A}_\pi$ internally. That is, to ensure that each instance of φ mimics the corresponding instance of π, $\tilde{\mathcal{S}}$ will run separate copies of $\mathcal{S}$ for each instance of π (and $\tilde{\mathcal{S}}$ then forwards messages between $\tilde{\mathcal{Z}}$ and the appropriate copy of $\mathcal{S}$ for each instance). We now prove that $\tilde{\mathcal{S}}$ satisfies the requirement for GUC-emulation via a hybrid argument (again, as is done in the proof of composition theorem).

Assume for the purpose of contradiction that $\text{EXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{D},\tilde{\mathcal{Z}}} \not\approx \text{EXEC}^{\bar{\mathcal{G}}}_{\varphi,\tilde{\mathcal{S}},\tilde{\mathcal{Z}}}$ (in particular, assume the distinguishing advantage of $\tilde{\mathcal{Z}}$ is ε). Let m be an upper bound on the number of instances of π which are invoked by $\tilde{\mathcal{Z}}$. For $l \leq m$, let $\tilde{\mathcal{S}}_l$ denote the ideal adversary for a modified execution $\text{EX}_l = \text{EXEC}(l,\varphi)^{\bar{\mathcal{G}}}_{\pi,\tilde{\mathcal{S}}_l,\tilde{\mathcal{Z}}}$ in which the first l instances of π are simulated[4] using instances of $\mathcal{S}$ and φ (as would be done by $\tilde{\mathcal{S}}$), but the remaining invocations of π by $\tilde{\mathcal{Z}}$ are in fact handled by genuine instances of π (with $\tilde{\mathcal{S}}_l$ simply forwarding messages directly to and from those instances, as $\mathcal{D}$ would). In particular, we observe that the modified interaction EX_0 is just the interaction with π, and EX_l is the unmodified interaction with $\tilde{\mathcal{S}}$ and φ replacing $\mathcal{D}$ and π. Then, by our assumption that the interactions with EX_0 and EX_m are distinguishable, there must be an $0 < l \leq m$ such that $\tilde{\mathcal{Z}}$ distinguishes between the modified interactions with EX_l and EX_{l-1} with advantage at least ε/m. We can construct an $\bar{\mathcal{G}}$-externally constrained environment $\mathcal{Z}^*$ from such a $\tilde{\mathcal{Z}}$ which succeeds in distinguishing the ensembles $\text{EXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{D},\mathcal{Z}^*}$ and $\text{EXEC}^{\bar{\mathcal{G}}}_{\varphi,\mathcal{S},\mathcal{Z}^*}$ with probability at least ε/m, contradicting the fact that π $\bar{\mathcal{G}}$-EUC-emulates φ.

The construction of $\mathcal{Z}^*$ is slightly involved, but on a high level, $\mathcal{Z}^*$ internally simulates the actions of $\tilde{\mathcal{Z}}$ including all ITIs activated by $\tilde{\mathcal{Z}}$ (other than those for $\bar{\mathcal{G}}$ and the l-th instance of π), but forwards all communications sent to the l-th instance of π to its own external interaction instead (which is either with a single instance of π and $\mathcal{D}$, or φ and $\mathcal{S}$). We observe that since π is subroutine respecting, the only way ITIs activated by the simulated $\tilde{\mathcal{Z}}$ may somehow share state information with the challenge instance of π is via access to the shared functionality $\bar{\mathcal{G}}$. Whenever an ITI invoked by the internal simulation of $\tilde{\mathcal{Z}}$ wishes to communicate with $\bar{\mathcal{G}}$, $\mathcal{Z}^*$ invokes a corresponding dummy party with the same pid and sid, and then forwards communications between the internally simulated ITI and the actual shared functionality $\bar{\mathcal{G}}$ in $\mathcal{Z}^*$'s external interaction via the dummy party. $\mathcal{Z}^*$ then outputs whatever the internally simulated copy of $\tilde{\mathcal{Z}}$ outputs. This interaction results in the simulated $\tilde{\mathcal{Z}}$ operating with a view that corresponds to either EX_{l-1} or EX_l (which $\tilde{\mathcal{Z}}$ can distinguish with probability at least ε/m), and thus $\mathcal{Z}^*$ successfully distinguishes with probability at least ε/m, as claimed, completing the contradiction.

□

[4]Technically, we *must* modify the execution experiment here, since it is the environment which attempts to invoke the challenge protocol π, which is beyond the control of the ideal adversary $\tilde{\mathcal{S}}$. Thus l and φ need to be specified as part of the execution experiment itself.

We observe that if a protocol π does not use any shared functionalities (*i.e.*, π is $\bar{\mathcal{G}}$-subroutine respecting for a null functionality that generates no output) then a corollary of the above claim states that π UC-emulates φ if and only if π GUC-emulates φ. This equivalence shows the power of the basic UC emulation security guarantee, since it is indeed equivalent to the seemingly stronger notion of GUC emulation (for any protocols which exist in the more limited basic UC setting).

Universal Composition. Finally, we generalize the universal composition theorem to hold also with respect to GUC-emulation. That is, consider a $\bar{\mathcal{G}}$-subroutine respecting protocol φ that is being used as a subroutine in some (arbitrary) larger protocol ρ. The new composition theorem guarantees that it is safe to replace a protocol φ with a different protocol π that merely $\bar{\mathcal{G}}$-EUC-emulates φ, and yet the resulting implementation of ρ (which now invokes π instead of φ) will fully GUC-emulate the original version (which had invoked φ).

The formal composition theorem appears in Theorem 1.2 above, which we now prove. The proof is similar in spirit to the proof of universal composition theorem (but here we no longer require the hybrid argument, since multiple protocol instances are already taken care of by the GUC setting).

Proof of Theorem 1.2. Since the notions of $\bar{\mathcal{G}}$-EUC-emulation and GUC-emulation are equivalent for subroutine respecting protocols which do not use shared functionalities other than $\bar{\mathcal{G}}$, it suffices to prove that if π GUC-emulates φ then $\rho^{\pi/\varphi}$ GUC-emulates ρ. (Of course, there is a corresponding loss of exact security as per Theorem 1.1.) Thus, it suffices to prove that the composition theorem holds for subroutine respecting protocols that GUC-emulate each other. As before, we shall refer to the "simulator $\mathcal{S}$" as the "ideal adversary", in order to avoid confusion.

The proof that GUC-emulation is composable follows the same general approach as the composition theorem for basic UC in [20], with some simplifications resulting from the use of unconstrained environments. We once again make use of an equivalent formulation of GUC-emulation with respect to dummy adversaries (the proof of equivalence for the GUC version of the dummy adversary formulation is also entirely analogous to the proof of the same statement for basic UC security). Thus, denoting the dummy adversary by $\mathcal{D}$, we wish to construct an adversary $\mathcal{A}_\rho$

such that

$$\text{GEXEC}_{\rho^{\pi/\varphi},\mathcal{D},\mathcal{Z}} \approx \text{GEXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}} \tag{1.1}$$

for any unconstrained environment $\mathcal{Z}$.

Since π GUC-emulates φ there is an adversary $\mathcal{S}$ such that $\text{GEXEC}_{\pi,\mathcal{D},\mathcal{Z}_\pi} \approx \text{GEXEC}_{\varphi,\mathcal{S},\mathcal{Z}_\pi}$ for any unconstrained environment $\mathcal{Z}_\pi$. That is, $\mathcal{S}$ expects to interact with many instances of φ, with the goal of translating them to mimic the action of corresponding instances of π from the viewpoint of any environment $\mathcal{Z}_\pi$. We will use $\mathcal{S}$ to construct $\mathcal{A}_\rho$ satisfying (1.1) above. Unlike the construction in the basic UC composition theorem, it is not necessary for $\mathcal{A}_\rho$ to run multiple copies of $\mathcal{S}$ (one for each session of π), since the GUC adversary $\mathcal{S}$ already deals with the scenario where multiple sessions of π are executing (as unconstrained environments may invoke multiple instances of the challenge protocol). Thus the construction of $\mathcal{A}_\rho$ here is simpler.

$\mathcal{A}_\rho$ will simply run a single copy of $\mathcal{S}$ internally, forwarding all messages intended for instances of π (which are sub-parties to instances of the challenge protocol ρ) sent by $\mathcal{A}_\rho$'s environment $\mathcal{Z}$ to its internal simulation of $\mathcal{S}$, as well as forwarding any messages from $\mathcal{S}$ back to $\mathcal{Z}$ as is appropriate. (Note that $\mathcal{A}_\rho$ need not simulate any interactions with instances of π that are invoked directly by $\mathcal{Z}$ rather than an instance of ρ, since those are not associated with the challenge protocol.) Additionally, $\mathcal{A}_\rho$ forwards $\mathcal{S}$'s interactions with instances of φ between the external instances of φ (again, only those which are sub-parties to instances of the challenge protocol ρ) and its internal copy of $\mathcal{S}$ as well. (Intuitively, $\mathcal{A}_\rho$ acts as the environment for $\mathcal{S}$ by forwarding some of its own interactions with $\mathcal{Z}$ concerning instances of π, and also copying its own interactions with external instances of φ. The reason $\mathcal{S}$ is not employed directly in place of $\mathcal{A}_\rho$ is that $\mathcal{A}_\rho$ must translate sessions of the "challenge protocol" ρ in order to isolate their sub-protocol invocations of φ, which is the challenge protocol that $\mathcal{S}$ expects to interact with. Thus, effectively, the instances of ρ itself simply become part of the environment for $\mathcal{S}$. Note that there may be many instances of ρ which are being translated, and each of those instances may invoke many instances of φ.)

In order to prove that $\mathcal{A}_\rho$ satisfies (1.1) we perform a standard proof by contradiction. Assume there exists an environment $\mathcal{Z}$ capable of distinguishing the interaction with $\mathcal{A}_\rho$ and ρ from the interaction with $\mathcal{D}$ and $\rho^{\pi/\varphi}$. We show how to construct an environment $\mathcal{Z}_\pi$ capable of

distinguishing an interaction between $\mathcal{S}$ and φ from an interaction with $\mathcal{D}$ and π, contradicting the fact that π GUC-emulates φ.

The construction of $\mathcal{Z}_\pi$ is again analogous to the technique applied in [20], with some additional simplification (*e.g.*, since there is no hybrid argument here, we may simply treat all instances of φ the same way). As a useful tool in describing the construction of $\mathcal{Z}_\pi$, we briefly define an "internal simulation adversary" $\hat{\mathcal{A}}_\rho$, which will be run internally by $\mathcal{Z}_\pi$ alongside an internally simulated copy of $\mathcal{Z}$. Whenever $\hat{\mathcal{A}}_\rho$ receives a message, it performs the same function as $\mathcal{A}_\rho$, only replacing $\mathcal{A}_\rho$'s communications with its internal simulation of the adversary $\mathcal{S}$ (along with its corresponding challenge protocol φ) by communications with the external adversary for $\mathcal{Z}_\pi$ (and its corresponding challenge protocol, which will either be π if the adversary is $\mathcal{D}$ or φ if the adversary is $\mathcal{S}$). We observe that if $\mathcal{Z}_\pi$'s adversary is $\mathcal{D}$, then $\hat{\mathcal{A}}_\rho$ also acts like $\mathcal{D}$, since it will merely forward all messages. Similarly, if $\mathcal{Z}_\pi$'s external adversary is $\mathcal{S}$, then $\hat{\mathcal{A}}_\rho$ will function identically to $\mathcal{A}_\rho$.

On a high level, the environment $\mathcal{Z}_\pi$ will internally simulate the environment $\mathcal{Z}$ and adversary $\hat{\mathcal{A}}_\rho$, externally invoking copies of any ITIs that are invoked by the simulations (with the exception of instances of $\mathcal{Z}$'s challenge protocol π, and any invocations of $\mathcal{Z}_\pi$'s challenge protocol made by $\hat{\mathcal{A}}_\rho$), appropriately forwarding any messages between those ITIs and its internal copies of $\mathcal{Z}$ and $\mathcal{A}_\rho$. Whenever the internal copy of $\mathcal{Z}$ wishes to invoke an instance of the challenge protocol ρ, the environment $\mathcal{Z}_\pi$ internally simulates the instance (by modeling an ITI running ρ with the specified identity), forwarding any communications between ρ and shared functionalities to external instances of those shared functionalities (such forwarding may be accomplished by $\mathcal{Z}_\pi$ externally invoking dummy parties with the same identities as the sub-parties of ρ that wish to communicate with the shared functionalities, and then forwarding communications through the dummy parties). Because ρ is subroutine respecting, it is safe to conduct such an internal simulation, as instance of ρ do not share state with any ITIs external to $\mathcal{Z}_\pi$ except via the shared functionalities (which are handled appropriately). Of course, whenever the internal simulation of $\hat{\mathcal{A}}_\rho$ wishes to communicate with an instance of its challenge protocol, $\mathcal{Z}_\pi$ will forward the communications to the correct instance of its own challenge protocol, as described above. When the internally simulated $\mathcal{Z}$ halts and provides output, $\mathcal{Z}_\pi$ similarly halts, copying the same output.

Now, we can observe that $\text{GEXEC}_{\pi,\mathcal{D},\mathcal{Z}_\pi} = \text{GEXEC}_{\rho^{\pi/\varphi},\mathcal{D},\mathcal{Z}}$ by considering that the internal

simulation conducted by $\mathcal{Z}_\pi$ will be a faithful recreation of the latter experiment. In particular, by its construction, the simulation of $\hat{\mathcal{A}}_\rho$ will simply act as the dummy adversary $\mathcal{D}$. Furthermore, the internal simulation of ρ is correctly replacing all invocations of φ by invocations of π (and thus is a perfect simulation of $\rho^{\pi/\varphi}$), while the rest of the experiment proceeds identically. A similar argument yields that $\text{GEXEC}_{\varphi,\mathcal{S},\mathcal{Z}_\pi} = \text{GEXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}}$. Previously, we assumed for the sake of contradiction that there exists an environment $\mathcal{Z}$ such that $\text{GEXEC}_{\rho^{\pi/\varphi},\mathcal{D},\mathcal{Z}} \not\approx \text{GEXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}}$. Combining this statement with the previous two equations yields the result that there exists an environment $\mathcal{Z}_\pi$ such that $\text{GEXEC}_{\pi,\mathcal{D},\mathcal{Z}_\pi} \not\approx \text{GEXEC}_{\varphi,\mathcal{S},\mathcal{Z}_\pi}$, contradicting the fact that π GUC-emulates φ, completing our proof. □

2 • Preliminaries and Tools

In preparation for the following chapters, we now present a collection of useful concepts, notations, definitions, and tools. While many of the tools discussed herein are tightly coupled to applications that first appear much later in the thesis, they are presented here together in order to assist the reader in identifying the relationships among the various tools and definitions. The reader is strongly advised to refer back to this chapter upon encountering the relevant topics later in the text.

2.1 Adversarial Models

Here we outline some additional details of our adversarial model that are not pre-specified by the GUC framework described in Chapter 1. Indeed, the reason we often refer to GUC as a "framework" rather than as a "model" is that it is flexible enough to encompass many possible models of adversarial behavior.

2.1.1 Corruption Models

In this work, we will consider several kinds of *corruption models*, which model attackers of varying strengths and capabilities. When providing feasibility results, we strive to provide security against the strongest possible class of attacks. Conversely, impossibility results carry the most weight when they hold even against weaker attacks, and therefore we will strive to use the weakest possible attacks when proving infeasibility.

The strongest adversarial model we will consider allows *adaptive corruptions*: the attacker is allowed to corrupt parties before, during, and after protocol executions. This means that the adversary can eventually corrupt all the parties that were involved in any given protocol execution. Furthermore, we assume that honest parties who are corrupted will provide a complete history of their internal state to the adversary – that is, we do not rely on the ability of honest parties to erase information. This is known as the *non-erasure* model. A slight weakening of this attack model permits honest parties to explicitly erase some "temporary" internal state

information, and this variant is therefore referred to as the *erasure model*. In practice, it is often possible to erase small quantities of short-lived data very effectively, which can justify use of the erasure model in some circumstances.

We also introduce a slightly weaker corruption model, in which we allow for *semi-adaptive corruptions*: the adversary is allowed to corrupt parties either before or after a particular round of the protocol execution has completed, but may not corrupt parties while that stage of the protocol is ongoing (for either party). The basic motivation for this model is that it seems highly unlikely that any attacker can manage to corrupt an honest party during the narrow window of time in which this round of the protocol execution occurs. (In practice, the window of vulnerability can be made quite short by introducing appropriate "timeouts" into the protocol.) This model is also closely related to the notion of *forward security*, in which corruptions that occur after the completion of a protocol do not affect its security (or that of any prior executions). Once again, we may consider both erasure and non-erasure variants of the semi-adaptive and forward security models.

Finally, the weakest adversarial model that we consider allows only *static corruptions*: the attacker is only allowed to corrupt *some* of the parties *before* the protocol has begun. In this corruption model we no longer need to distinguish between erasure and non-erasure variants, since any parties who are honest at the start of a protocol execution can never be corrupted and will never reveal their internal state to the adversary (therefore, it is irrelevant whether or not they are able to erase parts of their internal state).

In all cases, we will assume that adversarial corruption occurs on a "per party" basis – that is, either a particular party is corrupt (acting adversarially) in *all* of its protocol sessions, or it is honest in *all* of its protocol sessions. We will refer to this aspect of our modeling by saying that all corruptions must occur in a *PID-wise* manner.[1]

[1]Previous works in the UC framework generally allow corruptions of a party to occur in an individual session. Such modeling is straightforward when the entire state of the protocol is self-contained in a single session, but can unnecessarily complicated matters when some shared functionality is involved. For instance, if a party uses the same secret keys for "honest" sessions that it uses in "corrupt" sessions, it can obviously harm the security of its own "honest" sessions that way.

2.1.2 Security with Trusted Parties

Our protocols will often have to make use of some mutually trusted third party, such as a certification authority. In the real world, trust is often hard to come by. Therefore, we would like to place reasonable limits on the trust we require, and to distribute that trust as much as possible.

Of course, even when we are willing to place trust in some particular entity, we might not want to trust that entity for a very long time. For example, we might be concerned that a trusted party will eventually be corrupted by the adversary – either because the trusted party lacks adequate security measures (and is therefore subject to being "hacked" by a malicious entity), or because the trusted party's affiliations and allegiances may change with time. For this reason, we wish to consider security against adaptive corruption of the trusted third party (notice, this notion still makes sense even if we only model static corruptions for ordinary parties).

If we are to allow the eventual corruption of a trusted party, we cannot expect protocols that employ this particular trusted party to remain secure for use after the trusted party becomes corrupt. However, we might hope that past protocol executions (which occurred while the trusted party was still honest) remain secure. We say that such protocols are *forward secure* with respect to corruption of the trusted party.

N.B.: *All* of the protocol constructions proposed in this work are forward secure with respect to corruption of any trusted parties involved, except for the static secure protocols of Section 3.2.4. This is holds true even if we assume that the trusted party is not able to erase any state information (*i.e.*, the protocols have forward security with respect to the trusted party in the non-erasure model).

In all situations where a trusted party is required, it is possible to safely distribute the trust among a group of parties – provided only that a majority of the group remains trustworthy at all times (or else the "trusted party" being distributed among the group must be considered corrupt). In particular, it is well known that a UC secure protocol for computing (essentially) any efficient functionality can be obtained in the presence of an honest majority [19]. Distribution of

trust among a group with trustworthy majority merely requires that the group jointly compute the trusted operations using such UC secure protocols. In some particular cases, rather than relying upon this generic approach (which is highly inefficient), we will point out specific efficient methods for distribution of trust among such groups.

2.2 Global Setup Models

While cryptographers commonly strive to minimize the need for parties to place trust in entities that they do not control, it is well known that many important cryptographic tasks fundamentally require parties to place trust in some external *authority*. For example, it is impossible to establish the identity of another party over an unauthenticated network (such as the Internet) without first placing trust in a *certification authority*, or directly communicating at least once with the relevant party via some kind of authenticated channel (which is often impractical). Thus, cryptographic protocols frequently require access to some trusted setup mechanism, such as a certification authority who can provide a Public Key Infrastructure (PKI).

Even though we must occasionally place some trust in an external authority, it is still important to minimize the level of trust required. For instance, trusting an external authority not to lie about my identity to others (and not to lie about the identity of others to me) is not the same as trusting that authority with access to all my private information and private communications.[2] This concern is further highlighted by the fact that trust is often fleeting – a government that is trustworthy today may be overthrown in a coup tomorrow. In light of the preceding discussion, we will strive to limit the powers that must be entrusted to the authority, as well as to provide *forward security* with respect to the corruption of trusted parties, wherever possible.

To further complicate matters, we must also concern ourselves with the financial costs incurred by the trusted authority. If the costs associated with providing trusted services are too high, it can become difficult (or impossible) for parties to find entities they consider trustworthy that

[2]It is important to remember that if a corrupt authority could assume control of my identity and also lie to me about the identity of others, it will be able to gain access to my private communications by performing a "man-in-the-middle" attack. Still, "man-in-the-middle" attacks require pre-meditation and must be perpetrated in an active manner, which implies a high risk that the corrupt behavior of the authority will be noticed and exposed.

can also provide such costly services. The startup and maintenance costs for a modern PKI are already quite high, resulting in a limited availability of PKIs (despite the intense need for such services). Thus, we also strive to minimize the costs associated with implementing the trusted setup mechanism.

2.2.1 Common Reference String (CRS) Model

Perhaps the simplest (and least costly) form of global setup is a Common Reference String (CRS). In this setup model, formally described in Figure 2.1, the trusted authority simply samples a single random value and then publishes it once and for all. When $\bar{\mathcal{G}}_{crs}$ is implemented in practice, the trusted authority never even needs to interact with any parties at all after publishing the reference string – it is precisely this non-interactivity that makes the CRS model so "cheap" and attractive.

Note that this setup does not bind the identities of parties in any way, so clearly it is not useful for authentication purposes. Despite this shortcoming, a series of works inspired by [19, 23, 30] have sought to use the CRS model to construct UC secure protocols (while assuming the presence authenticated and private channels). However – lacking the availability of the GUC "shared functionality" modeling – those works implicitly assumed that the CRS would only be accessible to the adversary, and honest parties running a particular protocol. Notably, these prior works did not allow the environment to have direct access to the CRS, which can lead to a loss of deniability as we will see in Chapter 3.

We also observe that it is often possible to efficiently distribute trust in the CRS model using the "Multi-string Model" approach of [56].

2.2.2 Augmented Common Reference String (ACRS) Model

Unfortunately, as we will see in Chapter 4, the CRS setup model is not strong enough to enable GUC secure realizations of most cryptographic tasks – even if we assume that private and authenticated channels are available. This situation is extremely disappointing, since we seem to be left with no choice but to resort to using a full PKI when designing GUC secure protocols. Since PKIs are extremely costly, we would like to find a "cheaper" setup model that still allows us to

Shared Functionality $\bar{\mathcal{G}}_{crs}$

Parameterized by a distribution $\mathcal{U}$, $\bar{\mathcal{G}}_{crs}$ proceeds as follows, when activated by any party P:

Initialization: If this is the first activation, sample a reference string $\mu \xleftarrow{\$} \mathcal{U}$, and record the value μ.

Return the value μ to the activating party P.

Figure 2.1: The (global) Common Reference String shared functionality. This is a shared functionality in the GUC model, and therefore it differs from the original Common Reference String functionality $\mathcal{F}_{crs}$ discussed in [19, 23, 30], which only provides the reference string to the parties that take part in the actual protocol execution. In particular, the environment does not have direct access to the reference string in that model, whereas our *global* modeling allows the environment to have such access.

design GUC secure protocols. With that goal in mind, we introduce the Augmented Common Reference String (ACRS) model, described formally in Figure 2.2.

Much like the CRS model (and in contrast to the PKI model), the ACRS model does not require honest parties to interact with the trusted authority. Unlike the CRS model, though, the ACRS model requires the trusted authority to offer a (one-time use only) interactive "key retrieval service" for parties who choose to use it. The keys provided by the retrieval service are specific to the party identifier (pid) of the requesting party, but since honest parties are not required to obtain their keys the authority only expects to hand out a small number of keys in practice (even if there are a large number of parties using the setup). In particular, all protocols realized in the ACRS model cannot actually use the key retrieval service, since the model only allows corrupt parties to retrieve their keys. Thus, we are assured that honest parties need never communicate interactively with $\bar{\mathcal{G}}_{acrs}$ in practice.

As we will see in Chapter 4, in our protocols the keys provided by the "key retrieval service" will only useful for attacking one's own security. Somewhat counter-intuitively, it is necessary for corrupt parties to be capable of violating their own security if they wish to conduct attack simulations against our protocols, and therefore corrupt parties will be permitted to retrieve these keys precisely so that they may simulate attacks on a protocol (without actually having to attack the protocol in the real world). This is crucial for providing *deniability* (see Chapter 3), so that honest parties can argue that the protocol interactions they had with corrupt parties

did not actually take place, and were instead simulated by the corrupt parties (acting alone). Notice, since these keys can be used to break one's own security, it is actually important that honest parties not attempt to retrieve their keys – or else they might inadvertently harm their own security by revealing their key to the adversary (for instance, if the environment causes the honest party to run a malicious protocol that leaks secret keys). To summarize, the end result is that honest parties *must not* interact with the trusted authority to retrieve their keys, whereas corrupt parties *must be capable of* retrieving their keys by interacting with the trusted authority.

Shared Functionality $\bar{\mathcal{G}}_{acrs}$

Parameterized by a security parameter λ and a pair of algorithms $(\mathsf{Setup}, \mathsf{Extract})$, $\bar{\mathcal{G}}_{acrs}$ proceeds as follows, when activated by any party P:

Initialization: If no value has been previously recorded, sample random coins $r \xleftarrow{\$} \{0,1\}^{\lambda}$ and compute the CRS as $\mu \leftarrow \mathsf{Setup}(r)$, then record the pair (μ, r).

If P is honest, then return the value μ to P.

Key Retrieval: If P is corrupt, then compute $SK_P \leftarrow \mathsf{Extract}(r, \mu, P)$ and return the pair (μ, SK_P) to P.

Figure 2.2: The Augmented CRS shared functionality.

Remark: As was mentioned previously, it is possible to further reduce the level of trust required in the $\bar{\mathcal{G}}_{acrs}$ setup model by allowing the adversary to adaptively corrupt the trusted authority *after* the completion of all secure protocol sessions. This scenario can be modeled by adding a corruption instruction that causes the $\bar{\mathcal{G}}_{acrs}$ functionality to supply the value r to the adversary. If the security of any protocol sessions that have already completed prior to the corruption of the trusted authority is not compromised, we say that the protocols are *forward secure* with respect to corruptions of $\bar{\mathcal{G}}_{acrs}$. All of our protocols realized using the $\bar{\mathcal{G}}_{acrs}$ model will be forward secure with respect to the corruption of $\bar{\mathcal{G}}_{acrs}$.

Remark: In fact, it is even possible to naturally extend the efficient trust distribution technique of [56] for use in the ACRS model by simply substituting our GUC secure commitment schemes from Chapter 4 for the CRS based UC secure commitments used by [56]. Indeed, such distribution

of trust is even more important for the ACRS model, where the trusted authority must remain available in the long term in order to provide the key retrieval service. Using the approach of [56], the key retrieval service can remain available even after some of the parties in the trusted group have gone offline – provided, of course, that a majority of the still-available members of the group remain trustworthy.

2.2.3 Public Key Infrastructure (PKI) Model

In the Public Key Infrastructure (PKI) model, parties can request to register cryptographic public keys with a trusted "certification authority" that will publish the public key and attest to its validity. For the purposes of the present work, we choose to model a PKI that assumes *Key Registration with Knowledge* of secret keys. That is, parties will provide both their public keys *and* secret keys to the trusted certification authority, who will verify that the secret key corresponds to the registered public key. We formally describe this model using the key registration functionality described in Figure 2.3.

Intuitively, registration with knowledge ensures that parties cannot attempt to register completely malformed public keys or public keys belonging to other parties (since in either case, the registering party will not be able to provide a corresponding secret key). At first, it might seem that requiring parties to provide their secret keys to the certification authority increases the required level of trust. In practice, though, it does not matter much for our intended applications to authentication and commitment schemes. For instance, when using a PKI for authentication purposes, the registration authority can already impersonate any party (even unregistered parties) to any other party, even if no secret keys are provided during the registration process.[3]

[3]Technically, there are still some practical downsides to providing the certification authority with secret keys, even if the PKI is only used for authentication. Knowledge of secret keys allows the authority to impersonate parties in a totally transparent and spontaneous fashion, whereas it may be possible to detect impersonations that require the certification authority to substitute parties' public keys. However, the potential threat this poses is overshadowed by the loss of deniability that inevitably results when choosing not to provide secret keys to the certification authority. A possible compromise was suggested in [6]: have the parties provide only their public keys to the certification authority, and then (preferably while physically isolated from the network) perform a standard zero-knowledge proof of knowledge for the corresponding secret key.

Similarly, in the case of our application to commitment schemes, the certification authority can always perform a simple substitution of public keys in order to break exactly the same security properties of the commitments that are being protected by the secret keys (again, this can be done even if no secret keys are provided during registration).

In addition to requiring key registration with knowledge, we also model the requirement that honest parties protect their secret keys by refusing to provide them as input to all but a pre-specified protocol denoted by Φ. (More generally, we will allow a Φ to include a pre-specified list of protocols.) Indeed, since honest parties can be caused to run arbitrary protocols by the environment, if we don't place restrictions on which protocols are allowed access to the secret key some protocol might publicize the secret keys of honest parties or otherwise "leak" information about them. To formally model the intuitive requirement that secret keys are to be used only with protocol(s) Φ, we parameterize the key registration functionality by a description of the code for Φ and restrict the key registration functionality to provide honest parties with their secret keys only if they are running this code. Otherwise, the key registration functionality refuses to provide honest parties with their own secret keys, thus modeling the prohibition against using the secret keys with other protocols. Corrupt parties, on the other hand, are still allowed to retrieve their secret keys and use them in an arbitrary manner.

Remark: Our $\bar{\mathcal{G}}_{krk}$ functionality is very similar to that of [6]. However, [6] modeled their key registration functionality without benefit of the "shared functionality" paradigm introduced with the GUC model. Therefore, the entire PKI in their model is only visible to a single session of a single protocol – most importantly, the environment was not allowed access to public keys. Such modeling is not realistic, since public keys are long-lived and widely available in practice. By making $\bar{\mathcal{G}}_{krk}$ a shared functionality, we are able to properly model the environment's access to public keys – but we then have to add some extra logic to the functionality to ensure that secret keys are not leaked via abusive protocols. We view this extra logic as an advantage of the GUC modeling – one is now forced to explicitly specify the usage requirements for secret keys directly in the key registration functionality, thus making protocol designers aware of the requirements.

Shared Functionality $\bar{\mathcal{G}}_{krk}^{\Phi}$

Parameterized by a security parameter λ, a protocol (or, more generally, a list of protocols) Φ, and a (deterministic) key generation function Gen, shared functionality $\bar{\mathcal{G}}_{krk}$ proceeds thusly when running with parties $P_1, \ldots, P_n$:

Registration: When receiving a message (`register`) from an honest party P_i that has not previously registered, sample $r \xleftarrow{\$} \{0,1\}^\lambda$ then compute $(PK_i, SK_i) \leftarrow \mathsf{Gen}^\lambda(r)$ and record the tuple (P_i, PK_i, SK_i).

Corrupt Registration: When receiving a message $(\texttt{register}, r)$ from a corrupt party P_i that has not previously registered, compute $(PK_i, SK_i) \leftarrow \mathsf{Gen}^\lambda(r)$ and record the tuple (P_i, PK_i, SK_i).

Public Key Retrieval: When receiving a message $(\texttt{retrieve}, P_i)$ from any party P_j (where $i = j$ is allowed), if there is a previously recorded tuple of the form (P_i, PK_i, SK_i), then return (P_i, PK_i) to P_j. Otherwise return $(P_i, \perp)$ to P_j.

Secret Key Retrieval: When receiving a message $(\texttt{retrievesecret}, P_i)$ from a party P_i that is either *corrupt* or honestly running the protocol code for Φ, if there is a previously recorded tuple of the form (P_i, PK_i, SK_i) then return (P_i, PK_i, SK_i) to P_i. In all other cases, return $(P_i, \perp)$.

Figure 2.3: The Φ-Key Registration with Knowledge shared functionality.

Remark: In the non-erasure model, without loss of generality, we can restrict ourselves to parameterizing with a function Gen that always outputs $SK = r$. Therefore, the adversary will be able to learn the random coins used during generation of the public key PK_i by simply obtaining SK_i from $\bar{\mathcal{G}}_{krk}$ after corrupting the honest party P_i. Furthermore, as in the case of the $\bar{\mathcal{G}}_{acrs}$ above, we can model the scenario where the certification authority is eventually corrupted by the adversary. In this case, even the secret keys of honest parties are revealed to the adversary after $\bar{\mathcal{G}}_{krk}$ has become corrupt. Once again, all our protocols in the PKI model will be forward secure with respect to corruption of the certification authority in the non-erasure model.

Relationship to the "Bulletin Board" and "Bare Public Key" Models. One natural question that arises from our discussion of the PKI setup model is whether or not a weaker PKI modeling might suffice. For example, in the "bulletin board" setup model (also referred to as an "ideal certification authority" in [21]), parties are allowed to generate their own public keys and then post them in a public directory that allows anyone to obtain a particular party's

posted public key in an authenticated manner. The trusted entity responsible for maintaining the bulletin board never attempts to learn any secret keys, nor does it even verify that parties possess their own secret keys. Similarly, one might consider the "bare public key" model introduced by [24], which is identical to the bulletin board model except that *all* parties are required to post their public keys during an initialization phase that occurs before any network protocols are run (a slightly stronger requirement).

Unfortunately, it would seem that any such "weak" PKI model that does not allow corrupt parties to retrieve their own secret keys *cannot* be used to GUC-realize authentication *even against static corruptions* (see Section 3.2.2). Similarly, the impossibility result of Section 4.2 implies that these setup models cannot be used to securely realize bit commitments even if authenticated channels are already available. Nevertheless, we can still make use of a weak PKI setup by combining it with another setup, such as an ACRS. For example, assuming that Identity Based Encryption (IBE) schemes [13] exist, it is possible to obtain all the same feasibility results in the combined bulletin board/ACRS setup model that we achieve in Chapters 3 and 4 when using the (stronger) $\bar{\mathcal{G}}_{krk}$ setup notion. Informally, we can transform our $\bar{\mathcal{G}}_{krk}$-hybrid model protocols to use a combined $\bar{\mathcal{G}}_{acrs}$ and bulletin board hybrid model instead, using the following method:

Let the ACRS setup publish a global reference string containing 1) a reference string σ for use with a standard Non-Interactive Zero-Knowledge proof (NIZK) [47] and 2) the master public key MPK for an IBE scheme [13]. The ACRS setup entity will also provide corrupt parties with IBE decryption keys for their own identities (but it will *not* provide them with any trapdoors for the NIZK reference string).

Now, the original protocol in the $\bar{\mathcal{G}}_{krk}$-hybrid model can be altered to instead use a bulletin board (in combination with the aforementioned ACRS setup) by replacing all of the interactions with $\bar{\mathcal{G}}_{krk}$ as follows:

Registration: Party P_i first generates an ordinary a public key/secret key pair (PK_i, SK_i) along with an IBE encryption (under identity P_i) of the corresponding secret key SK_i, and a NIZK proof that the resulting IBE ciphertext actually contains an encryption of a valid secret key for PK_i. Party P_i then posts PK_i *along with* the IBE ciphertext and the NIZK

proof to the bulletin board. Furthermore, P_i records SK_i for later use.

Public Key Retrieval: Retrieve P_i's posting from the public bulletin board, and check the NIZK proof to make sure the IBE ciphertext contains a valid copy of SK_i corresponding to PK_i. If the NIZK proof verifies correctly, return PK_i as P_i's public key. Otherwise, return $\perp$.

Secret Key Retrieval: If the code of the original protocol is indeed allowed to access the honest party P_i's secret key in the $\bar{\mathcal{G}}^{\Phi}_{krk}$ model, then party P_i running the altered protocol will simply look up its internal record of SK_i and return it. Otherwise, return $\perp$.

Of course, we must also require that all honest parties in the bulletin board model do not use their secret keys *except* with $\bar{\mathcal{G}}^{\Phi}_{krk}$-hybrid protocols that have been altered in this fashion.

Intuitively, posting these IBE ciphertexts (along with the NIZK proof of validity) alongside the public keys assures all other parties that if some party P is corrupt, then P can always retrieve its own secret key by first obtaining the IBE decryption key for its own identity from the ACRS setup entity and then decrypting the IBE ciphertext. In particular, this guarantee enables us to employ the same GUC simulation techniques used by our $\bar{\mathcal{G}}_{krk}$-hybrid model protocols in order to simulate the "altered" protocols. We omit the formal proof of this claim[4] since the details are straightforward (and somewhat tedious).

In practice, however, there is little advantage to using this weakened setup model. Although the trusted entity responsible for maintaining the bulletin board does not learn any secret keys, as was previously mentioned, it can still violate the security of honest parties by generating a fresh public key and (illegitimately) posting it in place of the victim's own public key. However, this attack is more noticeable – for example, the victim might observe that an illegitimate public key has been issued for its identity, giving the victim the opportunity to "cry foul" in a public venue. On the other hand, when using our $\bar{\mathcal{G}}_{krk}$ PKI setup, the certification authority (*i.e.*, the trusted setup entity) might "silently" abuse knowledge of a party's secret key (for example, by impersonating the party) without providing the victim with any hint of the attack. Notice,

[4]Note, we do not claim to have a *general* transformation from protocols using $\bar{\mathcal{G}}_{krk}$ to protocols in the combined $\bar{\mathcal{G}}_{acrs}$ and bulletin board setup model. Rather, our *specific* protocols are easily shown to maintain their security after the above transformation.

however, that the ACRS setup entity also has the capability to silently abuse its knowledge of secret keys – therefore, when combining the bulletin board setup with the ACRS setup, there is still the potential for such "silent" attacks. Perhaps the best that can be said about the weakened model is that one has to place complete trust only in the ACRS setup entity (which is less expensive to maintain, increasing the likelihood that trustworthy parties are willing to do so), as well as placing slightly less trust in a bulletin board setup entity (since it is more likely to be detected if it "cheats").

2.2.4 The Random Oracle (RO) Model

The Random Oracle (RO) model is a very powerful and widely used technique for designing efficient cryptographic protocols (see [11]). Intuitively, in the RO model some particular cryptographic hash functions are assumed to be essentially opaque to analysis, allowing protocol designers to model them as if they were random functions. While it is provably impossible to instantiate any cryptographic hash function satisfying this "random oracle" property (*e.g.*, see [25]), practitioners continue to make use of the RO model since attacks that exploit the gap between well-designed cryptographic hash functions and (purely theoretical) random oracles are generally considered to be unlikely.

Although, as we will see, the RO model does not appear to be useful when we consider the *feasibility* of designing GUC secure protocols for most tasks, it can still be applied to improve the *efficiency* of GUC secure protocols. Since cryptographic protocols that are not highly efficient are unlikely to be used in practice, demonstrating that GUC secure protocols can be made efficient is an important goal of the present work. To that end, when studying the design of efficient GUC secure protocols in Chapter 5, we will demonstrate an application of the RO model to improving the efficiency of GUC secure protocols. A formal description of the RO model setup assumption is given in Figure 2.4.

Remark: Our modeling of random oracles differs from previous works in the UC framework, which modeled the random oracle using a standard *ideal functionality* rather than as a *shared functionality*. The interpretation of the ideal functionality modeling is that the cryptographic hash function used to instantiate the random oracle must be made available *only* to the parties

Shared Functionality $\bar{\mathcal{G}}_{ro}$

Parameterized by a security parameter λ, the shared functionality $\bar{\mathcal{G}}_{ro}$ proceeds as follows when activated by any party P:

Upon receiving a message (P, sid, x) from party P, if there is a record of the form (x, v) then return v to P. If no such record exists, then sample a random value $v \xleftarrow{\$} \{0,1\}^\lambda$, record the pair (x, v), and return v to P.

Figure 2.4: The (global) Random Oracle shared functionality.

participating in the protocol and to the adversary, but *not* to the external environment. This interpretation does not reflect reality – in fact, other parties who are not involved in the same protocol can also make use of the same cryptographic hash function. Once consequence of this unrealistic interpretation is that the UC simulator can seemingly replace the random oracle by intercepting random oracle queries (observing them), and "programming" arbitrary responses to those queries. On the other hand, modeling the random oracle as a shared functionality assumes that *everyone* has access to the cryptographic hash function, which is much more realistic. With this GUC based approach to modeling, the simulator can no longer replace the random oracle (nor even observe the random oracle queries) since the environment itself can interact directly with the random oracle, cutting the simulator out of the picture entirely. This has particular importance for the issue of deniability, which we study in Chapter 3.

2.3 Honest Verifier Zero-Knowledge

Zero-Knowledge (ZK) proof protocols, first introduced in [55], are a fundamental building block used in many cryptographic protocols. In particular, ZK proofs can be used to transform protocols that are only secure against so-called "honest but curious" adversaries (who always obey the protocol) into protocols that are secure even against malicious adversaries (who may attempt to deviate from the protocol).[5] Indeed, we will focus significantly on the matter of designing efficient GUC secure ZK proofs in Chapter 5 due to their widespread utility.

[5]Essentially, the transformation consists of having parties send ZK proofs showing the correctness of their protocol flows, relative to a particular (previously committed, but hidden) protocol input.

In this section, we discuss some specific definitions for more traditional ZK proof protocols with standalone security (rather than GUC security) that are used as building blocks in some of our other constructions (ultimately working towards GUC secure protocols). These definitions are not new, but we have slightly tailored some of the definitional details and presentational issues for our own applications.

2.3.1 Σ-Protocols

The most basic ZK proof protocols that we will consider in this work are commonly known as Σ-protocols (see [68, 35]). Intuitively, Σ-protocols are three move protocols for proving some particular NP-relation in zero-knowledge.

Definition 2.1. Let X, W, and $L \subset X$ be finite sets. We refer to an element $x \in X$ as an "instance", an element $w \in W$ as a "witness", and the set L as a "language". Furthermore, assume there is an efficiently testable binary relation R, such that for all $x \in L$ we have $(x, w) \in R$ for some $w \in W$. (Here we define "efficiently testable" to mean R runs in time polynomial in $|x|$, and therefore we must also have that $|w| \leq \mathsf{poly}(|x|)$.)

Consider a three move protocol run between a prover P, with input $(x, w) \in R$, and a verifier V with input x, executed as follows. P chooses some random coins r_a, computes $a \leftarrow A(x, w, r_a)$, and sends a to V. V then chooses a random string c (the "challenge"), and sends it to P. Finally, P computes and responds with $z \leftarrow Z(x, w, r_a, c)$. The verifier V then computes and outputs the result of $B(x, a, c, z)$, which returns either accept or reject. We require that A, Z, and B be (deterministic) poly-time algorithms, and that $|c| \leq \mathsf{poly}(|x|)$. Such a protocol is called an Σ-*Protocol* for the language L if it satisfies the following properties:

- **Completeness -** *If $(x, w) \in R$ then an interaction between $P(x, w)$ and $V(x)$ – following the procedure outlined above – causes the verifier to output* accept *with overwhelming probability.*

- **Special HVZK (Honest Verifier Zero-Knowledge) -** *There exists a PPT algorithm* ZKSim *(a "Zero-Knowledge simulator"), such that every PPT distinguisher D has at most negligible advantage in the following game:*

1. D computes $(x, w) \in R$ along with a challenge c, and sends (x, w, c) to the challenger.
2. The challenger chooses $b \in \{0, 1\}$ and some coins r at random, and computes the following:

   if $b = 0$: $\quad a \leftarrow A(x, w, r), \qquad z \leftarrow Z(x, w, r, c)$

   if $b = 1$: $\quad (a, z) \leftarrow \mathsf{ZKSim}(x, c, r)$

   The challenger then sends (a, z) to D.
3. D outputs a guess $\hat{b} \in \{0, 1\}$.
4. D's advantage is defined to be $|\Pr[\ b = \hat{b}\ ] - 1/2|$.

Attack Game 1: Special HVZK Attack Game

To simplify analysis, we additionally require that ZKSim *must always output either a "valid" pair* (a, z), i.e., *such that* $B(x, a, c, z)$ *outputs* accept, *or an error condition.*[6]

Intuition: *Essentially, this means that for any efficiently computable challenge c, it is possible to produce a protocol transcript computationally indistinguishable from an actual run with a prover (who knows w), even without knowledge of w. Furthermore, we require that the indistinguishability holds even if the distinguisher knows the witness w. (This additional requirement is slightly stronger than the usual HVZK property.) We chose to define the HVZK security property in the form of an interactive game in order to ensure that the challenge is generated computationally, which facilitates the use of certain number theoretic assumptions (such as Strong RSA) for constructing Σ-protocols.*

- **Special Soundness -** *There exists a PPT algorithm $\mathcal{E}_{\mathrm{rw}}$ (a "rewinding extractor"), such that every PPT adversary D wins the following game with at most negligible probability:*

[6] Note that this requirement does *not* further restrict the class of languages with Augmented Σ-protocols.

1. D computes (x, a, c, z, c', z'), where $c \neq c'$, and sends it to the challenger.
2. The challenger computes $w \leftarrow \mathcal{E}_{\text{rw}}(x, a, c, z, c', z')$.
3. D wins if both $B(x, a, c, z)$ and $B(x, a, c', z')$ return accept, but $(x, w) \notin R$.

Attack Game 2: Special Soundness Attack Game

Augmented Σ-Protocols

Since we are attempting to construct protocols that will remain secure even in the presence of adaptive corruptions, we would naturally like to ensure that Σ-protocol HVZK simulations are indistinguishable from actual "honest" proofs *even after random coins are revealed*. To that end, we will consider an "augmented" definition of Σ-protocols that adds a new *reverse state construction* property. This property will allow us to construct "fake" state information after honest runs of the Σ-protocol (with a prover who knows the witness), in order to make them appear as if they were instead output by the zero-knowledge simulator (which does not require knowledge of the witness). This property is closely related to the "non-erasure" Σ-protocols of [35], which instead satisfy the *state construction property* – that is, it is possible to create random coins making it appear as if a simulated proof (originally computed without the witness) were an honest proof. Both reverse state construction and state construction properties are used to deal with adaptive corruptions, but they are each applicable to different simulation scenarios. In particular, while we will be making use of reverse state construction, none of our simulation techniques happen to require the use of state construction.

We now formally state the properties of Augmented Σ-protocols (in particular, we define the reverse state construction property):

Definition 2.2. An *Augmented Σ-protocol* is a three move protocol which, in addition to satisfying the properties of Σ-protocols given in Definition 2.1, also satisfies the following property:

- **Reverse State Construction -** There exists a PPT algorithm RSC such that every PPT distinguisher D has at most negligible advantage in the following game:

1. D computes $(x, w) \in R$ along with a challenge c, and sends (x, w, c) to the challenger.
2. The challenger samples coins r_a and r_s from a uniformly random distribution, and then computes $(a, z) \leftarrow \mathsf{ZKSim}(x, c, r_s)$, $a' \leftarrow A(x, w, r_a)$, $z' \leftarrow Z(x, w, r_a, c)$, and $r'_s \leftarrow \mathsf{RSC}(x, w, r_a, c)$. Then the challenger chooses a random bit $b \in \{0, 1\}$, and proceeds as follows:

   if $b = 0$: Send (a, z, r_s) to D.

   if $b = 1$: Send (a', z', r'_s) to D.
3. D outputs a guess $\hat{b} \in \{0, 1\}$.
4. D's advantage is defined to be $|\Pr[\ b = \hat{b}\ ] - 1/2|$.

Attack Game 3: Reverse State Construction Attack Game

N.B.: For the case $b = 1$, the values of a' and z' being supplied to D in the game above come from the outputs of A and Z, not ZKSim. Therefore, in order for the values of a' and z' observed by D to remain consistent with r'_s, it must be true with overwhelming probability that $\mathsf{ZKSim}(x, c, \mathsf{RSC}(x, w, r_a, c))$ (the "simulation" output resulting from the coins produced by the Reverse State Construction procedure) yields the same choice of a' and z' as are produced by A and Z when using the random coins r_a. Otherwise, D would notice the inconsistency and could correctly guess that $\hat{b} = 1$ whenever a discrepancy occurs, giving D a non-negligible advantage.

Intuition: This additional property ensures that after a standard run of the Σ-protocol, it is possible for the prover to find some pseudo-random input for the simulator causing it output the same transcript. Furthermore, even a distinguisher who knows the witness cannot determine whether those pseudo-random coins were chosen using this reconstruction procedure or not (*i.e.*, whether they are truly random). It is easy to verify that this property actually implies the above Special HVZK property of Σ-protocols as well. The reasons for this particular additional requirement will become clear in Section 2.4.

2.3.2 Ω-Protocols

The notion of an Ω-protocol was introduced in [50]. While our notion of an Ω-protocol is the same in spirit as that in [50], we also introduce some new properties, and there are a few points where the technical details of our definition differ. We will need these slightly specialized definitions in order to accommodate the specific number theoretic constructions used in our application of Ω to construction efficient GUC secure protocols in Chapter 5.

Intuitively, Ω-protocols are Σ-protocols that have an extra "trapdoor soundness" property. In particular, Ω protocols accept a "reference parameter" as an additional input. A party who possesses a trapdoor for the reference parameter is capable of extracting the witness used by the prover from the transcript of any honest run of the HVZK proof. That is, it is possible to extract the witness from an Ω-protocol without rewinding the prover (when given knowledge of the trapdoor). The notion of trapdoor soundness is closely related to that of *verifiable encryption* [2, 17]. Indeed, all known constructions of Ω-protocols boil down to using a public key for a semantically secure encryption scheme as reference parameter, where the trapdoor is the secret key; the prover encrypts a witness, and then proves that it did so using a Σ-protocol.

Here we will also introduce an additional "system parameter" that is of little conceptual importance, but is merely a technical requirement for our number theoretic instantiations of Ω-protocols. The system parameter is used to determine the sets X, W, and L, over which the Ω protocol may operate. Note that the system parameter should never be "trapdoored".

We now proceed to formally define the properties of Ω protocols.

Definition 2.3. Let *ParamGen* be an efficient probabilistic algorithm that takes as input 1^λ, where λ is a security parameter, and outputs a *system parameter* Λ. The system parameter Λ determines the finite sets X, W, and $L \subset X$, as well as an efficiently testable binary relation R, such that for all $x \in L$ we have $(x, w) \in R$ for some $w \in W$. The sets X and W, and the relation R should be efficiently recognizable (given Λ). There is also an efficient probabilistic algorithm *RefGen* that takes as input a system parameter Λ and outputs a pair (ρ, τ), where ρ is called a *reference parameter*, and τ is called a *trapdoor*. Furthermore, suppose that there are algorithms

A, Z, and B satisfying the same properties as a Σ-protocol (only, here these algorithms are additionally parameterized by the system parameter Λ and the reference parameter ρ).

Such a protocol is called an Ω-*Protocol* for the language L if it satisfies all the security properties of Σ-protocols whenever Λ and ρ are honestly generated and the following additional property is satisfied:

- **Trapdoor Soundness -** There exists a PPT algorithm $\mathcal{E}_{\text{td}}$, called the *trapdoor extractor*, such that every PPT adversary D has at most negligible probability of winning in the following game:

1. The challenger generates a system parameter Λ and a reference parameter/trapdoor pair (ρ, τ), and then sends (Λ, ρ) to D.
2. $\mathcal{A}$ computes $x \in X$ and first flow a, then sends the pair (x, a) to the challenger.
3. The challenger generates a random challenge c, and sends it to D.
4. D generates a response z, and sends this to the challenger.
5. The challenger runs $\mathcal{E}_{\text{td}}$ on input $(\Lambda, \tau, x, a, c, z)$, obtaining a value w.
6. $\mathcal{D}$ wins if (a, c, z) is an accepting conversation for x, but $(x, w) \notin R$ (*i.e.*, w is not a witness for x).

Attack Game 4: Trapdoor Soundness

We also define a slightly weaker variant of Ω-protocols that have only *partial trapdoor soundness*. Intuitively, partial trapdoor soundness means that a party in possession of the trapdoor can only learn some function of the prover's witness from an honest proof transcript (rather than learning the entire witness). In particular, this weaker soundness requirement is useful in our number theoretic construction of an Ω-protocol for proving knowledge of a decommitments. Namely, in our construction, the trapdoor extractor outputs the message contained in the commitment *without* also extracting the random coins used to commit, even though these random coins are a part of the witness.

Definition 2.4. We say that an Ω-protocol is only *partial trapdoor sound* if it satisfies all the properties of Ω-protocols, except that the trapdoor soundness property is replaced with the following weaker requirement:

- **Partial Trapdoor Soundness -** There exists a PPT algorithm $\mathcal{E}_{\text{td}}$, called the *trapdoor extractor*, such that every PPT adversary D has at most negligible probability of winning in the following game:

1. The challenger generates a system parameter Λ and a reference parameter/trapdoor pair (ρ, τ), and sends (Λ, ρ) to D. Note that Λ defines X, L, W, R as above.
2. D computes $x \in X$, along with two accepting conversations (a, c, z) and (a, c', z') for x, where $c \neq c'$, and gives these to the challenger.
3. The challenger then runs $\mathcal{E}_{\text{rw}}$ on input $(\Lambda, \rho, x, a, c, z, c', z')$, obtaining $w \in W$.

   The challenger also runs $\mathcal{E}_{\text{td}}$ on input $(\Lambda, \tau, x, a, c, z)$, obtaining a value v.
4. D wins if w is not a witness for x, or if $v \neq f(w)$.

Attack Game 5: Partial Trapdoor Soundness

Remark: Using a standard rewinding argument ([10]), it is easy to show that partial trapdoor soundness property implies the trapdoor soundness property when f is the identity function, assuming the size of the challenge space is large (*i.e.*, super-polynomial).

Finally, we state an additional useful variant of Ω-protocols that is important in our applications: Ω-protocols with *dense reference parameters*, which we sometimes abbreviate to the shorter name *dense Ω-protocols*. Intuitively, this refers to the class of Ω-protocols with reference parameters that can be chosen via random sampling while remaining indistinguishable from those produced by *RefGen*. As we will see in Chapter 5, this enables us to use a coin-flipping protocol to allow parties to agree upon a reference parameter.

Definition 2.5. We say that a (partial trapdoor sound) Ω-protocol has *dense reference parameters*, or that it is a *dense Ω-protocol*, if it satisfies the following additional properties:

Recall that ρ belongs to $\widehat{\Phi}$, which is determined by Λ. Let Φ be some larger set, also determined by Λ. We call elements of Φ *extended reference parameters*. Further suppose that:

- we have an efficient algorithm that samples the uniform distribution on Φ — this algorithm takes Λ as input;
- we have an efficient algorithm that determines membership in Φ — this algorithm also takes Λ as input;
- we have an efficiently computable binary operation on Φ that makes Φ into an *abelian group*; the inverse operation of the group should also be efficiently computable;
- it is computationally infeasible to distinguish a random element of Φ from a random element of $\widehat{\Phi}$.

The last condition may be stated more precisely as saying that every PPT distinguisher D has at most negligible advantage in the following attack game:

1. The challenger generates a system parameter Λ. This determines sets $\widehat{\Phi}$ and Φ as above.
2. The challenger chooses $b \in \{0,1\}$ at random, and computes an extended reference parameter ρ in one of two ways, depending on b:
   - if $b = 0$, then $\rho \leftarrow \mathit{RefGen}(\Lambda)$;
   - if $b = 1$, then $\rho \xleftarrow{\$} \Phi$.

   The challenger sends ρ to D.
3. D outputs $\hat{b} \in \{0,1\}$.
4. D's advantage is defined to be $|\Pr[b = \hat{b}] - 1/2|$.

Attack Game 6: Dense Reference Parameters

2.3.3 Constructing Augmented Σ-protocols from One Way Functions

Here we briefly describe a generic technique for constructing an Augmented Σ-protocol for any NP-relation. First, we observe that every NP-relation is known to have such an ordinary Σ-protocol if one-way functions exist [48, 52, 12]. Specifically, the protocol of [12] (wherein the prover commits to a permutation of a graph with a Hamiltonian cycle, and is challenged to reveal either a cycle or the permutation) is easily shown to support the requirements of augmented Σ-protocols if we use a commitment scheme that yields pseudo-random commitments. In particular, the reverse state construction property can be achieved by simply producing random coins corresponding to the graph opened by the simulator, along with coins yielding the same values as all the unopened commitments (which were actual commitments edges in the graph, although we can now pretend that they were simply random strings by relying on the pseudo-randomness of the commitments). Note that the existence of the required pseudo-random commitment schemes based on one-way functions is a fairly straightforward consequence of [57] and [70].

2.3.4 Constructing Ω-protocols from Σ-protocols

We now briefly sketch how to efficiently construct an Ω-protocol Π for a relation R, given any efficient Σ-protocol Ψ for relation R. Furthermore, we show how to achieve the dense reference parameter property for the Ω-protocol, assuming the existence of a "dense" semantically secure encryption scheme.

Intuitively, we want to apply a transformation to Ψ that adds both the dense reference parameter and trapdoor extractability properties while carrying over Ψ's existing Σ-protocol properties.

Let the reference parameter for Π be the public key pk for a "dense" semantically secure encryption Enc (where the dense property of the encryption scheme simply satisfies the requirements of the Dense Reference Parameter property of Ω protocols). For instance, standard El-Gamal encryption will suffice for this purpose (under the DDH assumption). Let $\psi = \mathsf{Enc}_{pk}(s, m)$ denote an encryption of message m with random coins s.

Let a, z^c denote the first and last messages (respectively) of the prover in protocol Ψ when operating on input (x, w, r) and with challenge c, where $(x, w) \in R$ and r denotes the random

coins of the prover. The three messages to be sent in protocol Π will be denoted as a', c', z'.

Intuitively, we will use a cut-and-choose technique to provide extractability, and then amplify the soundness by parallel repetition k times. The first message a' of Π is constructed as follows:

1. For $i = 1, \ldots, k$, choose random coins r_i and compute a_i, z_i^0, and z_i^1 using the prover input (x, w, r_i).
2. For $i = 1, \ldots, k$, compute ciphertexts $\psi_i^0 = \mathsf{Enc}_{pk}(s_i^0, z_i^0)$ and $\psi_i^1 = \mathsf{Enc}_{pk}(s_i^1, z_i^1)$.
3. Set $a' := (\psi_1^0, \psi_1^1, \ldots, \psi_k^0, \psi_k^1)$.

The challenge c' sent to the prover in Π is a k-bit string $c' = c'_1 c'_2 \ldots c'_k$. The last message z' of protocol Π is then constructed as follows:

1. For $i = 1, \ldots, k$, set $z'_i := (s_i^{c'_i}, z_i^{c'_i})$.
2. Set $z' := (z'_1, \ldots, z'_k)$.

The verifier's algorithm for Π is simply constructed accordingly, verifying that all the ciphertexts were correctly constructed, and that the corresponding conversations for Ψ are valid.

Theorem 2.6. *Π constructed as above is an Ω-protocol for relation R, provided that Ψ is a Σ-protocol for relation R and Enc is a dense one-time semantically secure public key encryption scheme.*

The proof of this theorem is entirely straightforward, and is left as an exercise.

2.4 Identity-Based Trapdoor Commitments

We now define a tool that is central to our constructions in Chapter 4: Identity-Based Trapdoor Commitments (IBTCs). Our IBTC definition is actually a slight generalization of the identity-based chameleon hash functions first introduced in [3]. Any Identity-Based Chameleon Hash function is sufficient (but not *necessary*) to yield an IBTC.

Definition 2.7 (IBTC)**.** An Identity-Based Trapdoor Commitment scheme IC is given by a 5-tuple of PPT algorithms $IC = (\mathsf{Setup}, \mathsf{Extract}, \mathsf{Com}, \mathsf{ECom}, \mathsf{Eqv})$, with the following basic properties:

- Setup: A deterministic algorithm which takes as input a (uniformly random) "master secret key" $MSK \in \{0,1\}^\lambda$ (where λ is a security parameter), and outputs a public key PK.
- Extract: On input (ID, MSK) outputs a trapdoor SK_{ID} for identity ID.
- Com: A deterministic algorithm which takes as input a tuple of the form (PK, ID, d, m), and outputs a commitment κ for message m under identity ID using random input d. The domain from which d is chosen is denoted by $\mathcal{D}$. As a shorthand, we may write $\mathsf{Com}_{ID}(d, m)$ to denote $\mathsf{Com}(PK, ID, d, m)$.
- ECom: On input (PK, ID, SK_{ID}) outputs a pair (κ, α), to be used with Eqv.
- Eqv: On input $(PK, ID, SK_{ID}, \kappa, \alpha, m)$ produces a value $d \in \mathcal{D}$ such that $\mathsf{Com}_{ID}(d, m) = \kappa$. That is, d makes κ appear to be a commitment to m.

Following conventional terminology, we refer to κ as a "commitment" for identity ID and the pair (d, m) as the corresponding "decommitment". A commitment c for identity ID is said to "open to" a certain message m using the decommitment pair (d, m) if $c = \mathsf{Com}(PK, ID, d, m)$.

Furthermore, Identity-Based Trapdoor Commitments are required to satisfy the following security properties:

- **Binding -** Every PPT adversary $\mathcal{A}$ wins the following game with negligible probability:

1. The challenger computes $MSK \xleftarrow{\$} \{0,1\}^\lambda$, $PK \leftarrow \mathsf{Setup}(MSK)$, and then sends PK to $\mathcal{A}$.
2. $\mathcal{A}$ is allowed to query an oracle for $\mathsf{Extract}(PK, \cdot, MSK)$ (many times).
3. $\mathcal{A}$ outputs (ID, d, m, d', m').
4. $\mathcal{A}$ wins if ID was not submitted to the Extract oracle in Step 2 and $m \neq m'$, but $\mathsf{Com}_{ID}(d, m) = \mathsf{Com}_{ID}(d', m')$.

Attack Game 7: Binding Attack Game (for IBTCs)

- **Equivocability -** Every PPT adversary $\mathcal{A}$ has at most negligible advantage in the following game:

1. The challenger computes $MSK \xleftarrow{\$} \{0,1\}^\lambda$, $PK \leftarrow \mathsf{Setup}(MSK)$, and then sends MSK to $\mathcal{A}$ (unlike the previous game, where only PK was provided to $\mathcal{A}$).
2. $\mathcal{A}$ chooses an identity ID and a message m, and sends (ID, m) to the challenger.
3. The challenger chooses $b \in \{0,1\}$ at random, and computes $d \in \mathcal{D}$ in one of two ways, depending on b:
   - if $b = 0$, then $d \xleftarrow{\$} \mathcal{D}$;
   - if $b = 1$, then $SK_{ID} \leftarrow \mathsf{Extract}(PK, ID, MSK)$, $(\kappa, \alpha) \leftarrow \mathsf{ECom}(PK, ID, SK_{ID})$, $d \leftarrow \mathsf{Eqv}(PK, ID, SK_{ID}, \kappa, \alpha, m)$.

   The challenger then sends d to $\mathcal{A}$.
4. $\mathcal{A}$ outputs a "guess" $\hat{b} \in \{0,1\}$.
5. $\mathcal{A}$'s advantage is defined to be $|\Pr[\ \hat{b} = b\ ] - 1/2|$.

Attack Game 8: Equivocability Attack Game (for IBTCs)

Remark: Throughout this work, our efforts are directed towards obtaining security in the presence of adaptive corruptions. Therefore, we only consider IBTCs that utilize the random input to Com as the decommitment, rather than a more general notion that allows for Com to generate a separate decommitment value. Similarly, we require all the random coins input to Setup to come from MSK in order to enable forward security (*i.e.*, to tolerate adaptive corruption of the trusted setup party).

Intuition: Roughly speaking, the security properties of IBTCs consist of two natural computational security properties known as "binding" and "equivocability". The binding property states that any adversary, given only knowledge of PK and the secret keys of some users *other* than ID, cannot produce a commitment for ID with two different openings. The equivocability property guarantees that the output of Eqv is indistinguishable from a uniformly random choice of d, even

to an adversary who knows MSK (this is required for forward security reasons). In other words, Eqv allows one to open a special "equivocable commitment" κ produced by ECom in such a way that κ is indistinguishable from a normal commitment to m generated via $\mathsf{Com}(PK, ID, d, m)$ (rather than ECom), even though m had not yet been specified when κ was actually produced using ECom.

2.4.1 Constructing Identity-Based Trapdoor Commitments

Our approach is based on the technique from [46] for constructing commitment schemes from Σ-protocols, only we will be making use of Augmented Σ-protocols in order to obtain security against adaptive corruptions. In particular, we will make use of signature schemes that have Augmented Σ-protocols for proving knowledge of a signature. The following theorem summarizes the results of our IBTC construction technique:

Theorem 2.8. *There exists an efficient construction of an IBTC from any signature scheme with an "augmented Σ-protocol". In particular, such signature schemes exist if one-way functions exist, and can be efficiently constructed under the Strong RSA assumption.*

The proof follows immediately from the details of the construction, which we now outline:

Let $\Upsilon = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ denote any signature scheme that is existentially unforgeable against chosen message attack (*i.e.*, it is UF-CMA secure). Define a relation $R^{\Upsilon}(x, w)$ such that $(x, w) \in R^{\Upsilon}$ if either 1) $x = \langle VK, m \rangle$ and $w = \sigma$ such that $\mathsf{Ver}(VK, m, \sigma) = \mathsf{accept}$; or 2) $w = r_g$ such that $(VK, SK) = \mathsf{Gen}(r_g, 1^{\lambda})$ and SK is a valid signing key corresponding to VK (that is, w contains all the random coins used by the Gen algorithm when producing VK and SK). We say that Υ has an Augmented Σ-protocol if there is an Augmented Σ-protocol for the relation R^{Υ}.

Remark: The second case of R^{Υ}, which allows r_g to be used as a witness, is necessary to ensure that the properties of the Augmented Σ-protocol for the signature scheme are preserved even with respect to adversaries who eventually learn r_g. We ultimately rely on this property for "forward security" purposes, in case the trusted setup party who generates VK is later corrupted. Of course, since $\mathsf{Gen}(r_g, 1^{\lambda})$ yields the signing key SK, if the witness is r_g we can use it to compute

a signature witness σ satisfying the form of the first case. Therefore, if there is an Augmented Σ-protocol for signatures, we can easily transform it into an Augmented Σ-protocol for R^{Υ} by simply having provers with a witness r_g compute the appropriate signature witness, and then run the original Augmented Σ-protocol using the signature witness (of course, a prover who starts with a signature witness simply runs the original protocol). This transformation does not at all impact the efficiency of the original protocol, and it is easily verified that all properties of Augmented Σ-protocols hold for the transformed protocol with respect to the relation R^{Υ}. Therefore, we observe that it is sufficient to have an Augmented Σ-protocol for the signature scheme alone, despite our use of the slightly more involved relation R^{Υ}.

Given such a signature scheme, we can construct an Identity-Based Trapdoor Commitment scheme as follows:

- $\mathsf{Setup}(1^\lambda)$**:** Compute $(VK, SK) \leftarrow \mathsf{Gen}(r_g; 1^\lambda)$, where r_g denotes the random coins used by the Gen algorithm. Return the pair $(PK = VK, MSK = r_g)$.
- $\mathsf{Extract}(PK, ID, MSK)$**:** Run $\mathsf{Gen}(MSK, 1^\lambda)$ to obtain SK. Compute $\sigma_{ID} = \mathsf{Sig}(SK, ID)$. Return $SK_{ID} = \sigma_{ID}$.
- $\mathsf{Com}(PK, ID, d, m)$**:** Compute $(a, z) \leftarrow \mathsf{ZKSim}(x = \langle PK, ID\rangle, c = m, r_s = d)$, where ZKSim is the Σ-protocol simulator for R^{Υ}. Return the commitment $\kappa = a$.
- $\mathsf{ECom}(PK, ID, SK_{ID})$**:** Sample random coins r_a from a uniform distribution and then compute $a = A(x = \langle PK, ID\rangle, w = SK_{ID}, r_a)$. Return the pair $(\kappa = a, \alpha = r_a)$.
- $\mathsf{Eqv}(PK, ID, SK_{ID}, \kappa, \alpha, m)$**:** Set $r_s = \mathsf{RSC}(x = \langle PK, ID\rangle, w = SK_{ID}, r_a = \alpha, c = m)$. Return $d = r_s$.

Intuition: The message m is used as the "challenge" flow on which to run the Zero-Knowledge simulator for the Σ-protocol, producing the prover's first flow a and a response z. Only the first value, a, is used to form the commitment κ. The decommitment d consists of exactly the random coins r_s used to conduct the simulation. The prover's response flow z is not used at all. This is a slight departure from the technique of [46], which instead uses z as the decommitment

and requires the recipient to verify that $B(a, c, z)$ outputs accept. In this work, we allow the eventual (adaptive) corruption of the committer, in which case all random coins used in the HVZK simulation must be revealed. Since security must hold against an adversary who can eventually learn the random coins r_s (used by the HVZK simulation algorithm ZKSim to produce the pair (a, z) for the Σ-protocol), we can simplify our protocols by allowing the random coins r_s to serve directly as the decommitment. Nevertheless, when considering specific instantiations, in practice it may be more efficient to send z as the decommitment (using the approach of [46]) rather than r_s (even though r_s must still be provided to the adversary when the committer is corrupted).

Equivocation of a commitment is achieved by using knowledge of the witness w (*i.e.*, the equivocation trapdoor SK_{ID}) to "honestly" conduct the Σ-protocol proof (instead of simulating) so that the prover can later respond to any "challenge" (thus opening the commitment to any message), even though the first flow was already sent as the commitment. Once again, we avoid the need to send the response flow z by sending the random coins r_s as the decommitment instead, and, therefore, we make use of the Reverse State Construction algorithm to produce the decommitment.

We are now ready to prove Theorem 2.8.

Proof of Theorem 2.8. We will prove that the previously described implementation is a secure IBTC. It is easy to verify the correctness of the implementation, so we will proceed directly to prove the security properties:

- **Binding.** Suppose there is an adversary $\mathcal{A}$ breaking the binding property of the commitment scheme. We describe a reduction that forges signatures for Υ (in the UF-CMA attack game) by acting as the challenger for $\mathcal{A}$ in the security game for the binding property of the IBTC as follows:

  1. To forge a signature in a UF-CMA attack game with verification key VK, the challenger sets $PK = VK$ and sends PK to $\mathcal{A}$.
  2. The challenger then responds to $\mathcal{A}$'s queries to the $\mathsf{Extract}(PK, \cdot, MSK)$ oracle by

simply forwarding them to the signature oracle in the UF-CMA game, and returning the responses. (It is easy to verify that this yields the correct output.)

3. Wait until $\mathcal{A}$ outputs a tuple (ID, d, m, d', m').

4. If $\mathcal{A}$ wins the binding attack game, then ID was never queried to the oracle, and $\kappa = \mathsf{Com}_{ID}(d, m) = \mathsf{Com}_{ID}(d', m')$. In this case, we may compute pairs of the form $(\kappa, z) \leftarrow \mathsf{ZKSim}(\langle PK, ID\rangle, m, d)$ and $(\kappa, z') \leftarrow \mathsf{ZKSim}(\langle PK, ID\rangle, m', d')$. That is, we now have two accepting transcripts for the Augmented Σ-protocol which begin with the same first flow, enabling us to use the rewinding extractor (from the soundness property) to obtain: $w \leftarrow \mathsf{Ext}(\langle PK, ID\rangle, \kappa, m, z, m', z')$. With overwhelming probability, w is a valid witness and therefore either contains a signature σ_{ID} such that $\mathsf{Ver}(PK, ID, \sigma_{ID})$ accepts, or the random coins needed to recover the signing key itself (in which case it is easy to produce a forgery). Since ID was never queried to the oracle in the binding attack game, the reduction never queried it to the signing oracle in the UF-CMA game, and thus the reduction may output σ_{ID} as a new forgery.

That is, whenever $\mathcal{A}$ succeeds in breaking the binding property of our IBTC construction, the reduction outputs a forgery with overwhelming probability. Clearly then, if the signature scheme is unforgeable, the binding property must hold.

- **Equivocability.** Equivocability of commitments follows immediately from the Reverse State Construction property of Augmented Σ-protocols. Clearly, any adversary capable of distinguishing the output of Eqv from random coins (when given access to MSK), can also be used to distinguish the output of RSC (when using a witness containing r_g). That is, set $x = \langle PK, ID\rangle$ and $w = r_g$ in the Reverse State Construction game, and feed the resulting challenge to the adversary who breaks the equivocability of the commitment scheme, then output the same guess as the equivocability adversary. It is easy to verify that the resulting advantage in the Reverse State Construction game is the same as that of the adversary attacking the equivocability property of the commitment scheme.

□

Signature Schemes with Augmented Σ-Protocols

To complete the construction of IBTCs, it remains to construct a signature scheme with an augmented Σ-protocol for the knowledge of the signature. Recall that, by the result of Section 2.3.3, there exists an Augmented Σ-protocol for any NP-relation.

Since signature schemes can also be constructed from one-way functions, we immediately obtain a generic construction of IBTCs based on one-way functions. We also notice that we can have more efficient constructions if we use certain signature schemes based on specific number-theoretic assumptions. For example, we can use any "signature scheme with efficient protocols", such as the one of [16] based on strong RSA, or the Waters' signature scheme which we employ in Chapter 5.

2.5 Specialized Encryption Schemes

We make use of several specialized encryption schemes in our constructions, which we formally outline here.

2.5.1 Dual Receiver Encryption (DRE)

Conceptually, a *Dual Receiver Encryption* (DRE) scheme is similar to the notion of plaintext-aware encryption (via key registration) proposed in [58]. Intuitively, DRE allows anyone to encrypt a message to two parties, with a single ciphertext. Crucially, we are guaranteed that attempts to decrypt a ciphertext by either of the two recipients, will produce the *same result.* (Of course, each recipient should somehow know who the other intended recipient is, or else this notion is not very meaningful.)

Definition 2.9. We say that the 3-tuple of PPT algorithms, $(\mathsf{Gen}, \mathsf{DREnc}, \mathsf{DRDec})$ is a *Dual Receiver Encryption* (DRE) scheme if it satisfies the following properties:

Correctness: For any two values $r_0, r_1 \in \{0,1\}^\lambda$ and any message M, we require for $i = 0, 1$ that:

$$\mathsf{DRDec}(PK_i, SK_i, PK_{1-i}, \mathsf{DREnc}(PK_0, PK_1, M)) = M$$

where $(PK_i, SK_i) \leftarrow \mathsf{Gen}(r_i)$.

Soundness: Any PPT adversary $\mathcal{A}$ has at most negligible probability of winning the following game.

1. The challenger chooses $r_0 \in \{0,1\}^\lambda$ uniformly at random, generates a public key pair $(PK_0, SK_0) \leftarrow \mathsf{Gen}(r_0)$, and sends PK_0 to $\mathcal{A}$.
2. $\mathcal{A}$ outputs $r_1 \in \{0,1\}^\lambda$, and a ciphertext ψ.
3. $\mathcal{A}$ wins if r_1 generates a public key $(PK_1, SK_1) \leftarrow \mathsf{Gen}(r_1)$ with respect to which ψ does not decrypt consistently, *i.e.*, $\mathsf{DRDec}(PK_0, SK_0, PK_1, \psi) \neq \mathsf{DRDec}(PK_1, SK_1, PK_0, \psi)$.

CCA2 Security: Any PPT adversary $\mathcal{A}$ has at most negligible advantage in the following game:

1. The challenger chooses $r_0, r_1 \in \{0,1\}^\lambda$ uniformly at random, then generates public key pairs $(PK_i, SK_i) \leftarrow \mathsf{Gen}(r_i)$ for $i = 0, 1$, and sends (PK_0, PK_1) to $\mathcal{A}$.
2. The challenger answers arbitrary decryption queries from $\mathcal{A}$, *i.e.*, it provides $\mathcal{A}$ with access to an oracle for $\mathsf{DRDec}(PK_0, SK_0, PK_1, \cdot)$.
3. $\mathcal{A}$ chooses two messages M_0 and M_1, and sends (M_0, M_1) to the challenger.
4. The challenger chooses a bit $b \in \{0,1\}$ uniformly at random, computes the ciphertext $\psi \leftarrow \mathsf{DREnc}(PK_0, PK_1, M_b)$, and sends ψ to $\mathcal{A}$.
5. The challenger answers more decryption queries from $\mathcal{A}$, except for queries on ψ.
6. $\mathcal{A}$ outputs a guess bit $\hat{b}$.
7. $\mathcal{A}$'s advantage is defined to be $|\Pr[\hat{b} = b] - 1/2|$.

Furthermore, we restrict Gen to be a deterministic algorithm (so that all of its random coins are taken from the input r).

An example of such a DRE scheme is to use any CCA2 secure encryption scheme to encrypt the message twice, once under each public key, and then provide two Non-Interactive Zero-Knowledge (NIZK) proofs (of the kind used in [80]) that both ciphertexts contain equivalent plaintexts – one proof using a reference string taken from the first public key, and one proof

using a reference string taken from the second public key. This is similar to the approach used in [58], and the proof of security is analogous. More formally:

Theorem 2.10. *The scheme* $(\mathsf{Gen}, \mathsf{DREnc}, \mathsf{DRDec})$ *described below is a secure DRE scheme, when parameterized by any public key encryption scheme that is CCA2 secure (*i.e.*, secure against adaptive chosen ciphertext attacks), and any Non-Interactive Zero-Knowledge (NIZK) proof system.*

Gen: Given random coins as input, Gen uses some of them to generate a public key/secret key pair (pk, sk) for the CCA2 public key encryption scheme, and the rest to produce a reference string σ for the NIZK proof system. Gen outputs $(PK = \langle pk, \sigma \rangle, SK = sk)$.

DREnc: On input (PK_0, PK_1, M), DREnc first parses $PK_i = \langle pk_i, \sigma_i \rangle$, and then computes ciphertext c_i as the encryption of M under public key pk_i (for $i = 0, 1$). Next, DREnc computes a NIZK proof π_i under reference string σ_i, proving that c_0 and c_1 both valid ciphertexts containing encryptions of the same value (again, for $i = 0, 1$). Finally, DREnc outputs the tuple $\psi = (c_0, \pi_0, c_1, \pi_1)$.

DRDec: On input (PK_0, SK_0, PK_1, ψ), DRDec first parses the public keys PK_i as (pk_i, σ_i) (for $i = 0, 1$), and the ciphertext ψ as (c_0, π_0, c_1, π_1). Next, DRDec verifies that both π_0 and π_1 are valid NIZK proofs with respect to σ_0 and σ_1 (respectively) that c_0 and c_1 are valid ciphertexts containing an encryption of the *same* value under public keys pk_0 and pk_1 (respectively). If the verification fails, DRDec returns $\perp$. Otherwise, DRDec returns the decryption of c_0 using secret key SK_0.

Proof. The correctness property of the scheme is easily verified (following from completeness of the NIZK proof system, and correctness of the underlying CCA2 secure public key encryption scheme). To prove soundness, by way of contradiction, we simply observe that any adversary who defeats the soundness property of the DRE scheme can easily be used to violate the soundness property of the NIZK proof system used with respect to PK_0 (*i.e.*, the public key which is not under control of the adversary). Proving the CCA2 security of the DRE scheme is slightly more involved, and we proceed by means of two intertwined reductions, as follows. Suppose, by way

of contradiction, there is an adversary $\mathcal{A}$ that succeeds in the CCA2 Security attack game with advantage ε, where ε is non-negligible.

First, let us modify the original attack game by having the challenger replace the honestly generated values of σ_0 and σ_1 used in PK_0 and PK_1 with "trapdoored" (simulatable) reference strings, and then subsequently letting it simulate the two NIZK proofs (π_0 and π_1) rather than computing them honestly. By the zero-knowledge property of the NIZK proof system, the adversary's advantage in the new game, ε', must also be non-negligible.

Next, we consider an alternative version of this game, which we call the "Mixed Ciphertext Game". In this game, rather than computing challenge ciphertext $\psi \leftarrow \mathsf{DREnc}(PK_0, PK_1, M_b)$ honestly, the challenger instead computes c_0 as the encryption of M_b under pk_0, and c_1 as the encryption of M_{1-b} under pk_1 (while "faking" both proofs π_0 and π_1 with the NIZK simulator, as per the previously modified attack game). In all other respects, the game proceeds normally. Let δ denote the probability that $\mathcal{A}$ correctly guesses b (which corresponds to the message encrypted by c_0). (Note that this *success probability* is not be confused with $\mathcal{A}$'s "advantage" at this game.)

To analyze the adversary's success probability δ in the Mixed Ciphertext Game, we will consider a reduction to the security of the underlying public key encryption scheme. Given a public key pk, the reduction first chooses PK_0 normally (along with its secret key), but sets $PK_1 = pk$ (for which it does not know the secret key). Later, when $\mathcal{A}$ chooses messages (M_0, M_1), the reduction forwards these to its own challenger, receiving a ciphertext c in response. The reduction then computes ψ as in the Mixed Ciphertext Game, except that instead of choosing c_1 to be the encryption of M_{1-b}, it simply sets $c_1 = c$. Finally, when $\mathcal{A}$ outputs a guess bit $\hat{b}$, the reduction outputs the same guess bit.

Clearly, whenever c is an encryption of M_b, the resulting game corresponds to our modified version of the original CCA2 security attack game (since both c_0 and c_1 will encrypt the value M_b). Therefore, the reduction's distinguishing success probability in this event will be $\frac{1}{2} + \varepsilon'$ (since $\mathcal{A}$ guesses correctly with this probability, in the original game). On the other hand, if c is an encryption of M_{b-1}, then the reduction's success probability is δ, since the game played by the reduction corresponds exactly to the Mixed Ciphertext Game. Thus, the overall success probability of this reduction is $\frac{1}{2}(\frac{1}{2} + \varepsilon' + \delta)$. If the underlying public key encryption scheme is CCA2 secure, this implies that $\delta \approx \frac{1}{2} - \varepsilon'$ (that is, $\delta = \frac{1}{2} - \varepsilon'$ up to negligible factors).

Finally, we arrive at a contradiction by giving a reduction attacking the CCA2 security of the underlying public key encryption scheme. Given a public key pk, our reduction first chooses a random bit $\beta \in \{0,1\}$, then generates PK_1 normally (along with its secret key), but sets $PK_0 = pk$ (for which it does not know the secret key). During the game, when the reduction must answer decryption queries from $\mathcal{A}$, it uses the secret key for PK_1 rather than PK_0 in order to decrypt (this change will not matter as long as the NIZK proof system remains sound, and we will neglect the negligible probability of failure in the present calculations for the sake of clarity). When $\mathcal{A}$ chooses messages (M_0, M_1), the reduction forwards these to its own challenger, receiving a ciphertext c in response. The reduction then computes ψ by setting $c_0 = c$ and c_1 to be an encryption of M_β. Finally, when the reduction obtains a guess bit $\hat{b}$ from $\mathcal{A}$, it checks whether $\hat{b} = \beta$. If so, the reduction outputs $\hat{b}$. Otherwise, the reduction outputs a random bit.

To analyze the success probability of $\mathcal{A}$ in our reduction, we observe that one of two possible conditions occurs: either 1) the challenge ciphertext c corresponds to a matching encryption of M_β, or 2) it corresponds to a mismatched encryption of $M_{1-\beta}$. In case 1, the game plays out identically to the original CCA2 attack game (modified to use simulated proofs). Therefore, the overall success probability in case 1 is $\frac{1}{2} + \varepsilon' + \frac{1}{2}(\frac{1}{2} - \varepsilon')$ (accounting for the random guess that is made whenever $\hat{b}$ does not match β). In case 2, the game plays out identically to the Mixed Ciphertext Game described above. Thus, the overall success probability in case 2 is $\frac{1}{2}\delta$ (since the reduction will guess randomly whenever $\mathcal{A}$ "correctly" deduces $\hat{b}$ relative to $c_0 = c$, but will always output an incorrect guess whenever $\mathcal{A}$ guesses $\hat{b}$ inconsistently with c_0). Given that case 1 and case 2 are equiprobable, we find the overall success probability of the reduction to be $\frac{3}{8} + \frac{\varepsilon' + \delta}{2}$. If the underlying public key encryption scheme is CCA2 secure, this implies that $\delta \approx \frac{1}{4} - \varepsilon'$, contradicting our earlier result that $\delta \approx \frac{1}{2} - \varepsilon'$. □

2.5.2 Non-Committing Encryption (NCE)

We will also be making use of 2-round non-committing encryption (NCE) scheme, of the kind first proposed by [22]. Intuitively, an NCE scheme is a semantically secure encryption scheme with the extra property that it allows a simulator to produce a public key/ciphertext pair which it can later "open" to an arbitrary message. That is, the simulator can produce a secret key

such that decrypting the ciphertext with that secret key yields the message. Furthermore, the simulator can even show random coins making it appear that the secret key and public key were honestly generated, and that the message was legitimately encrypted to produce the ciphertext.

To simplify our notation, we make use of non-committing encryption for λ-bit messages, although it is typically defined in the literature for 1-bit messages. (In particular, we wish to encrypt messages proportionately sized to the security parameter, which we also denote by λ.) Informally, we refer to an NCE scheme as a 5-tuple of algorithms $(\mathsf{NCGen}, \mathsf{NCEnc}, \mathsf{NCDec}, \mathsf{NCSim}, \mathsf{NCEqv})$ where NCEnc and NCDec satisfy the usual notion of semantic security for encryption, and (1) NCSim outputs a public key pk, a ciphertext γ, and some auxiliary information α, (2) The public keys and ciphertexts output by NCSim are indistinguishable from public keys output by NCGen and ciphertexts resulting from encryption under such keys, and (3) Given the output of NCSim and any message M, NCEqv can produce a secret sk and computationally random coins r^*, r_γ such that $\mathsf{NCGen}(r^*) = (pk, sk)$, $\mathsf{NCDec}(sk, \gamma) = M$, and $\mathsf{NCEnc}(r_\gamma; pk, M) = \gamma$. That is, NCE schemes can operate just like normal encryption schemes, but it is also possible to generate indistinguishable "rigged" public keys and ciphertexts such that the ciphertexts can be made to appear as if they contain any desired plaintext (with some appropriate choice of secret key). Details regarding the construction of such NCE schemes can also be found in [22, 34, 65].

At a high level, one can construct a NCE scheme from any public key (1-bit) encryption scheme with the following special properties: 1) there is an "oblivious key generation" algorithm for sampling fresh public keys without learning their corresponding secret keys (and yielding a distribution on the public key space that is indistinguishable from that of the ordinary key generation process), 2) given a fixed public key, there is an "oblivious sampling" algorithm for producing a valid ciphertext encrypting a random bit *without* learning the corresponding plaintext bit (and yielding a distribution on the ciphertext space that is indistinguishable from that of the ordinary encryption process). For the purposes of adaptive security (in the non-erasure model), we also require that appropriate random coins can be produced for the oblivious key generation and sampling algorithms, *a posteriori*, so as to make "normally" produced public keys and ciphertexts appear as if they were produced by the oblivious procedures. To construct a NCE scheme for 1-bit messages from such an "obliviously samplable" public key encryption scheme, the recipient will essentially choose λ pairs of public keys $(pk_{i,0}, pk_{i,1})$ (for $i = 1, \ldots, \lambda$),

where λ is a security parameter. Instead generating all the public keys in the usual manner, the recipient randomly chooses a bit β, and generates only the public keys of the form $pk_{i,\beta}$ in the normal fashion. The remaining public keys of the form $pk_{i,1-\beta}$ are all "obliviously sampled" using the algorithm implied by property 2 above. (This is the NCGen procedure.) To encrypt a bit m (*i.e.*, to implement NCEnc), the sender will generate one ciphertext pair corresponding to each public key pair. For each public key pair, the sender randomly selects one of the two public keys with which to encrypt the bit m, while the other public key is used to "obliviously sample" a random ciphertext. The resulting 2λ ciphertexts are then sent to the recipient. On average, 3/4 of the ciphertexts sent in this fashion will encode the correct bit m. In order to decrypt (*i.e.*, to implement NCDec), the recipient opens the λ ciphertexts encrypted under the "normal" public keys (for which it knows the corresponding secret key), and computes the bit occurring in the majority of such plaintexts.

Intuitively, the simulation procedure (NCSim) for the scheme above can be implemented by first generating all public keys using the normal procedure (rather than generating half of them obliviously). Ciphertexts can then be produced (using normal encryption) so that a majority of bits encrypted under one half of the key pairs are 0, and the majority in the other half are 1. To perform the equivocation (NCEqv), simply select the appropriate half of the public keys (with respect to which a majority of the ciphertexts encode the desired message bit m) to be the "normal" ones, and then "pretend" that the other public keys were obliviously generated. Similarly, an appropriately chosen subset of the undesirable ciphertexts can be made to look as if they had been obliviously sampled. Finally, we note that this entire scheme can be repeated λ times, in parallel, in order to encrypt λ-bit messages (*i.e.*, encrypting one λ-bit message will require $2\lambda^2$ ciphertexts).

2.5.3 Dense PRC Secure Encryption

The commitment protocol of [30] (which uses a common reference string) requires the use of a CCA secure public key encryption scheme with Pseudo-Random Ciphertexts (PRC). The CCA security requirement arises due to the fixed nature of the global public encryption key, which is reused during every run of the protocol. Our protocol, on the other hand, will not make use of

any fixed public keys for encryption. Instead, we will be using a coin-flipping protocol to choose a fresh public key for each run of our commitment protocol (this is discussed in more detail below). One advantage of doing this is that we will require only semantically secure encryption with PRC, since each key is used only once. However, there is also a disadvantage: the public key is chosen as the result of a fair coin-flipping protocol (which will be uniformly random in some finite group), so the encryption scheme must have "dense" public keys. Informally, this means we require public keys to be computationally indistinguishable from random. Fortunately, schemes satisfying these requirements exist under widely used computational assumptions, such as the DDH assumption (in fact, standard El-Gamal encryption suffices). The formal definition of security for Dense PRC encryption provided below has the flavor of "left-or-right" security, which is most convenient for the proof of security of our protocol.

Definition 2.11 (Dense PRC Encryption)**.** A public key encryption scheme $\text{PRC} = (G, E, D)$, satisfying the standard correctness properties for public key encryption, is said to be *Dense PRC* secure if it satisfies the following additional constraints:

1. Public keys produced by G lie within some abelian group Φ, with efficiently computable inverse and group operations.

2. Membership in Φ is efficiently testable.

3. The uniform distribution on Φ is efficiently samplable.

4. The distribution of public keys produced by G is computationally indistinguishable from the uniform distribution on Φ.

5. For all PPT adversaries $\mathcal{A}$:

$$\Pr[A^{LR(\cdot,0)}(1^\lambda) = 1] - \Pr[A^{LR(\cdot,1)}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

   where the answer to a query of the oracle $LR(m, b)$ is computed by obtaining $(K, K^{-1}) \leftarrow G(1^\lambda)$ and returning the pair $\langle K, E_K(m)\rangle$ if $b = 0$, or returning the uniformly random pair $\langle U, R\rangle$ if $b = 1$ (where U is chosen from a group Φ, and R from $\{0,1\}^{|E_U(m)|}$). We need not allow $\mathcal{A}$ to query the oracle more than once, but this restriction is unimportant as long as fresh keys are computed on each query to the oracle.

Intuition: The final condition in the definition captures the main security property of Dense PRC: an honestly generated public key (output by G) and an honestly generated ciphertext (output by E) are indistinguishable from a randomly chosen public key and a random bit string of appropriate length. Note that it must be possible to encrypt messages to random public keys, and that the result ciphertexts must still appear to be random (since otherwise we would have a way to distinguish random public keys from honestly generated ones). This is important for our purposes, since our protocols require parties to encrypt messages under a randomly generated public key. The remaining properties of Dense PRC schemes are merely technical requirements that enable two parties to jointly select a random public key via a standard "coin-flipping protocol".

3 • Deniability and Deniable Authentication

3.1 Deniability and Full Simulatability

Historically, there have been many different definitions of deniability for cryptographic protocols (*e.g.*, [44, 42, 73, 37, 39]). In fact, to this day, the issue of defining what we mean by the term "deniability" is almost entirely unsettled in the cryptographic community. Partly, this is because deniability seems to mean different things in different contexts. However, the definitional problem runs much deeper than this, since the are rarely any consistent definitions even for well defined and well studied cryptographic tasks such as authentication.

At an intuitive level, a cryptographic protocol is said to be "deniable" if it is possible for the protocol participants to deny their participation by arguing that any "evidence" of said participation (as might be obtained by other, potentially corrupt protocol participants and observers) could have been fabricated without their involvement.[1] While it may sound natural, this intuition is not generally of much help in proposing a formal definition of deniability, since it is open to many different interpretations. For instance, how plausible must the claim of the putative protocol participant be (namely, the claim that he or she was "framed" by fabricated evidence)? Most everyone would probably agree that it would not be believable if the participant's argument

[1] There are other applications of the word "deniability" that we do not consider here, since they carry a completely different intuitive meaning. For instance, it is sometimes said that a voting protocol is "deniable" if the participants are inherently unable to prove how they have voted to others – even if they intentionally deviate from the voting protocol (this requirement is often referred to as the "secret ballot principle"). Such notions are more appropriately captured by the term "incoercibility", rather than "deniability", since they offer security to parties who act dishonestly (ostensibly, under coercive influence). Throughout this work, we consider all parties who intentionally deviate from protocol specifications to be corrupt, and therefore we do not attempt to guarantee their security in any sense. Indeed, it is generally impractical or impossible to provide such security to parties who deviate arbitrarily from the protocol, since they might simply choose to "hand over" control of their identity (and all their secrets) to an adversarial entity. However, for certain specialized circumstances (such as with voting systems), it makes sense to consider security for *coerced* parties who may deviate from the honest protocol. Extension of our notion of deniable realizations to this setting is left to future work.

claims that that the "whole world" conspired against him or her. At the other extreme, the claim of denial might merely state that a *single*, unnamed individual was responsible for fabricating evidence of a party's participation. Thus, if there exists even one individual who might plausibly have the means and motive to fabricate such evidence, the claim would certainly be believable (and if we believe no such individual exists, then no claim of denial can possibly be believed anyway). So, ideally, we should strive to obtain the latter form of deniability. Yet, this too is not clearly defined – after all, what constitutes plausible "means" to fabricate evidence? If the practical price of fabricating evidence were very high, then we might not find the claim of denial very plausible even though anyone could have fabricated the evidence (after paying the aforementioned high price). Indeed, any serious discussion of *plausible* deniability is inherently subjective, since plausibility is a subjective notion.

Our goal in the present discussion is therefore not to define deniability once and for all, but rather to outline a general approach to understanding and achieving a natural notion of deniability for well-defined cryptographic tasks. To maintain a focus on objective goals, our notion of deniability will be quantifiably "plausible" *in a computational sense*. Ultimately, the degree of plausibility achieved by our definitions and constructions *outside of the computational sense* will be left to the subjective view of the individual. Therefore, the notion of deniability we seek is that of *full simulatability* – any evidence of protocol interaction with a particular party can be simulated in a *computationally efficient* manner without the involvement of that party.

Of course, we must define our notion of deniability with respect to the task at hand – for instance, if one is running a contract signing protocol that is designed to provide the other party with a "digital signature" of some statement, then one should not later expect to deny having signed the statement since, by definition, digital signatures preclude the possibility of such a denial. Thus, if we are to define a general notion of deniability for cryptographic tasks, then our goal should be for protocols to provide no evidence of interaction that is *not inherently revealed* by the task itself. For example, in the case of the contract signing protocol, there should be no evidence that the contract was signed via the protocol interaction – it should be possible for the signer to plausibly claim that this particular signature was simply communicated directly (perhaps even to a third party) and that no sophisticated contract signing protocol ever took place.

We are now in a position to state a very elegant and general definition of deniability for cryptographic tasks. Recalling the basic notion of the trusted party paradigm (as outlined in the Introduction), we intuitively expect realized protocols to guarantee the same security as the ideal functionalities they realize. In particular, the adversary should learn nothing more from attacking the protocol than could be learned from attacking its corresponding ideal functionality. Protocols realized with such a guarantee are inherently deniable, since a protocol participant can plausibly claim that any information sent during the protocol session could have been obtained by an adversary (given only knowledge of the same information revealed by the ideal functionality[2] for the task under consideration) via an attack simulation conducted entirely without the putative participant's actual participation.

Of course, it goes without saying that realized protocols only guarantee the same "security" as the ideal functionality does when viewed in the particular context of a *security model*. To preserve our notion of deniability (and security in general), it is crucial that the security model be made as close to reality as possible. Trivially, if the capabilities of a real world adversary are not adequately captured by the model, attacks against the "security" (and deniability) of the protocol might be possible in practice even though the protocol satisfies the (unrealistically weak) security requirements of the model. Surprisingly, it is also possible that a security model featuring an adversary with *greater capabilities* than the real world adversary will lead to a problem with deniability (even though the security model simply appears to be "too strong"). Unlike most natural security properties that we might consider, the plausibility of a denial depends upon the ability of an adversary to accomplish a particular "attack" – namely, the adversary must plausibly be capable of *simulating* (fabricating) evidence of protocol interactions. If the adversary of the security model is more powerful than the real world adversary, it is entirely possible that the only reason the adversary learns nothing useful by attacking "secure" protocols is that the adversary is already powerful enough to independently generate similar information. If the real world

[2] Again, if the output of the ideal functionality "incriminates" a party by revealing some of his secrets, the resulting protocol does not meet a very intuitive interpretation of the word "deniable" (as was the case in the contract signing example above). Still, the protocol itself may be said to be "as deniable" as the functionality it realizes, which is the most one can ask for. If this is not sufficiently "deniable" in practice, then the only recourse is to change the functionality itself, since that is where the problem lies.

adversary is not as powerful, then perhaps it can still gain something useful by attacking the protocol after all, since it cannot exercise the same natural powers as the adversary of the security model. Indeed, we will see a concrete example of this phenomenon in Chapter 4.

To summarize the lessons of the preceding discussion, we *will not* attempt to formally define the meaning of "deniability" itself, since it is highly subjective by nature. Rather, for any particular cryptographic task $\mathcal{F}$, we will say that:

> *A protocol π is a* deniable realization *of $\mathcal{F}$ if it securely realizes the task $\mathcal{F}$ in a* realistic *security model, in such a way that any attack on π can be simulated by attacking $\mathcal{F}$ using roughly equivalent resources.*

In other words, by our definition, any "lack of deniability" that is evident in π (but not in $\mathcal{F}$) can be directly attributed to an inaccuracy in the security model, or an excessively expensive simulation procedure. Therefore, we view the loss of deniability merely as a first sign that the security model or the "proof of security" (in the form of an attack simulation) is flawed, since deniability is automatically assured by a realistic security model and a reasonable security proof. To reiterate, *any* inaccuracy in the security model can present a problem for deniability (*e.g.*, even when the adversary in the model is unrealistically *more* powerful than any real life adversary).

With this definition in mind, we will study the application of the GUC security model to the problem of *deniable authentication* (in this chapter) and to the construction of deniable realizations for general cryptographic tasks (in Chapter 4). As we will see, our particular deniable realizations of cryptographic tasks will guarantee that even "on line" (interactive) deniability is preserved, since our simulator can very practically be run in real time. Indeed, as long as an honest party P never deviates from the protocol, it is not possible for other (even corrupt) protocol participants to conclusively demonstrate P's participation in our "fully simulatable" protocol sessions to a third party, *even while the protocol is ongoing*! This simulation guarantee will hold even in the presence of arbitrary concurrent protocol executions in the network, and even when the honest party participates in those concurrent protocols. Such a strong notion of deniability is essentially unheard of in the literature.[3]

[3]The model of [59] contains a similar notion, but they do not allow adaptive corruptions (which are very important in practice), and their constructions require the use of special hardware "signature cards" that are tamper-resistant.

3.2 Deniable Authentication

Following our definition of deniability from the previous section, we approach the problem of designing deniable authentication protocols by simply defining the task of authentication. In the UC framework, the task of authentication is commonly modeled by the ideal functionality $\mathcal{F}_{auth}$, shown in Figure 3.1. The $\mathcal{F}_{auth}$ functionality very cleanly captures the task of authenticated message transmission (over public channels). Similarly, the simpler task of *identification* can be modeled via the functionality $\mathcal{F}_{id}$, shown in Figure 3.2. Finally, we will also give the definition of a task that is closely related to authentication, but has seemingly stronger requirements: symmetric key exchange. The ideal functionality for symmetric key exchange, $\mathcal{F}_{ke}$, is shown in Figure 3.3. Unlike $\mathcal{F}_{auth}$, which reveals the message to the adversary, $\mathcal{F}_{ke}$ must not merely transmit a key in secret, but it must *randomly sample* the key as well. This would appear to make $\mathcal{F}_{ke}$ a much more complex functionality, but in practice, we will show that is closely related to $\mathcal{F}_{auth}$ (see Section 3.2.4 below).

Functionality $\mathcal{F}_{auth}$

1. Upon receiving an input $(\mathtt{send}, \mathsf{sid}, m)$ from party S, do: If $\mathsf{sid} = (S, R, \mathsf{sid}')$ for some R, then output $(\mathtt{sent}, \mathsf{sid}, m)$ to the adversary, and, after a delay, provide the same output to R and halt. Otherwise, ignore the input. (The adversary controls the channel delay.)
2. Upon receiving $(\mathtt{corruptsend}, \mathsf{sid}, m')$ from the adversary, if S is corrupt and $(\mathtt{sent}, \mathsf{sid}, m)$ was not yet delivered to R, then output $(\mathtt{sent}, \mathsf{sid}, m')$ to R and halt.

Figure 3.1: The message authentication functionality (see [20]).

Functionality $\mathcal{F}_{id}$

1. Upon receiving an input $(\mathtt{identify}, \mathsf{sid})$ from party S, do: If $\mathsf{sid} = (S, R, \mathsf{sid}')$ for some R, then output $(\mathtt{identified}, \mathsf{sid})$ to the adversary, and, after a delay, provide the same output to R and halt. Otherwise, ignore the input. (The adversary controls the channel delay.)

Figure 3.2: A simple party identification functionality.

Functionality $\mathcal{F}_{ke}$

$\mathcal{F}_{ke}$, running with parties S and R and adversary $\mathcal{S}$ proceeds as follows:

1. Upon receiving message of the form $(\texttt{keyexchange}, \mathsf{sid}, S, R)$ from party S, if S is not yet "active", mark S "active", and a send public delayed output $(\texttt{keyexchange}, \mathsf{sid}, S, R)$ to R. (Otherwise, ignore the message).
2. Upon receiving a message of the form $(\texttt{keyexchange}, \mathsf{sid}, S, R)$ from party R, if R is not yet "active", mark R as "active" and send a public delayed output $(\texttt{active}, \mathsf{sid}, S, R)$ to S. (Otherwise, ignore the message.)
3. Upon receiving a message of the form $(\texttt{setkey}, \mathsf{sid}, S, R, k')$ from $\mathcal{S}$, if R is corrupt and S is "active", then output $(\texttt{setkey}, \mathsf{sid}, S, R, k')$ to S and R, and halt. If R is "active" but not corrupt, then sample a fresh key $k \xleftarrow{\$} \{0,1\}^\lambda$ and send the message $(\texttt{setkey}, \mathsf{sid}, S, R, k)$ to R. Furthermore, if S is "active", then send the delayed message $(\texttt{setkey}, \mathsf{sid}, S, R, k)$ to S as well. In all cases, this completes the protocol, and the functionality halts.

Figure 3.3: An ideal functionality for symmetric key exchange.

Remark: It is important to note that the ideal functionality $\mathcal{F}_{auth}$ captures *precisely* the definition of "authenticated channels" that is used throughout the literature. Authenticated channels are widely assumed to do nothing more nor less than $\mathcal{F}_{auth}$ accomplishes. Notably, most works in the area of secure multi-party computation (and the UC framework in particular) assume the existence of such authenticated channels (with the notable exception of the extensive literature regarding key exchange protocols). Therefore, there is a great need for secure realizations of the $\mathcal{F}_{auth}$ functionality.

Given that we want to realize these functionalities in a deniable way, we will focus our efforts on using the most accurate possible modeling to capture the natural security requirements of protocols running on the Internet. In particular, it is well known that it is impossible to realize either of these functionalities (barring any physical realization via a secure communications line), unless parties already possess some secret information which they can use to differentiate themselves from other parties. Therefore, in practice, parties who wish to perform authentication nearly always make use of a PKI (as described in Section 2.2.3). The GUC framework is uniquely suited to accurately model global setups such as PKI (which cannot be directly modeled in the UC framework), and therefore we will focus our attention on security models in the GUC framework.

Furthermore, we will assume that both the authenticating party S and the intended recipient R have public keys. This is in contrast to many of the deniable authentication models previously found in the literature [44, 45, 43, 63, 73, 37], in which the recipient is not assumed to have a public key. In our setting, such modeling would not be acceptable, since the $\mathcal{F}_{auth}$ functionality does not permit S to "accidentally" authenticate the message to anyone other than the intended recipient R – therefore, in our setting R must have a means of establishing identity as well (such as a public key). Furthermore, none of those works achieve our notion of "on-line deniability", where simulation must be done in a straight-line fashion (so it can be performed in real-time) and yet the protocol remains concurrently composable with other arbitrary protocols. While security with concurrent self-composition was an explicit goal in several of the prior works [44, 45, 63, 37], to the best of our knowledge all prior constructions achieving concurrent security use timing assumptions [44] which, though somewhat reasonable, seem preferable to avoid.[4] More importantly, the deniable authentication protocols we study benefit from the strong guarantees provided by the GUC security modeling, and therefore remain secure even when executed concurrently with other *arbitrary* protocols, something not addressed in previous works.

We first demonstrate that many previous works on deniable authentication suffer from practical security flaws that can allow an attacker to violate the deniability of the protocol. Next, we will give a result showing that some security flaws are *inherent* in protocols for deniable authentication that rely on a PKI, since realizing the (deniable) authentication functionality in a realistic model is *impossible* if we require security against adaptive corruptions (or even forward security, without erasures). Finally, we will examine our best options for realizing authenticated channels in light of this negative result, ultimately achieving a very practical compromise that allows us to salvage the basic model of (deniable) authenticated channels that is used implicitly throughout the literature on multi-party computation (which we also rely upon for our protocols in Chapters 4 and 5).

[4] It is not entirely clear whether plugging a generic concurrent ZK proof [79] into any existing deniable authentication protocol would yield a concurrently-secure protocol. In any case, concurrent ZK proofs require super-logarithmic round complexity [27] and do not result in efficient protocols.

3.2.1 Attacks on Deniability

The basic notion of deniable authentication was originally introduced by Dwork et al. in [44], using a model where only the sender (who is being authenticated) has a public key. While [44] focuses on achieving deniable authentication even for concurrent protocol executions (via the use of timing assumptions), the requirement for authenticating to a *specific* party was not considered. Essentially, by assuming that only the sender has a public key, the model of [44] makes it impossible for the sender to conclusively determine which party it is authenticating the message to. Of course, in reality, the particular party the sender is authenticating to can matter a great deal. For instance, if Bob intends to send a politically subversive message to fellow political activist Alice, there would likely be undesirable consequences for Bob if he accidentally authenticates the message to a government official instead. Perhaps more importantly, the standard notion of authenticated channels inherently assumes that the message is being authenticated to a specific recipient, and is therefore incompatible with this "single public key" deniable authentication. This means that the notion of deniable authentication proposed in [44], as well as any subsequent works in the "single public key" setting (*e.g.*, [45, 43, 71, 63, 73, 37]), are insufficient to achieve some authentication requirements that are very important in practice.

In fact, in any model where *only* the sender possesses a secret (for authentication purposes), the following simple attack on deniability will succeed *even* if the messages for the authentication protocol are correctly routed to the intended recipient without interference. Suppose that S intends to authenticate the message M to a some party R, who is attempting to prove to judge J that S is the true sender of M. Rather than directly running the authentication protocol in the usual fashion, R will ask the judge J to compute the protocol messages on behalf of R (which is certainly possible, since R is not assumed to have any secrets used by the authentication protocol). R then simply forwards all messages from J to S, and from S to J. If the protocol completes successfully, then S will really be authenticating the message M directly to J (instead of R). Therefore, in such models, S cannot have deniability with respect to an interactive judge J, irrespective of the underlying authentication protocol (provided, of course, that the authentication itself is sound).

Of course, one might be tempted to think that a practical form of deniability can still be

obtained – namely, protocols might still be deniable with respect to an "offline" judge, who is only presented with the "evidence" *a posteriori*. However, when considering protocols running in a more complex environment (such as the Internet), it is nearly inevitable that some "online" activity occurring concurrently with the protocol can be used by a corrupt party to assist in the production of "evidence" – even if the judge itself is "offline" (non-interactive). To provide a more concrete example, imagine there is a time-stamping bulletin board available somewhere on the Internet. This bulletin board allows parties to publicly post information, and the post is automatically time-stamped and assigned a unique identification number (many such services are widely available today). A corrupt recipient can post the protocol messages it receives directly to the bulletin board, and then formulate its protocol responses based on the identification numbers assigned to the resulting posts. This attack automatically defeats the deniability of any protocol with a simulation procedure that requires rewinding (such as the protocols of [44, 63, 37]), since the (external) bulletin board cannot be rewound. Furthermore, the bulletin board even provides a complete "paper trail" for the authentication that is being publicly attested to (albeit unwittingly) by the party who maintains the bulletin board. This paper trail would allow any judge to subsequently verify the message authentication, assuming only that the bulletin board was honest. This holds true for any authentication protocol that does not require the recipient to have a secret key and uses a rewinding simulator.

Lest one suspect that this kind of naturally occurring online attack is specific to protocols that have rewinding simulators, we refer the reader to [45], which features deniable authentication protocols in the "trusted center" model (where there is a trusted third party with a public key, in addition to the sender). In a remark on the limitations of the deniability provided by their protocols, the authors describe how a corrupt recipient can gather evidence against the sender by interleaving messages with authentication sessions involving other recipients. This demonstrates that other Internet users running the same protocol can also provide a practical source of "online" deniability attacks. The lesson to be learned is that it is unrealistic to consider deniability only against an "offline" judge for Internet protocols, since there are many (publicly available) "online" activities on the Internet which can be easily used to help incriminate parties.

Having demonstrated several serious issues with the deniability of protocols in the "single public key" model, we now proceed to survey some works that allow both the sender and recipient

to have public keys. A prototypical technique underlying many such works was introduced by Jakobsson et al. in [60], in the form of designated verifier proofs (and the related notion of designated verifier signatures). Intuitively, the approach taken by [60] is to prove statements of the form "*either* x is true, or I know the secret key of the verifier". Clearly, the verifier will be incapable of convincing anyone else that x is true after seeing such a proof. Similarly, in the context of deniable authentication, the sender will provide a proof that essentially says "*either* I am authenticating the message, *or* I know the recipient's secret key". Unfortunately, this type of statement contains a subtle flaw that compromises the deniability of the authentication protocol: the proof clearly implicates *either* the sender *or* the recipient in the communication. It is not possible for both parties to jointly deny the protocol, since the statement proved essentially says that *either* the message was really authenticated by the sender, *or* the proof was really produced by the receiver. In the case of a non-interactive authentication protocol using this approach (*e.g.*, using a designated verifier signature), the aforementioned flaw trivially enables even an "offline" judge to exclude the possibility that the authentication was completely fabricated by a third party – clearly, at least one of the two (allegedly) communicating parties was involved. In case the authentication is done via an interactive proof protocol instead, the attack we previously described using a time-stamping bulletin board can be used to achieve the same effect.

On a related note, we observe that [60] is not the only technique to suffer from the problem of *jointly* incriminating at least one of the parties involved. For example, various works on "deniable" ring authentication [71, 82, 83] suffer from a similar weakness. In particular, even when all parties are honest, transcripts of deniable ring authentications in the setting of the works by Susilo et al. [82, 83] reveal evidence that at least one of the members of the ring participated *or* that the intended recipient participated. Since we are considering a setting where there is only a single sender who wishes to authenticate, the "ring" can include at most the sender and the recipient – therefore, even honest transcripts will incriminate at least one of the two. (In the setting of [71], the receiver is not required to have a secret key, and therefore the attacks suggested in our earlier discussion of such "single public key" models will apply.)

We will proceed with our survey by considering some of the more recent attempts at deniable authentication (again, in settings where the receiver is also allowed to have a public key). We begin with the various protocols proposed for use in deniable key exchange by Di Raimondo et al.

in [38] (which were offered as replacements for the seriously flawed deniable authentication protocol presented in [14]). Following an approach similar to that of our construction in Section 3.3, the constructions of [38] are actually "deniable key exchange" protocols, which can be used to authenticate messages via the additional application of a standard Message Authentication Code (MAC). The three key exchange protocols suggested for use in deniable applications by [38] are known as SIGMA, SKEME, and HMQV. (The deniability properties of SIGMA and SKEME are explored more formally in [39], using models with considerably weaker security requirements than the one presented in this work.) We now consider attacks on the deniability of each of these protocols in turn.

The SIGMA key exchange protocol is signature based (indeed, the name derives from "SIGn and MAc"). Essentially, both parties must send ordinary digital signatures of some (ostensibly random) values during the SIGMA authentication protocol. Since the protocol transmits digital signatures from both the sender and the recipient in the clear, both the sender and the recipient inherently provide some direct evidence that messages were being sent. This evidence immediately renders the SIGMA approach incompatible with the standard notion of authenticated channels (which does not provide any evidence that any messages are being sent at all), yet this fact alone might not be considered a very significant problem – after all, given that one has a public key, we naturally expect it to be used. However, Di Raimondo et al. already pointed out in [38], it is also possible for the recipient to encode arbitrary statements into one of the "random" group elements that will be signed by the sender. Thus, it is easy to obtain convincing evidence that the sender was attempting to communicate with a particular recipient – in fact, one might reasonably take issue with the claim that SIGMA is at all deniable. The authors of [38] still advocate the SIGMA protocol though, due to its "perfect forward secrecy" property (*i.e.*, the exchanged key can be made forward secure in the erasure model). Of course, SIGMA does not provide "forward deniability" (since the protocol is not deniable even when no compromise has occurred), which is of far greater concern for our purposes.

The second suggested protocol, SKEME, provides a noticeable improvement in security over SIGMA (although SKEME does not provide the same "perfect forward secrecy" guarantee as SIGMA, we are unconcerned with that particular security notion in our setting anyway, since we do not use the exchanged key for encryption). Unfortunately, SKEME still falls far short of

the deniability guarantees we would like. In a nutshell, parties running the SKEME protocol perform a standard Diffie-Hellman key exchange, as well as sending each other random values encrypted under their respective public keys. The final shared key is then derived from the resulting Diffie-Hellman key, as well as both of the random values. Mutual authentication of the SKEME key exchange is assured by virtue of each party's ability to decrypt the random value sent by the other party. If the sender uses the exchanged key to authenticate a message, that authentication is ultimately being provided through the ability of the sender to decrypt a ciphertext. Thus, a judge may gather evidence against the sender by collaborating with a recipient (who is acting as an informant), and supplying this recipient with a ciphertext of the judge's own choosing. If the recipient sends the judge's ciphertext as part of the SKEME key exchange, and then provides the judge with all the details of the SKEME key exchange and the subsequent message authentication, the judge will be able to verify the authentication (which could not have been fabricated by the recipient alone, since the recipient did not know the random value contained in the judge's ciphertext). Notice, the judge does *not* need the recipient's secret key in order to perform the verification (a single decryption of the ciphertext from the sender suffices, since the SKEME key is just a function of the two encrypted values and the Diffie-Hellman key exchange). Still, attacking the deniability of SKEME in this fashion is more challenging than the aforementioned simplistic and direct attacks on the deniability of SIGMA.

The third suggested protocol, HMQV, satisfies the strongest form of deniability among the protocols suggested by [38]. In fact – unlike SIGMA and SKEME – the basic HMQV protocol appears to satisfy even the GUC security requirements for (deniable) key exchange, albeit only for static corruptions. (We offer an even more efficient protocol with the same security guarantees for the static corruption model in Section 3.2.3.) However, the HMQV protocol still does not offer forward security (namely, it fails to provide forward deniability). At a high level, the key exchanged in the HMQV protocol is derived from a combination of an interactive Diffie-Hellman key exchange (using freshly generated public keys) with a non-interactive Diffie-Hellman key exchange (using the fixed long-term public keys of the communicating parties). Since the protocol messages of HMQV are essentially just *unauthenticated* Diffie-Hellman keys, it is possible for an "informant" to pretend to be the recipient while exchanging a key with the sender, allowing the informant to gather useful evidence. For instance, consider an adversary (informant) who cuts

the wire to the intended (honest) recipient and then performs the standard HMQV key exchange protocol with the sender. By providing the judge with the Diffie-Hellman secret it used in the HMQV protocol, as well the subsequent authenticated message from the sender, the adversary can incriminate the sender to any judge who obtains the honest recipient's secret key (*e.g.*, if the judge gets a warrant to seize the recipient's keys, these can also be used to incriminate the sender). That is, although the adversary itself cannot verify the sender's authentication (since it does not have the recipient's secret key, which is needed to complete the non-interactive Diffie-Hellman exchange), the adversary can still provide the information information necessary for the judge to verify the sender's authentication (after the judge obtains the recipient's secret key, of course). Therefore, the protocol is not forward secure for the sender with respect to the eventual "corruption" (or even a passive compromise) of the recipient.[5] This attack can be dramatically strengthened (removing the need for a third-party adversary/informant altogether) if we disallow erasures – in this case, even the past transcripts of ordinary protocol executions between honest parties will suffice to incriminate the sender to any judge who can subpoena the "internal state" data from the honest recipient's machine (*i.e.*, this data would include the temporary secret used by the recipient for the interactive Diffie-Hellman exchange).

Perhaps most importantly, we note that all three protocols (SIGMA, SKEME, and HMQV) are vulnerable to attacks by a fully interactive "online" judge. Rather than belaboring the point by exploring such attacks in further detail, here we will simply quote Di Raimondo et al. in [38], who already noticed this flaw: "... a proof of communication [with party B] can be provided from [a party] A to [a judge] J if A and J actively collaborate at the time the communication is happening, in particular during the run of the key exchange protocol." Whereas the authors of [38] suggest that these attacks are of minimal significance in practice, we feel that this is not the case. As we have already argued above with the example of the time-stamping bulletin board, it is often possible to leverage resources that are present in the external network to achieve the same effect as an "online" interactive judge. Therefore, the Internet itself makes this kind of attack

[5] Although this is somewhat counter-intuitive, we *must* believe that the recipient remains honest until after the attack is been completed, since the recipient would have been able to "simulate" the sender by colluding with the adversary. This makes sense in real life, since a judge might very reasonably be convinced that the recipient was not colluding with an informant (prior to the seizure of the recipient's data).

against deniability quite practical indeed. (Of course, we stress again that the deniability of the HMQV protocol only suffers from these attacks when considering the requirements of forward security,[6] whereas the other protocols are not even secure in the static corruption model when an online judge is available.)

Quite recently, deniable authentication protocols based on random oracles and the (non-standard) Knowledge of Exponent Assumption (KEA) have been proposed (*e.g.*, [73, 86, 61]). In general, such protocols are inherently undeniable[7] in the real world, since they are analyzed under the assumption that only the communicating parties can issue random oracle queries or compute exponentiations (whereas, in reality, these tasks may actually be performed by an external third party). For this reason, even the so-called uncontrollable Random Oracle (uRO) model (also referred to as the non-programmable random oracle model), suggested by Pass in [73] for the purposes of achieving deniability, does *not* suffice to achieve deniability in practice. For example, the protocol simulation techniques used by [73, 61] in the uRO model require the simulator to "extract" (passively observe) all the random oracle queries that are issued as part of the simulated protocol session. However, this sort of "extraction" is certainly not possible in the real world, where an external party (such as the judge, or a trusted friend) might be computing some protocol messages on behalf of the party who is attempting to gather evidence against the sender. As a concrete example, consider the uRO-Auth protocol of [61], where the sender has a public key with a trapdoor permutation T_s. In short, in the uRO-Auth protocol, the recipient R computes $H(r, R, S, m)$ (where H is the random oracle, and r is a nonce) and sends it along with $T_s(r)$ to the sender S. The sender checks the consistency of the protocol flow from R, and then authenticates by responding with $H(r, S, R, m)$ (notice the reversal of the sender and recipient identities). Simulation of this protocol is done by "extracting" the value of r from the random oracle input, following the intuition that a party who computes $H(r, R, S, m)$ can surely

[6] A trivial modification of the HMQV protocol can essentially be used to achieve our definition in Section 3.3, but only if we allow erasures. Indeed, the requirement for erasures seems to be HMQV's only significant shortcoming when compared to our protocol (which can be viewed as a feasibility result for the non-erasure model). Given that HMQV is far more efficient than our construction, currently, HMQV may be the best choice for use in practical systems.

[7] To the best of the authors' knowledge, the protocol of Section 5.5 is the first exception to this rule. Indeed, even this exceptional case is made possible due to the use of the random oracle for efficiency purposes only, rather than fundamentally relying upon the random oracle model for security purposes.

compute $H(r, S, R, m)$. However, this intuition is misleading. If the judge chooses r and provides $H(r, R, S, m)$ along with $T_s(r)$ to the recipient R, the judge will be convinced that S participated if R can produce the value $H(r, S, R, m)$. After all, since the random oracle query was issued by the judge, R has no way of extracting the value of r needed to compute $H(r, S, R, m)$ (but, of course, S can learn it by inverting the trapdoor permutation). The protocol of [73] can be attacked in exactly the same fashion, and the protocol of [86] (which depends upon the KEA assumption) also falls prey to an analogous attack (where the judge computes the exponentiation so that the exponent cannot be extracted by the simulator).

Before concluding our survey (which is not exhaustive), we briefly consider some related works that feature secure models similar to our own. Independently, Hofheinz et al. [59] also proposed a notion of deniability based on the UC framework that is closely related to our GUC notion. However, the model of [59] already assumes the presence of authenticated channels (without describing any appropriate method of realizing them over unauthenticated links), and therefore they do not consider the problem of deniable authentication. Additionally, the deniable protocols they construct require all parties to have so-called "signature cards" (a device with tamper-proof hardware containing a signing key). This is a very strong/expensive setup assumption, and even with this setup their protocols are only secure against static corruptions. Furthermore, since the notion of deniability in [59] depends upon the ability of parties to access their own signature cards, a judge can obtain incriminating evidence against Alice by first confiscating Bob's signature card and then asking Bob to run the protocol with Alice. The related setup model of [64], which assumes the availability of tamper-proof "secure token" hardware, suffers from a similar problem with deniability (and the specific protocols proposed in that work, which were never specifically claimed to be deniable, are vulnerable to other attacks on deniability as well).

Finally, we note that the modeling of deniable authentication in [61] is also based on an approach derived from UC security. However, the random oracle in the first protocol of [61] (discussed above) is unrealistically modeled as a "local" ideal functionality (*i.e.*, the environment is not allowed to issue oracle queries directly), whereas we model random oracles using "global" shared functionalities (see Section 2.2.4). The second protocol presented in [61], which does not use the random oracle model, is only secure with respect to static corruptions (and is significantly more complicated than our proposed solution for the static corruption scenario).

In summary, we have demonstrated explicit attacks against the forward security of all previously proposed deniable authentication protocols (as well as giving many attacks that work even in the static corruption scenario). These attacks "slip through the cracks" in the security provided by previous deniable authentication models, due to unrealistic modeling. Indeed, it seems very likely that many more practical attacks of a similar nature to the ones described herein exist. Of course, this leaves us with the question of how to design more secure deniable authentication protocols, since previous solutions all seem to suffer from practical flaws. Suitably armed with knowledge of these practical attacks on the deniability of authentication protocols, and our more realistic security framework for modeling the requirements of deniability, we are prepared to investigate potential approaches to the design of more secure solutions in depth.

3.2.2 Impossibility of Deniable Authentication in the PKI Model

Given that so many previous authentication protocols seem vulnerable to practical attacks on deniability (despite being designed specifically to resist such attacks), one begins to wonder why this problem is so difficult. We now prove a startling (and seemingly quite harsh) impossibility result: adaptively secure deniable authentication and key exchange (in fact, even identification) is impossible in the PKI model, even if parties prove the knowledge of their secret keys, even in erasures are allowed, even if each secret key is used only once, and even if no man-in-the-middle attacks are allowed. In the non-erasure model, we can also rule out even deniable protocols which are only forward-secure.

Before stating the formal impossibility result and giving its proof, we first comment on why the impossibility result is not as obvious as it might otherwise seem to be. At first, it appears that since the behavior of the sender S can be simulated in a straight-line manner *without* its secret key SK_S, there is an attacker who can impersonate the sender to the recipient R (by running the simulator). Of course, one of the reasons why this does not have to work is that R might use its own secret key SK_R for verification. In particular, a simulated transcript might look different than the real one to R, since R can employ knowledge of SK_R to distinguish the transcript (whereas the adversary does not have this ability). One "fix" to this problem is to (adaptively) corrupt R and then check the simulated transcript from R's viewpoint. Unfortunately, if R is

corrupted too early (say, at the very beginning), it could be the case that knowledge of R's secret key is subsequently employed by the simulator in order to simulate the proper transcript (without talking to S or obtaining SK_S). Notice that such a simulation does not contradict soundness, since, in the real world, R would know that he is not simulating the conversation with S. On the other hand, if R is corrupted too late (say, at the very end), the initial flows from the "then-honest" party R were also chosen by the simulator, so there is no guarantee that they correspond to the behavior of a real R interacting with the sender's simulator.

Theorem 3.1. *There does not exist an adaptively-secure protocol* Π *for realizing the deniable identification functionality* $\mathcal{F}_{id}$ *in the* $\bar{\mathcal{G}}^{\Pi}_{krk}$*-hybrid model. Moreover, the impossibility holds even under the following additional assumptions/constraints:*

- *Secure data erasures are allowed.*
- *Each honest party* P *will not use its secret key* SK_P *for more than one identification session (either as a sender or as a recipient).*
- *The attacker* $\mathcal{A}$ *will either try to impersonate the sender to an honest recipient exactly once, or will try to impersonate the recipient to an honest sender exactly once. In particular,* $\mathcal{A}$ *will not mount the man-in-the-middle attack against two honest parties.*

Since identification $\mathcal{F}_{id}$ is trivial to implement from either $\mathcal{F}_{auth}$ (by having S authenticate any fixed message) or $\mathcal{F}_{ke}$ (by having S reveal the exchanged key), we get:

Corollary 3.1. *Under the same conditions, it is impossible to securely realize adaptively secure deniable authentication* $\mathcal{F}_{auth}$ *and deniable key exchange* $\mathcal{F}_{ke}$ *functionalities.*

Proof. Let Π be any protocol for deniable identification using $r = r(n)$ rounds, and assume toward a contradiction that Π is adaptively secure. Without loss of generality, we assume that the receiver goes first, and that the final message of the protocol is sent by the sender. In particular, we let $\alpha_1, \alpha_2, \ldots, \alpha_r$ denote the messages sent by the receiver R and $\beta_1, \ldots, \beta_r$ denote the response messages sent by the sender S. For convenience, we let α_{r+1} denote the binary decision bit of the receiver indicating whether or not R accepted. Throughout the protocol, we denote the current state of the sender and the receiver by ω_S and ω_R, respectively. This evolving state will include all the information currently stored by the given party, except for its secret key.

Because we allow erasures, the current state does not include any information previously erased by this party.

We already stated that we only consider two kinds of attackers: sender impersonator $\mathcal{A}_S$ and receiver impersonator $\mathcal{A}_R$. The sender impersonator $\mathcal{A}_S$ will talk with an honest receiver R, while the receiver impersonator $\mathcal{A}_R$ will talk to an honest sender S. By assumption, there exists efficient simulators Sim_R and Sim_S for $\mathcal{A}_S$ and $\mathcal{A}_R$, respectively: the job of Sim_R is to simulate the behavior of R when talking to $\mathcal{A}_S$, while the job of Sim_S is to simulate the behavior of S when talking to $\mathcal{A}_R$. Moreover, the GUC security of Π implies that Sim_S and Sim_R have to work given only oracle access to R and S, respectively.[8] In particular, this means that in each round $1 \leq i \leq r$,

- As long as neither S or R is corrupted, Sim_S (resp. Sim_R) will receive some arbitrary message α_i (resp. β_i) and have to generate a "good-looking" response β_i (resp. α_{i+1}). Moreover, it has to do so *without the knowledge of the secret keys SK_S and SK_R or any of the future messages $\alpha_{i+1}, \ldots$ (resp. $\beta_{i+1}, \ldots$).*
- If S (resp. R) is corrupted, Sim_S (resp. Sim_R) will be given the secret SK_S (resp. SK_R), and will then be responsible to generate a "consistent-looking" internal state ω_S (resp. ω_R) for the corresponding party at round i. The pair (SK_S, ω_S) (resp. (SK_R, ω_R)) will then be given to the attacker and the environment.

From this description, we make our first key observation: as long as S and R are not corrupted, it is within the power of our attackers $\mathcal{A}_S$ and $\mathcal{A}_R$ to internally run the simulators Sim_S and Sim_R, respectively. In particular, we can make meaningful experiments where $\mathcal{A}_S$ runs Sim_S against an honest receiver R, or $\mathcal{A}_R$ runs Sim_R against an honest sender S. Of course, *a priori* it is unclear what happens during these experiments, since Sim_S was only designed to work against attackers $\mathcal{A}_R$ who *do not know* SK_R (as opposed to R itself, who certainly knows it), and similarly for Sim_R. Indeed, the bulk of the proof consists of showing that such "unintended" usages of Sim_S and Sim_R nevertheless result in the "expected" behavior. In particular, we give a sequence or

[8] This is because, without loss of generality, $\mathcal{A}_S$ and $\mathcal{A}_R$ are simply the dummy parties forwarding the messages of the environment, and the simulator has to work for any environment. In fact, this property follows whenever there is an external "judge" with whom the adversary may interact when gathering evidence of protocol interactions.

"hybrid experiments" culminating by showing that, without knowing the secret key of the sender, the simulator Sim_S can still successfully imitate the sender to an honest receiver, contradicting the soundness of identification.

The Hybrid Experiments. The hybrid experiments we define will be parameterized by the round number i, roughly indicating for how many rounds the attacker will run the corresponding simulator Sim_S or Sim_R before corrupting the sender and the receiver. In particular, for each round $0 \leq i \leq r-1$ we will have 8 experiments REAL-R_1^i, IDEAL-R_1^i, REAL-S_1^i, IDEAL-S_1^i, REAL-S_2^i, IDEAL-S_2^i, REAL-R_2^i, IDEAL-R_2^i (except REAL-R_1^i would be defined for $i = r$ as well). The notation above can be interpreted as follows. The "REAL/IDEAL" distinction dictate whether the corresponding sender (resp. receiver) impersonator runs against a real receiver R (resp. sender S) or against the simulator Sim_R (resp. Sim_S). In particular, the *definition* of the corresponding attacker is always given in the real world, and the corresponding ideal world is then *syntactically derived* in order to analyze the behavior of the same attacker against the simulator. The "-R/-S" suffix corresponds to the identity of the party the attacker is trying to fool (or their simulator if in the ideal world). Finally, the subscript "$_1/_2$" corresponds to whether the attacker corrupts the parties after the receiver's message α_{i+1} (in which case the attacker is denoted by $\mathcal{A}$), or after the sender's response β_{i+1} (in which case the attacker is denoted by $\mathcal{B}$). For example, the hybrid REAL-S_2^i will correspond to running the real sender S against an attacker $\mathcal{B}_R^i$ (defined in this experiment), and having $\mathcal{B}_R^i$ corrupt the parties after the flow β_{i+1}, while the hybrid IDEAL-S_2^i corresponds to running the same attacker against the simulator Sim_S.

The formal definition of all the hybrids is given in Figure 3.1, and will be further elaborated below. We mention, though, that in each of the hybrids the attacker will output a some decision bit (roughly corresponding to whether the receiver accepts in the given experiment), and we will only care about the probability that this bit is 1. To simplify the notation, we write $\Pr[X]$ to denote the corresponding probability of outputting 1 in hybrid X. Also, for hybrids X and Y, we write $X \equiv Y$ to indicate that $\Pr[X] = \Pr[Y]$ (even though the experiments might differ in other regards), and $X \approx Y$ to indicate the $|\Pr[X] - \Pr[Y]|$ is a negligible function of the security parameter. With this convention, we will show the following chain of implications for

any $0 \leq i \leq r-1$:

$$\begin{aligned} \text{REAL-}R_1^i &\approx \text{IDEAL-}R_1^i \equiv \text{IDEAL-}S_1^i \approx \text{REAL-}S_1^i \equiv \text{REAL-}S_2^i \\ &\approx \text{IDEAL-}S_2^i \equiv \text{IDEAL-}R_2^i \approx \text{REAL-}R_2^i \equiv \text{REAL-}R_1^{i+1}. \end{aligned} \tag{3.1}$$

We now proceed by examining the corresponding hybrids and showing the above chain one implication at a time.

Experiment REAL-R_1^i: attacker A_S^i against honest R

1. For $j = 1$ to i:
   - $\mathcal{A}_S^i$ receives α_j from R, and forwards this message (internally) to Sim_S.
   - $\mathcal{A}_S^i$ obtains β_j (internally) from Sim_S, and forwards this message to R.
2. $\mathcal{A}_S^i$ receives α_{i+1} from R. It then corrupts R and is given the key sk_R and the state ω_R.
3. $\mathcal{A}_S^i$ corrupts S and gets the the key sk_R (the ω_S is ignored). It then gives sk_S to Sim_S, which outputs ω'_S.
4. Using $sk_S, \omega'_S, sk_R, \omega_R$, adversary $\mathcal{A}_S^i$ runs the protocol to completion. It outputs 1 iff the result is that R outputs "accept".

Experiment IDEAL-R_1^i: attacker $\mathcal{A}_S^i$ against simulator Sim_R

1. For $j = 1$ to i:
   - $\mathcal{A}_S^i$ receives α_j (externally) from Sim_R, and forwards this message (internally) to Sim_S.
   - $\mathcal{A}_S^i$ obtains β_j (internally) from Sim_S, and forwards this message to Sim_R.
2. $\mathcal{A}_S^i$ receives α_{i+1} from Sim_R. It then corrupts R. When this happens, Sim_R gets sk_R, generates state ω'_R, and gives (sk_R, ω'_R) to $\mathcal{A}_S^i$.
3. $\mathcal{A}_S^i$ corrupts S and gets sk_S. (It does not matter what state ω_S is output by Sim_R.) $\mathcal{A}_S^i$ gives sk_S to its internal copy of Sim_S, which outputs ω'_S.
4. Using $sk_S, \omega'_S, sk_R, \omega'_R$, adversary $\mathcal{A}_S^i$ runs the protocol to completion. It outputs 1 iff the result is that R outputs "accept".

Experiment REAL-S_1^i: attacker $\mathcal{A}_R^i$ against honest S

1. For $j = 1$ to i:
   - $\mathcal{A}_R^i$ obtains α_j (internally) from Sim_R, and forwards this message to S.
   - $\mathcal{A}_R^i$ receives β_j from S, and forwards this message (internally) to Sim_R.
2. $\mathcal{A}_R^i$ obtains α_{i+1} (internally) from Sim_R.
3. $\mathcal{A}_R^i$ corrupts S, and receives in return sk_S and ω_S.
4. $\mathcal{A}_R^i$ corrupts R, and receives in return sk_R and state ω_R (that will be ignored). $\mathcal{A}_R^i$ gives sk_R to its internal copy of Sim_R, which outputs state ω'_R.
5. Using $sk_S, \omega_S, sk_R, \omega'_R$, adversary $\mathcal{A}_R^i$ runs the protocol to completion. It outputs 1 iff the result is that R outputs "accept".

Experiment IDEAL-S_1^i: attacker $\mathcal{A}_R^i$ against simulator Sim_S

1. For $j = 1$ to i:
   - $\mathcal{A}_R^i$ obtains α_j (internally) from Sim_R, and forwards this message (externally) to Sim_S.
   - $\mathcal{A}_R^i$ receives β_j (externally) from Sim_S, and forwards this message (internally) to Sim_R.
2. $\mathcal{A}_R^i$ obtains α_{i+1} (internally) from Sim_R.
3. $\mathcal{A}_R^i$ corrupts S. When this happens, Sim_S gets sk_S, generates state ω'_S, and gives (sk_S, ω'_S) to $\mathcal{A}_R^i$.
4. $\mathcal{A}_R^i$ corrupts R and gets sk_R. (It does not matter what state ω_R are output by Sim_S, since $\mathcal{A}_R^i$ will anyway ignore them.) $\mathcal{A}_R^i$ gives sk_R to its internal copy of Sim_R, which outputs state ω'_R.
5. Using $sk_S, \omega'_S, sk_R, \omega'_R$, adversary $\mathcal{A}_R^i$ runs the protocol to completion. It outputs 1 iff the result is that R outputs "accept".

Experiment REAL-S_2^i: attacker $\mathcal{B}_R^i$ against honest S

1. For $j = 1$ to i:
   - $\mathcal{B}_R^i$ obtains α_j (internally) from Sim_R, and forwards this message to S.
   - $\mathcal{B}_R^i$ receives β_j from S, and forwards this message (internally) to Sim_R.
2. $\mathcal{B}_R^i$ obtains α_{i+1} (internally) from Sim_R, and forwards this message to S. It receives in return a message β_{i+1} from S.
3. $\mathcal{B}_R^i$ corrupts S, and receives in return sk_S and state ω_S.
4. $\mathcal{B}_R^i$ corrupts R, and receives in return sk_R and state ω_R (that will be ignored). $\mathcal{B}_R^i$ gives sk_R to its internal copy of Sim_R, which outputs state ω'_R.
5. Using $sk_S, \omega_S, sk_R, \omega'_R$, adversary $\mathcal{B}_R^i$ runs the protocol to completion. It outputs 1 iff the result is that R outputs "accept".

Experiment IDEAL-S_2^i: attacker $\mathcal{B}_R^i$ against simulator Sim_S

1. For $j = 1$ to i:
   - $\mathcal{B}_R^i$ obtains α_j (internally) from Sim_R, and forwards this message to Sim_S.
   - $\mathcal{B}_R^i$ receives β_j from Sim_S, and forwards this message (internally) to Sim_R.
2. $\mathcal{B}_R^i$ obtains α_{i+1} (internally) from Sim_R, and forwards this message to Sim_S. It receives in return a message β_{i+1} from Sim_S.
3. $\mathcal{B}_R^i$ corrupts S. When this happens, Sim_S gets sk_S, generates state ω'_S, and gives (sk_S, ω'_S) to $\mathcal{B}_R^i$.
4. $\mathcal{B}_R^i$ corrupts R, and receives in return sk_R. (It does not matter what state ω_R are output by Sim_S, since $\mathcal{B}_R^i$ will anyway ignore them.) $\mathcal{B}_R^i$ gives sk_R to its internal copy of Sim_R, which outputs state ω'_R.
5. Using $sk_S, \omega'_S, sk_R, \omega'_R$, adversary $\mathcal{B}_R^i$ runs the protocol to completion. It outputs 1 iff the result is that R outputs "accept".

Experiment REAL-R_2^i: attacker $\mathcal{B}_S^i$ against honest R

1. For $j = 1$ to i:
   - $\mathcal{B}_S^i$ receives α_j from R, and forwards this message (internally) to Sim_S.
   - $\mathcal{B}_S^i$ obtains β_j (internally) from Sim_S, and forwards this message to R.
2. B_S^i receives α_{i+1} from R, and forwards this message (internally) to Sim_S. It obtains a message β_{i+1} from Sim_S.
3. $\mathcal{B}_S^i$ corrupts R, and gets sk_R and ω_R.
4. $\mathcal{B}_S^i$ corrupts S and obtains sk_S and state ω_S (which are ignored). $\mathcal{B}_S^i$ gives sk_S to its internal copy of Sim_S to obtain ω'_S.
5. Using $sk_S, \omega'_S, sk_R, \omega_R$, adversary $\mathcal{B}_S^i$ runs the protocol to completion. It outputs 1 iff the result is that R outputs "accept".

Experiment IDEAL-R_2^i: attacker $\mathcal{B}_S^i$ against simulator Sim_R

1. For $j = 1$ to i:
   - $\mathcal{B}_S^i$ receives α_j (externally) from Sim_R, and forwards this message (internally) to Sim_S.
   - $\mathcal{B}_S^i$ obtains β_j (internally) from Sim_S, and forwards this message (externally) to Sim_R.
2. B_S^i receives α_{i+1} (externally) from Sim_R, and forwards this message (internally) to Sim_S. It obtains a message β_{i+1} from Sim_S.
3. $\mathcal{B}_S^i$ corrupts R. When this happens, Sim_R gets sk_R, generates state ω'_R, and gives (sk_R, ω'_R) to $\mathcal{B}_S^i$.
4. $\mathcal{B}_S^i$ corrupts S and obtains sk_S. (It does not matter what state ω_S are output by Sim_R, since $\mathcal{B}_S^i$ will anyway ignore them.) $\mathcal{B}_S^i$ gives sk_S to its internal copy of Sim_S to obtain ω'_S.
5. Using $sk_S, \omega'_S, sk_R, \omega'_R$, adversary $\mathcal{B}_S^i$ runs the protocol to completion. It outputs 1 iff the result is that R outputs "accept".

Table 3.1: Experiments used in the impossibility proof.

Experiment REAL-R_1^i. This experiment is defined for $0 \leq i \leq r$, and defines a sender impersonator $\mathcal{A}_S^i$ interacting with an honest R. It runs the simulator Sim_S until receiving α_{i+1} from R. Then it corrupts R and learns (SK_R, ω_R). After this it corrupts S and learns SK_S. Now, it tells the internal simulator Sim_S that S was corrupted first (although this is not the case), so Sim_S only expects SK_S in order to output the simulated state ω'_S.[9] Finally, it simulates the behavior of Π until completion with the sender's side holding (SK_S, ω'_S), and the receiver's side holding (SK_R, ω_R).

Experiment IDEAL-R_1^i. By the alleged security of Π, we have REAL-$R_1^i \approx$ IDEAL-R_1^i, where the latter experiment runs the same attacker $\mathcal{A}_S^i$ against the simulator Sim_R. The main difference here is that the receiver's state ω'_R after the corruption is now simulated by Sim_R. Notice, since R is corrupted first, Sim_R only gets to know SK_R when producing ω'_R. The next implication in (3.1) is that IDEAL-$R_1^i \equiv$ IDEAL-S_1^i, but we first need to define REAL-S_1^i before talking about IDEAL-S_1^i.

Experiment REAL-S_1^i. This experiment is defined for $0 \leq i \leq r-1$, and defines a receiver impersonator $\mathcal{A}_R^i$ interacting with an honest S. It runs the simulator Sim_R until receiving α_{i+1} from Sim_R. Then it corrupts S and learns (SK_S, ω_S). After this it corrupts R and learns SK_R. Once again, it tells the internal simulator Sim_R that R was corrupted first (although this is not the case), so Sim_R only expects SK_R in order to output the simulated state ω'_R. Finally, it simulates the behavior of Π until completion with sender's side holding (SK_S, ω_S), and the receiver's side holding (SK_R, ω'_R). By the alleged security of Π, we have that REAL-$S_1^i \approx$ IDEAL-S_1^i, where the latter is expanded below.

Experiment IDEAL-S_1^i. This experiment runs the same attacker $\mathcal{A}_R^i$ against the simulator Sim_S. The main difference here is that the sender's state ω'_S after the corruption is now simulated by Sim_S. Notice, since S is corrupted first, Sim_S only gets to know SK_S when producing ω'_S. We can now verify the crucial fact that IDEAL-$R_1^i \equiv$ IDEAL-S_1^i. At the first glance, this is quite

[9] We remark that this will happen in most of our games: when the attacker internally runs the simulator Sim_P for a party P (either S or R) to produce the fake state ω_P, the attacker always tells this internal simulator that the party is corrupted first, even though, in reality, this party was corrupted second. In a moment, we will see that this subtlety is crucially used.

unexpected, since, in the one case, we have an adversary $\mathcal{A}_S^i$ interacting with Sim_R and, in the other case, we have a different adversary $\mathcal{A}_R^i$ interacting with Sim_S. However, remember that $\mathcal{A}_S^i$ really runs Sim_S internally, and $\mathcal{A}_R^i$ really runs Sim_R internally. Thus, in both experiments we are effectively running the two simulators Sim_S and Sim_R against each other.

A bit more precisely, the only syntactic difference between the two ideal experiments lies in that the order of the corruptions, which is different. A closer look, however, reveals that the order does not matter because of the way the experiments are defined. Specifically, in experiment IDEAL-R_1^i the (external) simulator Sim_R observes the corruption of R occurring first, and outputs state ω'_R without any knowledge of SK_S. In this same experiment, the (internal) simulator Sim_S is not "told" about the corruption of R, and outputs state ω'_S without knowledge of SK_R. Similarly, in experiment IDEAL-S_1^i the (internal) simulator Sim_R "thinks" that corruption of R occurs first, and outputs ω'_R without knowledge of SK_S; on the other hand, the (external) simulator Sim_S observes that corruption of S occurs first, and outputs ω'_S without knowledge of SK_R. Therefore, the probability that the adversary outputs 1 is indeed identical in each case, as claimed: IDEAL-$R_1^i \equiv$ IDEAL-S_1^i.

Experiment REAL-S_2^i. This experiment is defined for $0 \leq i \leq r-1$, and is very similar to REAL-S_1^i. (Indeed, we will shortly argue that REAL-$S_2^i \equiv$ REAL-S_1^i.) As with REAL-S_1^i which defined a receiver impersonator $\mathcal{A}_R^i$ (interacting with an honest S), here we define a slightly different receiver impersonator $\mathcal{B}_R^i$. Concretely, the only difference between the two experiments is that, in REAL-S_2^i, the adversary forwards α_{i+1} to the honest sender S (before corrupting S), while in REAL-S_1^i the adversary does not (and corrupts S right away). However, since the response β_{i+1} is computed honestly in both experiments, either by the actual sender S in REAL-S_2^i, or by the first attacker $\mathcal{A}_R^i$ running the code of S in REAL-S_1^i, this minor difference has no effect on the probability that the adversary outputs 1 in either experiment. Thus, REAL-$S_2^i \equiv$ REAL-S_1^i.

In essence, the point of this experiment is to "move a step forward" (on the sender side), so that the parties are now corrupted after the flow β_{i+1}.

Experiment IDEAL-S_2^i. By the alleged security of Π, we have that REAL-$S_2^i \approx$ IDEAL-S_2^i, where the latter experiment runs the same attacker $\mathcal{B}_R^i$ against the simulator Sim_S. In particular, the

main difference from from the real experiment is that the sender's state ω'_S after the corruption is now simulated by Sim_S. Notice, since S is corrupted first, Sim_S only gets to know SK_S when producing ω'_S.

Once again, the next implication in the chain (3.1) is that IDEAL-$S_2^i \equiv$ IDEAL-R_2^i, but we first need to define REAL-R_2^i before talking about IDEAL-R_2^i.

Experiment REAL-R_2^i. This is defined for $0 \leq i \leq r-1$, and defines a sender impersonator $\mathcal{B}_S^i$ interacting with an honest R. It runs the simulator Sim_S until receiving β_{i+1} from Sim_S. Then it corrupts R and learns (SK_R, ω_R). After this it corrupts S and learns SK_S. Once again, it tells the internal simulator Sim_S that S was corrupted first (though this is not the case), so Sim_S only expects SK_S in order to output the simulated state ω'_S. Finally, as before, it simulates the behavior of Π until completion with sender's side holding (SK_S, ω'_S), and receiver's side holding (SK_R, ω_R). By the alleged security of Π, we have REAL-$S_1^i \approx$ IDEAL-S_1^i, where the latter is expanded below.

Experiment IDEAL-R_2^i. This experiment runs the same attacker $\mathcal{B}_S^i$ against the simulator Sim_R. The main difference here is that the receiver's state ω'_R after the corruption is now simulated by Sim_R. Notice, since S is corrupted first, Sim_R only gets to know SK_R when producing ω'_R. Similar to our previous analysis of IDEAL-S_1^i and IDEAL-R_1^i, we may notice here also that experiment IDEAL-R_2^i is just a syntactic re-writing of experiment IDEAL-S_2^i. Indeed, the only difference lies in the order of corruptions; once again, however, the order does not matter because of the way the experiments are defined. The argument is analogous to the previous one: in experiment IDEAL-S_2^i the (external) simulator Sim_S observes the corruption of S occurring first, and outputs state ω'_S without knowledge of SK_R. In this same experiment, the (internal) simulator Sim_R is not "told" about the corruption of S, and outputs state ω'_R without knowledge of SK_S. Similarly, in experiment IDEAL-R_2^i the (internal) simulator Sim_S "thinks" that corruption of S occurs first, and outputs ω'_S without knowledge of SK_R; the (external) simulator Sim_R observes that corruption of R occurs first, and outputs ω'_R without knowledge of SK_S. Therefore, the probability that the adversary outputs 1 is indeed identical in each case, as claimed: IDEAL-$S_2^i \equiv$ IDEAL-R_2^i.

Experiment REAL-R_1^{i+1}**.** To complete chain (3.1), we must argue that REAL-$R_2^i \equiv$ REAL-R_1^{i+1} (for $0 \leq i < r$), where the latter experiment simply advances the already defined experiment REAL-R_1^i by one round. The argument is very similar to the proof that REAL-$\mathcal{S}_1^i \equiv$ REAL-$\mathcal{S}_2^i$, which essentially advanced the honest sender one step forward. Here, instead, we are advancing the honest receiver one step forward, by letting him compute α_{i+2}. Indeed, the only difference between the two experiments REAL-R_2^i and REAL-R_1^{i+1} is that, in REAL-R_1^{i+1}, the adversary receives α_{i+2} from the honest receiver R (before corrupting R), while in REAL-R_2^i the adversary does not (and corrupts R right away). However, since the message α_{i+2} is computed honestly in both experiments, either by the actual receiver R in REAL-R_1^{i+1}, or by the attacker $\mathcal{B}_S^i$ running the code of R in REAL-R_2^i, this minor difference has no effect on the probability that the adversary outputs 1 in either experiment. Thus, REAL-$R_1^{i+1} \equiv$ REAL-R_2^i, completing the proof of chain (3.1).

Connecting The Endpoints. Using chain (3.1) for $0 \leq i \leq r-1$, and the fact that the number of rounds r is polynomial in the security parameter, we conclude that REAL-$R_1^0 \approx$ REAL-R_1^r. To see that this leads to contradiction, let us examine these "endpoints" more closely.

Experiment REAL-R_1^0**.** Looking at Figure 3.1, here we are effectively simulating the *run of the honest sender against the honest receiver*. Indeed, on the sender's side, since the sender is corrupted before he sends any messages, and the initial state ω_S of such sender is empty, the attacker $\mathcal{A}_S^0$ is simply running the code of S. On the receiver's side, $\mathcal{A}_S^0$ lets honest R send α_1, and then corrupts R and still runs it honestly, which is equivalent to running R all the way. Since Π presumably implements $\mathcal{F}_{id}$, this means that $\Pr[\text{REAL-}R_1^0]$ is negligibly close to 1 (because the honest sender should identify to the honest receiver in the ideal model). By our proof that REAL-$R_1^0 \approx$ REAL-R_1^r, we get that $\Pr[\text{REAL-}R_1^r]$ is also negligibly close to 1. However, let us examine the latter experiment now.

Experiment REAL-R_1^r**.** Looking at Figure 3.1, here the sender impersonator $\mathcal{A}_S^r$ runs the simulator Sim_S against the honest receiver R all the way until R outputs its decision bit α_{r+1}. Only after this point does it corrupt R and S and outputs this decision bits α_{r+1}. As we just argued, $\Pr[\alpha_{r+1} = 1]$ must be negligibly close to 1. But this clearly leads to a contradiction:

without corrupting the sender S, the sender impersonator $\mathcal{A}_S^r$ caused honest R to accept with probability negligibly close to 1, which should never happen in the ideal model.

This completes the proof. □

Assuming data erasures are not allowed, we now extend the above proof to handle *forward-secure attackers*, which can only corrupt the parties *after the end* of the protocol. This is quite interesting, since many previous approaches to deniability only protected an honest sender against a malicious receiver. Here, we show that a malicious impersonator can always break the deniability of two honest parties (only one of which needs to be present), provided the distinguisher can later obtain the keys of the sender and the receiver.

To see this, we notice that the sender/receiver impersonators used in the proof always stopped the protocol somewhere in the middle, by simultaneously corrupting both parties (and then doing some internal simulation). Instead, we can let them explicitly abort the protocol at the same point in time they used to corrupt the parties, and only then to corrupt the parties and do the corresponding internal simulation (thus, the corruption only occurs after the protocol has completed). Of course, if erasures were allowed, the parties could always erase all their temporary state the moment the abort happens. However, without erasures, the corresponding simulator would still have to produce a consistent state of the party prior to the abort, and the same impossibility proof goes through. Thus,

Theorem 3.2. *If data erasures are not allowed, it is impossible to realize the identification, authentication, or key exchange functionalities with forward security (in any $\bar{\mathcal{G}}_{krk}$-hybrid model). This holds even under the remaining constraints of Theorem 3.1. In particular, semi-adaptive security (without erasures) is also impossible.*

Finally, we note that the weaker "bare public key" or "bulletin board" setup models (see Section 2.2.3) are not sufficient to realize these functionalities at all, even with mere static security. This seems to imply that key registration with knowledge is an unavoidable requirement for the PKI setup.

Theorem 3.3. *It is impossible to realize the identification, authentication, or key exchange functionalities in either the bare public key or bulletin board setup models. This impossibility holds even in the static corruption model.*

The proof of this theorem is quite simple. In either of the two setup models, the simulator has no means of obtaining *any* secret keys. Therefore, if a simulator exists for some identification protocol (in the scenario where the receiver is corrupt but the sender is honest), that same simulation procedure can be used by *anyone* in order to authenticate on behalf of the honest sender, violating the soundness of the identification protocol. We omit a more formal exposition of the proof by contradiction, since the details are trivial.

3.2.3 Gap Between Static and Adaptive Corruptions

To the best of our knowledge, previously all known tasks realizable against static attackers can still be realized against adaptive attackers (though perhaps less efficiently and/or under stronger assumptions). Thus, now demonstrate the first example of a security task where static security is not just a matter of simplicity or efficiency, but a matter of *feasibility*. Namely, we show how to GUC-realize the functionality $\mathcal{F}_{auth}$ secure against static corruptions in the PKI model, giving us the first provable separation between static and adaptive security. We accomplish this by employing a standard tool we refer to as Non-Interactive Authenticated Key Exchange (NI-AKE), which we now define formally.

Definition 3.4 (Non-Interactive Authenticated Key Exchange (NI-AKE))**.** A *Non-Interactive Authenticated Key Exchange* (NI-AKE) is given by a pair of poly-time algorithms $(\mathsf{Gen}, \mathsf{SymExt})$, satisfying the following properties:

> **Correctness of Symmetric Extraction:** For any choice of r_i, r_j, let $(PK_i, SK_i) \leftarrow \mathsf{Gen}(r_i)$ and $(PK_j, SK_j) \leftarrow \mathsf{Gen}(r_j)$. Then $\mathsf{SymExt}(SK_i, PK_j) = \mathsf{SymExt}(SK_j, PK_i)$.
>
> **Indistinguishability of Extracted Keys:** No PPT distinguisher D succeeds with more than negligible advantage at the following game:
>
> 1. The challenger samples random values $r_0 \xleftarrow{\$} \{0,1\}^\lambda$ and $r_1 \xleftarrow{\$} \{0,1\}^\lambda$, and computes $(PK_0, SK_0) \leftarrow \mathsf{Gen}(r_0)$ and $(PK_1, SK_1) \leftarrow \mathsf{Gen}(r_1)$. The values PK_0 and PK_1 are given to D.
> 2. The challenger samples a random bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, the challenger computes $k \leftarrow \mathsf{SymExt}(SK_0, PK_1)$. If $b = 1$, the challenger samples $k \xleftarrow{\$} \{0,1\}^\lambda$. The value k is

then given to D.

3. D computes a guess bit $\hat{b} \leftarrow D(PK_0, PK_1, k)$.
4. D's advantage is defined to be $\Pr[\hat{b} = b] - 1/2$.

The basic intuition for this definition is that availability of "authenticated" public keys might allow the parties to agree upon a shared secret k, where the availability of the sender's secret key and the receiver's public key (or the receiver's secret key and the sender's public key) enables the sender (and receiver) to compute k non-interactively (*i.e.*, no further communication needs to occur once the parties possess each other's public keys).

The standard technique for implementing NI-AKE is based on the Decisional Diffie Hellman (DDH) assumption, as follows. Assume we will operate in some multiplicative group $\mathbb{G}$ of order q generated by some generator g, where the DDH assumption[10] holds. If each party P_i has a secret key $x_i \in \mathcal{Z}_q$ and corresponding public key $y_i = g^{x_i}$, then P_i and P_j will non-interactively share a common key $g^{x_i x_j} = y_i^{x_j} = y_j^{x_i}$ (*i.e.*, $\mathsf{SymExt}(x_i, y_j) := {y_j}^{x_i}$). However, by the DDH assumption, the key value $g^{x_i x_j}$ still looks like a random group element to an attacker who only knows the public keys y_i and y_j.

Given any secure NI-AKE, our statically secure authentication protocol is very simple: to authenticate a message M to a receiver R, the sender simply computes a Message Authentication Code (MAC) tag for the tuple (sid, S, R, M) using the key k that he non-interactively shares with the receiver, while the receiver verifies the tag using the same key. A bit more formally,

Theorem 3.5. *Assuming the existence of a NI-AKE* $(\mathsf{Gen}, \mathsf{SymExt})$*, there exists an efficient protocol* Φ *such that* $\mathcal{F}_{auth}$ *can be UC-realized in the* $\bar{\mathcal{G}}_{krk}^{\Phi}$*-hybrid model, with static security.*

Proof. As already stated, the Gen function will define the public and secret keys for each party, and Φ uses SymExt to obtain a MAC key k_{ij} for use between a given pair of parties P_i and P_j. To authenticate a message M, send M along with the MAC of $(\mathsf{sid}, P_i, P_j, M)$, where the MAC can be implemented using a pseudo-random function. We briefly sketch how to do the simulation, omitting the details of the formal proof of simulation indistinguishability (which are quite standard).

[10] DDH states that the tuple (g, g^a, g^b, g^{ab}) is computationally indistinguishable from (g, g^a, g^b, g^c), where a, b, c are random in $\mathcal{Z}_q$.

If both parties are honest, the simulation initially samples a random key k_{ij}, which it will use in place of the legitimate shared key (as the simulator does not have access to either party's secret key, so it cannot compute the actual shared key value). To authenticate a message, the simulator computes the MAC using k_{ij} and allows the recipient to accept the ideal authentication if and only if the MAC tag is allowed to pass through the network unaltered by the (simulated) adversary. Since a staticly corrupting adversary can never learn the secret keys corresponding to either of these two honest parties, the key k_{ij} chosen by the simulator remains indistinguishable from the legitimate shared key to the adversary. Indistinguishability of this simulation procedure follows directly from the security properties of the NI-AKE, and from the unforgeability property of the MAC.

If one party P_i is corrupt, retrieving the party's secret key SK_i allows the simulator to run $\mathsf{SymExt}(SK_i, PK_j)$ to get the same shared MAC key k_{ij} as the one obtained by its honest partner P_j via $\mathsf{SymExt}(SK_j, PK_i)$. This holds with all but negligible probability even if the corrupt party P_i chose malicious randomness r_i for Gen, as allowed by the $\bar{\mathcal{G}}_{krk}$ functionality. Given this key, the simulator can perfectly simulate the behavior of the honest party. Indistinguishability of this simulation procedure follows trivially. □

Remark: The above protocol is not forward secure (let alone adaptively secure), even if we allow erasures. Intuitively, the adversary can detect simulated authentication from real authentication between two honest parties by later corrupting either one of the honest parties, deriving the appropriate MAC key with SymExt, and then checking the previously recorded MACs. Still, given our impossibility results, this protocol based on using a MAC with NI-AKE essentially matches the best we can do if we insist on realizing $\mathcal{F}_{auth}$ in the non-erasure model. We conjecture that it is also impossible to realize $\mathcal{F}_{auth}$ with a forward secure protocol even if erasures are allowed, and that therefore there is no remaining gap for further improvement in this direction.

Remark: Although it might appear that NI-AKE is being used to realize $\mathcal{F}_{ke}$, and that here we are combining $\mathcal{F}_{ke}$ with a MAC in order to realize $\mathcal{F}_{auth}$, this is *not* the case. NI-AKE does not trivially realize $\mathcal{F}_{ke}$, since the exchanged key is *fixed*. That is, parties running multiple instances

of $\mathcal{F}_{ke}$ expect to see fresh keys for each instance of the key exchange protocol, but any number of instances of NI-AKE will always output the same key. However, we can easily combine NI-AKE with a highly efficient *symmetric key authenticated encryption* scheme in order to realize $\mathcal{F}_{ke}$. Intuitively, the protocol is the same as the one above, only the MAC is replaced by a (symmetric key) authenticated encryption of the fresh random key (which is being exchanged). The proof security is entirely analogous, except we also rely upon the privacy property of the authenticated encryption in order to hide the exchanged key from the adversary.

3.2.4 Equivalence of $\mathcal{F}_{auth}$ to Shared Key Model

In this section we show that if parties have access to a basic key exchange functionality $\mathcal{F}_{ke}$, as described in Figure 3.3, then it is possible to realize $\mathcal{F}_{auth}$ with adaptive security. Conversely, if parties are given access to $\mathcal{F}_{auth}$, it is possible to realize $\mathcal{F}_{ke}$ with adaptive security. Aside from showing equivalence (under appropriate assumptions) between deniable authentication and deniable key exchange, which is a simple extension of existing work applied to our setting, we discuss the importance of this result in terms of contrasting public key infrastructure (PKI), implied by the $\bar{\mathcal{G}}_{krk}$ functionality, and "symmetric key infrastructure" (SKI), implied by the $\mathcal{F}_{ke}$ functionality.

Contrasting SKI and PKI. To obtain a symmetric key infrastructure (*i.e.*, a shared key setup) from $\mathcal{F}_{ke}$, we would like to ensure that each pair of parties need only invoke one instance of $\mathcal{F}_{ke}$, at initialization time. The resulting shared key should then be reused in all future sessions. One way of ensuring this one-time setup would be to define a separate shared functionality for symmetric key registration, akin to that of $\bar{\mathcal{G}}_{krk}$. Indeed, with some care, it is possible to formalize such one-time functionality. However, for the sake of simplicity,[11] we will instead apply the "JUC

[11] A complete formal definition of the shared functionality for symmetric key setup is somewhat difficult to work with, since it adds the additional complexity of interaction at setup time to the already subtle issue of protecting keys from being leaked by protocols other than Φ. With some care, however, these details could be easily filled in. In practice, this approach seems counter-intuitive anyway, since the main idea here is to replace the one-time symmetric key setup functionality by a realization using a key exchange protocol, rather than using a trusted party to implement the shared functionality (as in the case of PKI).

Theorem" [31], which will effectively allow us to obtain the same conclusion. The JUC Theorem approach enables us to realize the "multi-session extension" of $\mathcal{F}_{auth}$ (*i.e.*, a single instance of $\mathcal{F}_{auth}$ which can authenticate an arbitrary number of messages), using only a single instance of $\mathcal{F}_{ke}$ at initialization time. Whereas the JUC Theorem does not help us to model public key infrastructure, since the public keys must be made available to the environment (and indeed, this is the reason why we need to use the GUC framework), we do not encounter this difficulty in the symmetric key setting. Its easy to see why: if the shared keys are directly provided to each pair of communicating parties at setup time, no information about those keys is ever published to the environment (in reality, there is no certification authority publicizing keys as there was for the PKI model).

With this discussion in mind, one can interpret the equivalence between $\mathcal{F}_{auth}$ and $\mathcal{F}_{ke}$ to mean that a Symmetric Key Infrastructure is sufficient to realize $\mathcal{F}_{auth}$ with security against adaptive corruptions, whereas Public-Key Infrastructure is not. And this leads to the following puzzling observation. It is already well known that secure channels alone are not sufficient to UC-realize most useful two-party functionalities, such as bit commitments or zero knowledge [19, 23, 29]. On the other hand, the main result of [6] (as well as the upcoming result in Chapter 4) shows that $\bar{\mathcal{G}}_{krk}$ is sufficient to UC/GUC-realize any well-formed functionality with adaptive security, *assuming the availability of secure channels*. The upshot is that, absent physically secure channels, our current feasibility results imply that *both PKI and SKI* are needed in order to UC/GUC-realize most useful two party functionalities with adaptive security.[12]

Equivalence of $\mathcal{F}_{ke}$ and $\mathcal{F}_{auth}$. Both directions are fairly simple and appear in the literature in different contexts. To implement $\mathcal{F}_{auth}$ (an even the multi-session extension of $\mathcal{F}_{auth}$) from $\mathcal{F}_{ke}$ with adaptive security, we essentially need to MAC the given message using the ideal shared key that the parties obtain from $\mathcal{F}_{ke}$. (This is similar to the statically secure protocol in Section 3.2.3, except the NI-AKE piece obtained using the PKI is replaced here by an ideal call to $\mathcal{F}_{ke}$.) Similarly, in the other direction, we show how to realize $\mathcal{F}_{ke}$ given $\mathcal{F}_{auth}$, using a

[12] Alternatively, Chapter 4 shows that an Augmented Common Reference String (ACRS) can be used to replace the PKI. Therefore, it is also possible UC-realize useful two party functionalities given both ACRS and SKI. We remark that the impossibility of implementing $\mathcal{F}_{auth}$ using PKI easily extends to include ACRS setups alongside the PKI.

natural protocol based on non-committing public key encryption (NCE) [22]. Intuitively, NCE schemes can operate just like normal encryption schemes, but it is also possible to generate indistinguishable "rigged" public keys and ciphertexts, such that the ciphertexts can be made to appear as if they contain any desired plaintext (with some appropriate choice of secret key). This rigging allows the simulator to ensure that a transcript simulated between two honest parties can be made to appear as though they exchanged any particular symmetric key (a crucial property for deniability, in the event that symmetric keys are ever leaked). Informally, the protocols is quite simple: the initiator of the exchange sends a public key pk of the NCE to the responding party over an authenticated channel (given by $\mathcal{F}_{auth}$), and the responding party replies with a ciphertext containing key k encrypted under the non-committing public key pk.

We now formally prove the the equivalence of $\mathcal{F}_{auth}$ to the Shared Key model.

Theorem 3.6. *If one way functions exist, then there exists a protocol Φ_{mac} that UC-realizes $\mathcal{F}_{auth}$ in the $\mathcal{F}_{ke}$ hybrid model, even against adaptive corruptions. Furthermore, there also exists a protocol that UC-realizes the natural multi-session extension of $\mathcal{F}_{auth}$ in the $\mathcal{F}_{ke}$ hybrid model, requiring only a single call to $\mathcal{F}_{ke}$, even against adaptive corruptions.*

Proof of Theorem 3.6. Our protocol will make use of a standard cryptographic Pseudo-Random Function (PRF) family, denoted $\mathbb{F} = \{f_k \mid k \in \{0,1\}^\lambda\}$ where for a random $k \in \{0,1\}^\lambda$ we have that the function $f_k : \{0,1\}^* \to \{0,1\}^\lambda$ is computationally indistinguishable from a random function to any distinguisher that is not given k (such function families are known to exist if one way functions exist). Since the multi-session version of the protocol Φ_{mac} implies the existence of the single-session version (and indeed, is essentially identical to it), we will describe only the multi-session protocol. Intuitively, the protocol is to obtain a symmetric key (per pair of communicating parties), and use the PRF with that key as a "Message Authentication Code" (MAC) to authenticate the message together with its sub-session identifier. See Figure 3.4 for a formal description of the protocol.

To prove that the above protocol emulates $\mathcal{F}_{auth}$ even in the presence of adaptive corruptions, we describe a UC-simulator for protocol Φ_{mac}, and prove that for any adversary $\mathcal{A}$, the output of $\mathcal{S}$ is indistinguishable to all environments $\mathcal{Z}$. $\mathcal{S}$ uses the standard approach of running a copy of $\mathcal{A}$ internally, and forwarding all communications between $\mathcal{A}$ and $\mathcal{Z}$. Recall that $\mathcal{S}$ will

Protocol Φ_{mac} for Realizing $\mathcal{F}_{auth}$

The protocol Φ_{mac}, running with parties S and R (the sender and recipient, respectively), proceeds as follows:

Initialization: The first time S wishes to communicate with R, the following initialization protocol is run.

1. S invokes $\mathcal{F}_{ke}$ with input $(\texttt{keyexchange}, \mathsf{sid}, S, R)$, and then waits to receive a message $(\texttt{setkey}, \mathsf{sid}, S, R, k)$ from $\mathcal{F}_{ke}$. Upon receipt, S records the message containing the key.
2. R invokes $\mathcal{F}_{ke}$ with input $(\texttt{keyexchange}, \mathsf{sid}, S, R)$, and waits to receive a message $(\texttt{setkey}, \mathsf{sid}, S, R, k)$ from $\mathcal{F}_{ke}$. Upon receipt, R records the message containing the key.

Authentication: To authenticate a message M from S to R with sub-session identifier ssid, the following authentication protocol is run.

1. S computes $\tau = f_k(\mathsf{sid} \| \mathsf{ssid} \| M)$ and sends $(\mathsf{sid}, \mathsf{ssid}, S, R, M, \tau)$ to R.
2. Upon receipt of a message $(\mathsf{sid}, \mathsf{ssid}, S, R, M, \tau)$ from S, R checks if $\tau \stackrel{?}{=} f_k(\mathsf{sid} \| \mathsf{ssid} \| M)$, and outputs $(\texttt{authentic}, \mathsf{sid}, \mathsf{ssid}, S, R, M)$ if so. Otherwise, the message is ignored.

Figure 3.4: Protocol Φ_{mac} for realizing the multi-session extension of $\mathcal{F}_{auth}$ in the $\mathcal{F}_{ke}$-hybrid model.

control the transmission delays of $\mathcal{F}_{auth}$, so not only will $\mathcal{S}$ receive notifications when S attempts to authenticate a message M to R, but $\mathcal{S}$ can decide whether or not to allow R to receive the message. Upon seeing an attempted message authentication, $\mathcal{S}$ must simulate the behavior of the "real world" authentication protocol Φ_{mac} for S and R (including all the protocol communication, which takes place over insecure channels). To accomplish this, $\mathcal{S}$ will simply run the protocol Φ_{mac} on behalf of S and R, as they would in the real world, but $\mathcal{S}$ will play the role of $\mathcal{F}_{ke}$ whenever S or R invoke it as part of their protocol. Playing the role of $\mathcal{F}_{ke}$ allows $\mathcal{S}$ to learn the symmetric key k (either by observing the key set by a corrupt party R, or by choosing the key when the parties are honest) – which, in turn, allows $\mathcal{S}$ to honestly simulate computations in Φ_{mac} which depend upon knowledge of k. To ensure that the output distributions of the parties are accurately simulated, the simulator will have to handle two concerns: 1) Recalling that $\mathcal{F}_{auth}$ immediately notifies $\mathcal{S}$ when a message is sent by S, whenever an honest sender S sends message m to R, $\mathcal{S}$ will delay the delivery of m to R until a (simulated) session of protocol Φ_{mac} results in R accepting the same authenticated message m. 2) Whenever an honest recipient R accepts some message m from S in the simulated interaction of protocol Φ_{mac}, then $\mathcal{S}$ activates an instances of $\mathcal{F}_{auth}$ on behalf of the sender (which *must* be corrupt, or else $\mathcal{S}$ cannot activate $\mathcal{F}_{auth}$ on the sender's behalf) with message m for recipient R. The only case not covered by either case 1 or 2 occurs when both S and R are honest, yet a simulated interaction of Φ_{mac} for R accepts some message m that was not actually sent by S (that is, in the equivalent "real world" interaction, an honest R would accept such an authenticated message from S even though S never sent it). In this case, we say that the simulation has failed – but we note that this occurs with at most negligible probability by security of the PRF family, since only $\mathcal{S}$ knows k (and $\mathcal{S}$ will never compute a valid tag τ for any message that was not actually sent by S, although the adversary may see many valid tags for messages with different sids). Whenever this unlikely failure condition does not occur, the simulation of the protocol is entirely honest (*i.e.*, the simulated interaction is identically distributed to the real world interaction, including the outputs of the parties), it follows immediately that it is indistinguishable to $\mathcal{Z}$.

□

Theorem 3.7. *Assuming the existence of non-committing encryption, there exists a protocol that UC-realizes $\mathcal{F}_{ke}$ in the $\mathcal{F}_{auth}$-hybrid model, even against adaptive corruptions.*

Proof of Theorem 3.7. Here we will only sketch the proof, which is a trivial consequence of the secure message transmission property of non-committing encryption used with authenticated channels (in fact, [34] essentially defines non-committing encryption as the realization of the secure message transmission functionality in the authenticated channels model).

Let $(\mathsf{NCGen}, \mathsf{NCEnc}, \mathsf{NCDec}, \mathsf{NCSim}, \mathsf{NCEqv})$ be a NCE scheme. Then the protocol for realizing $\mathcal{F}_{ke}$ proceeds as follows:

1. S generates $(pk, sk) \leftarrow \mathsf{NCGen}()$ and sends a (public) authenticated message of the form $(\mathtt{keyexchange}_1, \mathsf{sid}, S, R, pk)$ to R.

2. R samples a random key $k \xleftarrow{\$} \{0,1\}^\lambda$ and computes $\gamma \leftarrow \mathsf{NCEnc}(pk, k)$, then sends a (public) authenticated message $(\mathtt{keyexchange}_2, \mathsf{sid}, S, R, \gamma)$ to S.

In standard fashion, the UC-simulator $\mathcal{S}$ for this protocol will run an internal copy of the real-world adversary $\mathcal{A}$, and feed it with simulated protocol flows (while simply forwarding all communications taking place between $\mathcal{A}$ and $\mathcal{Z}$). To simulate the protocol when the parties are honest at the outset, $\mathcal{S}$ will generate pk and γ using NCSim. When the receiver or sender is corrupt at the outset, the simulation is to just run the honest protocol on behalf of the honest party. To handle an adaptive corruption of either party after the first message, the simulator simply chooses a random message and runs NCEqv with it to obtain the corresponding sk and random coins r^* that make it appear as if (pk, sk) were produced by NCGen (these coins will be used to simulate internal state of the sender if need be). The protocol then proceeds on behalf of the remaining honest party as if the corrupted party had been corrupt at the outset. To handle adaptive corruption of a party after the completion of the protocol, $\mathcal{S}$ will learn the key k from $\mathcal{F}_{ke}$ which was exchanged by the honest parties, and then using NCSim to produce consistent coins for both S and R that make it appear is if γ contained a legitimate encryption under pk of k.

The proof that the simulated interaction is indistinguishable from the real interaction is a straightforward series of reductions to the security properties of NCE. The most complicated

case arises when the honest parties are corrupted at the end of the protocol, and in this case we use the full strength of NCE to show that the simulator can make it appear as if the protocol contained a key exchange of key k, which it did not know when it generated the protocol flows (even including appropriate state information for both S and R, in the non-erasure model). □

Remark: It is not necessary to use a non-committing encryption scheme in the protocol for realize $\mathcal{F}_{ke}$ if we only concern ourselves with security against static corruptions. In that case, standard public key encryption schemes will clearly suffice (this follows from trivial simplifications to the proof above, by removing all adaptive corruptions from the analysis). An even more efficient method of realizing $\mathcal{F}_{ke}$ with security against static corruptions is described in the final remarks of Section 3.2.3.

3.3 A Mostly Deniable Key Exchange Protocol

Given the result of the previous sections, one might wonder if it possible to achieve adaptive security in the UC model at all, whenever physically secured channels are unavailable – message authentication is impossible in this case (as is symmetric key exchange). Indeed, with the exception of key exchange protocols, virtually all known constructions of UC secure protocols already assume the availability authenticated channels. Must we abandon all those protocols? In this section, we will give one possible avenue for salvaging existing adaptively secure protocols from this seemingly catastrophic situation: we show how to implement UC authenticated channels that support adaptively secure protocols, even though the process of establishing the authenticated channel is not itself adaptively secure.

In particular, we will establish authenticated channels by using a special one-time symmetric key exchange protocol, and employing the result of Section 3.2.4 to build authenticated channels on top of the key exchange. Of course, realizing $\mathcal{F}_{ke}$ itself is impossible, even if we only ask for forward security, as per our previous discussion.

Therefore, instead of attempting to realize $\mathcal{F}_{ke}$ directly, we will consider a similar functionality, dubbed *Key Exchange with Incriminating Abort* $\mathcal{F}_{keia}$ (shown in Figure 3.5), which provides the

adversary the additional capability to ask for a protocol *abort*. Whenever the functionality is not asked to abort, it behaves identically to $\mathcal{F}_{ke}$. However, if it is asked to abort, the functionality allows the adversary to somehow *incriminate* one of the participating parties (namely, S). We must also make one additional concession to the impossibility result – our realization of $\mathcal{F}_{keia}$ can only tolerate semi-adaptive corruptions. (Given that we will be allowing the protocol to incriminate S whenever it aborts, it makes sense that we cannot allow fully adaptive corruptions – after all, the adversary could always ensure that the protocol does not abort by adaptively corrupting the parties immediately before an abort occurs.)

In practice, since we will only run this key exchange once (at setup time), forward security is much more important than tolerating fully adaptive corruptions. If the key exchange protocol succeeds (with no adaptive corruption occurring during the protocol execution), then we can still use adaptively secure protocols realized in the $\mathcal{F}_{ke}$-hybrid model, and they will retain their adaptive security. Again, intuitively, the only difference between the $\mathcal{F}_{keia}^{\mathsf{IncProc}}$-hybrid model and the $\mathcal{F}_{ke}$-hybrid model is that parties may be incriminated when the key exchange is aborted – *before* any other protocols requiring the $\mathcal{F}_{ke}$-hybrid model (such as the realization of $\mathcal{F}_{auth}$ given in Section 3.2.4) are even run. In other words, $\mathcal{F}_{keia}$ *almost* represents a deniable realization of $\mathcal{F}_{ke}$: if we could somehow guarantee that $\mathcal{F}_{keia}$ never aborts, then it would GUC-realize $\mathcal{F}_{ke}$ (and, in turn, so would any protocol realizing $\mathcal{F}_{keia}$).

Remark: Ironically, any protocol that GUC-realizes $\mathcal{F}_{keia}$ is a "deniable realization" of $\mathcal{F}_{keia}$, but $\mathcal{F}_{keia}$ itself is not intuitively "deniable". The situation becomes clearer when viewing $\mathcal{F}_{keia}$ itself as separate cryptographic tool that attempts to realize $\mathcal{F}_{ke}$ but falls short of the mark (due to the possibility of aborts). Then we may comfortably say that, of course, a "deniable realization" of $\mathcal{F}_{keia}$ is not a "deniable realization" of $\mathcal{F}_{ke}$. As long as we consider deniability in terms of the task being realized, rather than the question of whether or not the protocol itself is "deniable", then we may avoid confusion.

In order to model the incrimination, $\mathcal{F}_{keia}$ is directly described in the $\bar{\mathcal{G}}_{krk}$-hybrid model, and it is assumed that $\mathcal{F}_{keia}$ will be supplied with access to the secret keys of the activating parties. Such access to keys will not be used by $\mathcal{F}_{keia}$ unless an abort occurs, in which case $\mathcal{F}_{keia}$ will

Functionality $\mathcal{F}_{keia}^{\mathsf{IncProc}}$

$\mathcal{F}_{keia}$, which is parameterized by an "incrimination procedure" $\mathsf{IncProc}$, and a security parameter λ proceeds as follows, when running in the $\bar{\mathcal{G}}_{krk}$-hybrid model with parties S and R (who have already registered secret keys SK_S and SK_R, respectively) and adversary $\mathcal{S}$:

1. Upon receiving a message of the form $(S, \mathtt{keyexchange}, \mathsf{sid}, S, R, SK_S)$ from party S, if there are no previous records, then record the value $(\mathtt{keyexchange}, \mathsf{sid}, S, R, SK_S)$, mark S "active", and a send public delayed output $(\mathtt{keyexchange}, \mathsf{sid}, S, R)$ to R. (Otherwise, ignore the message.)

2. Upon receiving a message of the form $(R, \mathtt{keyexchange}, \mathsf{sid}, S, R, SK_R)$ from party R, if R is not yet "active", mark R as "active" and send a public delayed output $(\mathtt{active}, \mathsf{sid}, S, R)$ to S. (Otherwise, ignore the message.)

3. Upon receiving a message of the form $(\mathtt{setkey}, \mathsf{sid}, S, R, k')$ from $\mathcal{S}$, if R is corrupt and S is "active", then output $(\mathtt{setkey}, \mathsf{sid}, S, R, k')$ to S and R, and halt. If R is "active" but not corrupt, then sample a fresh key $k \xleftarrow{\$} \{0,1\}^\lambda$ and send the message $(\mathtt{setkey}, \mathsf{sid}, S, R, k)$ to R. Furthermore, if S is "active", then send the delayed message $(\mathtt{setkey}, \mathsf{sid}, S, R, k)$ to S as well. In all cases, this completes the protocol, and the functionality halts.

4. Upon receiving a message of the form $(\mathtt{abort}, \mathsf{sid}, S, R)$ from $\mathcal{S}$, if S is "active", send $(\mathtt{abort}, \mathsf{sid}, S, R)$ as a delayed message to S and mark S "aborted". If R is "active", send $(\mathtt{abort}, \mathsf{sid}, S, R)$ as a delayed message to R (*i.e.*, $\mathcal{S}$ need not notify either party that the protocol was aborted, and may still cause R to output a key using a `setkey` message, but cannot cause S to output a key once an abort has occurred).

5. Upon receiving a message of the form $(\mathtt{incriminate}, \mathsf{sid}, S)$ from $\mathcal{S}$, if this is the first time receiving such a message and S is currently "aborted" and honest, then run the procedure $\mathsf{IncProc}(\mathsf{sid}, S, R, PK_S, PK_R, SK_S)$.

Figure 3.5: The ideal functionality for Key Exchange with Incriminating Abort, parameterized by an incrimination procedure $\mathsf{IncProc}$ which runs only if the key exchange is aborted by the adversary.

use the secret key of S to produce some kind of "evidence" that S attempted to exchange keys with R (the secret key of R is never used). The nature of this "evidence" will depend on the details of our realization of $\mathcal{F}_{keia}$, but its affect on the security in the ideal model will be limited – intuitively, it can only provide some evidence that S used $\mathcal{F}_{keia}$ attempting to talk to R, but will not do any further harm security in the ideal world. This is primarily due to the restriction that the PKI modeled by $\bar{\mathcal{G}}_{krk}$ is used by honest parties *only* to run the key exchange protocol. To make the basic structure of the functionality as general as possible, we parameterize $\mathcal{F}_{keia}$ by an *incrimination procedure*, IncProc, which is some additional code in the functionality that can be changed to suit different realizations of $\mathcal{F}_{keia}$.

Intuition: The main complication to understanding the security guarantees provided by $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ is that it is not clear what "damage" the incrimination procedure IncProc might do after a protocol abort. When considering the security provided by $\mathcal{F}_{keia}$ in the ideal model, it helps to recall that honest parties will not use $\bar{\mathcal{G}}_{krk}$ for anything other than key exchange.[13] Once any particular IncProc has been specified, irrespective of the kind of information from S's secret key that IncProc may give to the adversary, the potential security consequences of the information revealed to the adversary in the ideal model can be classified into two criteria: 1) it might allow the adversary to prove that S attempted a key exchange with R that aborted (since the adversary can produce information which can't be simulated without access to S's secret key, and that can only be obtained via an aborted key exchange), and 2) the security of future key exchanges involving S may be affected, since some information about S's secret key has been leaked. It should be clear that, given any particular IncProc, if we can successfully realize $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ then the security of future key exchanges must not be affected by aborts (otherwise, the realization would be insecure). So, in truth, we only need to worry about the first kind of consequence. Indeed, the whole point of giving the adversary access to IncProc is that the adversary will obtain some information that it could not have simulated unless S first invoked $\mathcal{F}_{keia}$ in an aborted attempt to do a key exchange with R – and this is precisely the kind of effect that we mean by "incrimina-

[13] For simplicity, our definition of $\bar{\mathcal{G}}_{krk}$ only requires that honest parties do not use their *secret keys* with other protocols. Technically, this is fine, as long as the implications of doing otherwise have been properly accounted for. That is, any protocols designed to use $\bar{\mathcal{G}}_{krk}^{\Phi}$ must obviously take the information leaked by Φ into account.

tion". Therefore, $\mathsf{IncProc}$ should capture the kind of incriminating information that is obtained by the real world adversary after a protocol abort (as precisely as possible). This allows us to accurately model incrimination (*i.e.*, non-simulatable protocol flows) in the ideal world for the first time.

Finally, it is important to remember that $\mathcal{F}_{keia}$ requires access to the secret keys of the parties, and, therefore, $\bar{\mathcal{G}}_{krk}$ must be parameterized with the code of the ideal protocol for $\mathcal{F}_{keia}$ (to enable honest parties running $\mathcal{F}_{keia}$ to retrieve their own secret keys). As a notational convenience, wherever we write $\bar{\mathcal{G}}_{krk}^{\Phi}$ in reference to a protocol Φ for realizing $\mathcal{F}_{keia}^{\mathsf{IncProc}}$, we assume that $\bar{\mathcal{G}}_{krk}$ is also implicitly parameterized with the code of $\mathcal{F}_{keia}^{\mathsf{IncProc}}$. Also, recall that corruptions are assumed to be be PID-wise, so a party is considered to be either honest in all sessions or corrupt in all sessions.

At the core of our constructions for realizing $\mathcal{F}_{keia}$ is a technique we refer to as *Dual Receiver Encryption* (DRE), described above in Section 2.5. DRE schemes guarantee that both parties who are recipients of a given ciphertext will decrypt it in the same way. This will allow us to ensure a symmetry between S and R that enables either of them to simulate the actions of the other party in response to any particular protocol flow (yet still preventing the adversary from seeing the flows when the parties are honest).

Our suggested protocol for realizing $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ is summarized in Figure 3.6. Intuitively, the incrimination procedure $\mathsf{IncProc}$ will expect the adversary to supply a ciphertext that matches the form of ψ_1 in the protocol below, and will then use S's secret key to decrypt ψ_1 and compute a corresponding flow ψ_2. The incrimination procedure hands ψ_2 to the adversary, along with the random coins used to construct the encryption and the key pair for the non-committing encryption scheme.

We now state and prove a formal theorem regarding the realizability of $\mathcal{F}_{keia}^{\mathsf{IncProc}}$.

Theorem 3.8. *Assuming the existence of a Dual Receiver Encryption scheme* $(\mathsf{Gen}, \mathsf{DREnc}, \mathsf{DRDec})$ *and a Non-Committing Encryption scheme* $(\mathsf{NCGen}, \mathsf{NCEnc}, \mathsf{NCDec}, \mathsf{NCSim}, \mathsf{NCEqv})$*, there is a protocol* Φ_{dre} *that realizes* $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ *in the* $\bar{\mathcal{G}}_{krk}^{\Phi_{dre}}$*-hybrid model with semi-adaptive security, where* $\mathsf{IncProc}(\mathsf{sid}, S, R, PK_S, PK_R, SK_S)$ *proceeds as follows:*

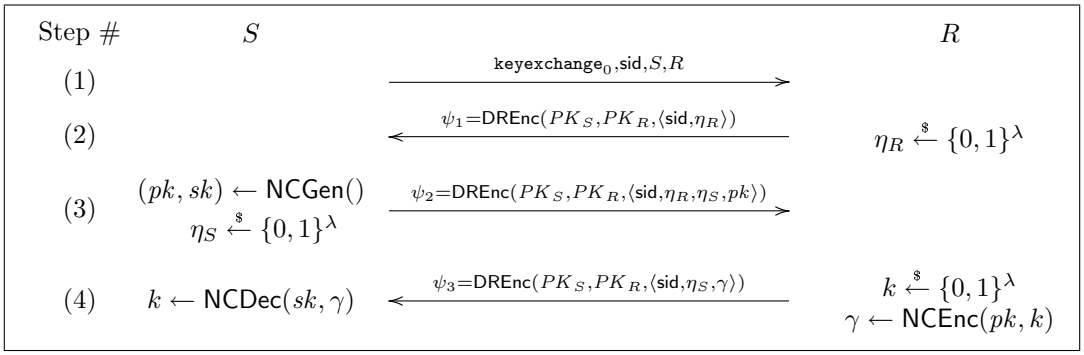

Figure 3.6: A graphical illustration of Protocol Φ_{dre} for realizing $\mathcal{F}_{keia}$. S and R check consistency of each flow immediately upon receiving it; if the flow is not consistent, the protocol is aborted.

1. *Upon receiving a message of the form* $(\texttt{incriminate}_1, \mathsf{sid}, S, R, \psi_1)$ *from* $\mathcal{S}$*, the functionality computes* $\langle \mathsf{sid}', \eta_R \rangle \leftarrow \mathsf{DRDec}(SK_S, PK_R, \psi_1)$*. If* $\mathsf{sid}' = \mathsf{sid}$*, the functionalities continues by sampling a random nonce* $\eta_S \xleftarrow{\$} \{0,1\}^\lambda$ *along with random coins* $r^*, r_\psi \xleftarrow{\$} \{0,1\}^\lambda$*, generating a fresh key pair for NCE* $(pk, sk) \leftarrow \mathsf{NCGen}(r^*)$*, and then computing the dual-receiver encryption* $\psi_2 \leftarrow \mathsf{DREnc}(r_\psi; PK_S, PK_R, \langle \mathsf{sid}, \eta_R, \eta_S, pk, \rangle)$*. Otherwise,* $\psi_2 \leftarrow \perp$*. Finally, the functionality responds by sending the message* $(\texttt{incriminate}_2, \mathsf{sid}, S, R, \psi_2, r^*, r_\psi)$ *to* $\mathcal{S}$*.*

Furthermore, Φ_{dre} *also realizes* $\mathcal{F}_{ke}$ *in the* $\bar{\mathcal{G}}_{krk}^{\Phi_{dre}}$*-hybrid model with static security. We also observe that here* $\mathsf{IncProc}$ *can be simulated using the secret key of* R *instead of* S*, so in practice, it can only incriminate one of the pair* $\{S, R\}$*, and not specifically* S*.*

Remark: As we will see, the incrimination procedure must provide the random coins r^* and r_ψ to $\mathcal{S}$ in order to enable it to simulate consistent semi-adaptive corruptions of S without erasures, after an aborted protocol run occurs. If we were to allow S to random coins used in aborted protocol runs, it would only be necessary for $\mathsf{IncProc}$ to provide ψ_2. However, given that r^* and r_ψ are freshly chosen by $\mathsf{IncProc}$ independent of S's secret key, it does not do significant "harm" to leak this additional information in the ideal model. Indeed, even leaking the full decryption of of ψ_1 would not do considerable harm in this instance – however, we strive to leak the minimal amount of information necessary to make $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ realizable.

Proof of Theorem 3.8. We begin by specifying the protocol we use.

Protocol Φ_{dre} for Realizing $\mathcal{F}_{keia}$

The protocol Φ_{dre}, when running with parties S and R (who have already retrieved each other's public keys, as well as their own secret keys), proceed as follows (we assume that parties S and R receive messages of the expected form at each step, or else they abort):

1. S sends the message $(\texttt{keyexchange}_0, \mathsf{sid}, S, R)$ to R.
2. R samples a random value $\eta_R \xleftarrow{\$} \{0,1\}^\lambda$, computes $\psi_1 \leftarrow \mathsf{DREnc}(PK_S, PK_R, \langle \mathsf{sid}, \eta_R \rangle)$. R then sends $(\texttt{keyexchange}_1, \mathsf{sid}, S, R, \psi_1)$ to S.
3. S computes $\langle \mathsf{sid}', \eta_R \rangle \leftarrow \mathsf{DRDec}(SK_S, PK_R, \psi_1)$. If $\mathsf{sid}' \neq \mathsf{sid}$, then S aborts. Otherwise, S samples a random value $\eta_S \xleftarrow{\$} \{0,1\}^\lambda$, generates a fresh key pair for NCE $(pk, sk) \leftarrow \mathsf{NCGen}()$, and computes $\psi_2 \leftarrow \mathsf{DREnc}(PK_S, PK_R, \langle \eta_R, \eta_S, pk, \rangle)$. S then sends $(\texttt{keyexchange}_2, \mathsf{sid}, S, R, \psi_2)$ to R.
4. R computes $\langle \eta_R', \eta_S, pk \rangle \leftarrow \mathsf{DRDec}(SK_R, PK_S, \psi_2)$. If $\eta_R' \neq \eta_R$, then R aborts. Otherwise, R samples a random session key $k \xleftarrow{\$} \{0,1\}^\lambda$ and computes $\psi_3 \leftarrow \mathsf{DREnc}(PK_S, PK_R, \langle \eta_S, \mathsf{NCEnc}(pk, k) \rangle)$. R then sends $(\texttt{keyexchange}_3, \mathsf{sid}, S, R, \psi_3)$ to S. R then outputs k as its session key.
5. S computes $\langle \eta_S', \gamma \rangle \leftarrow \mathsf{DRDec}(SK_S, PK_R, \psi_3)$. If $\eta_S' \neq \eta_S$ then S aborts. Otherwise, S outputs outputs $\mathsf{NCDec}(sk, \gamma)$ as its session key.

Figure 3.7: Protocol Φ_{dre} for realizing $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ using Dual Receiver Encryption.

In order to simplify the proof that Φ_{dre} GUC-realizes $\mathcal{F}_{keia}^{\mathsf{IncProc}}$, we will make use of the EUC model (relying upon the proof of equivalence from Chapter 1). Recall that the EUC model is essentially identical to the standard UC modeling, only the environment $\mathcal{Z}$ is allowed direct access to the shared functionality (in our case, that is $\bar{\mathcal{G}}_{krk}^{\Phi_{dre}}$). That is, we will only need to show that a single instance of Φ_{dre} with session sid, interacting with $\mathcal{A}$, can be substituted by a single instance of $\mathcal{F}_{keia}$ interacting with $\mathcal{S}$ – even when the environment $\mathcal{Z}$ is allowed to invoke $\bar{\mathcal{G}}_{krk}^{\Phi_{dre}}$ (using parties with any session IDs other than sid). Unfortunately, things are slightly complicated by the ability of the environment to invoke many sessions of Φ_{dre} and $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ that use the same instance of $\bar{\mathcal{G}}_{krk}^{\Phi_{dre}}$ (although those sessions will not share the same sid with the protocol we are substituting). In particular, this empowers the environment to use any information about party's secret keys that it might obtain from other sessions of Φ_{dre} and $\mathcal{F}_{keia}^{\mathsf{IncProc}}$. Therefore, throughout the proof, we will need to argue that providing the environment with access to such information does not help it distinguish the simulation. Fortunately, in the case of our protocol, this is a

straightforward argument since the session id sid is included in all the ciphertexts that might be revealed by other protocol sessions. Therefore, even if the environment can learn ciphertexts for other sessions, they will not be useful in constructing flows for the session under attack.

We construct our EUC simulator $\mathcal{S}$, which runs an internal copy of $\mathcal{A}$, as follows.

Handling interactions between $\mathcal{A}$ and $\mathcal{Z}$: All messages between $\mathcal{Z}$ and $\mathcal{A}$ are forwarded directly by $\mathcal{S}$ between its internal copy of $\mathcal{A}$ and the external environment $\mathcal{Z}$.

Simulating with honest S and honest R: Whenever $\mathcal{S}$ receives a notification from $\mathcal{F}_{keia}$ that S is attempting to exchange a key with R, it initiates a simulation of protocol Φ_{dre} on behalf of S and R. The simulator chooses random values for η_R and η_S, and uses NCSim to produce $\hat{pk}, \hat{\gamma}$, and α. The simulator then computes the honest flow in Step 2 from R to S, and "sends" the flow over the channel in its internal simulation of $\mathcal{A}$. Now there are two possible continuations: 1) If $\mathcal{A}$ allows this protocol flow to pass unaltered, then $\mathcal{S}$ continues the process by computing and sending (internally) the corresponding honest flow in Step 3 for S (however, pk was generated via NCSim instead of the NCGen used in the real interaction). The simulator then continues by responding according to the protocol for R in Step 4, except that $\hat{\gamma}$ is used instead of a freshly encrypted key. If $\mathcal{A}$ alters this final flow, the simulator will cause S to abort, but will still send a `setkey` to $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ to cause R to output a random key. 2) If $\mathcal{A}$ alters the first flow of the protocol (in Step 2) to contain a ciphertext ψ_1', then $\mathcal{S}$ will immediately issue an `abort` message to $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ in order to gain access IncProc. However, $\mathcal{S}$ will not yet deliver the abort notification to S or R. To compute the flow from S in Step 3 of the protocol, $\mathcal{S}$ will invoke IncProc using the ciphertext ψ_1'. This will return some ciphertext ψ_2' along with some random coins r^* and r_ψ. The simulator can then ψ_2' on behalf of S, and record the random coins for future use in the event that S is corrupted after the protocol aborts. If $\mathcal{A}$ sends any further message to S (or R), then $\mathcal{S}$ sends the `abort` message to S (or R).

We note that if either honest party is not first marked as "active" in the key exchange protocol, the simulator will refuse to send protocol flows on that party's behalf (intuitively, this is because the environment has not instructed the party to conduct a key exchange). Therefore, all protocol flows originating with $\mathcal{A}$ will be considered "altered" flows, and the

simulation for the activated party can proceed as above, except that no abort notifications are passed to the inactive party.

Even if the key exchange protocol aborts, we observe that the simulator knows all the random coins necessary to simulate the internal state of both S and R at all times. Namely, the simulator has the random coins used to form any protocol flows which it sent on behalf of those parties, even in the case that it sends a flow constructed by $\mathsf{IncProc}$ (since $\mathsf{IncProc}$ additionally provides $\mathcal{S}$ with the random coins used by the incriminating flow, except for the nonces themselves, which $\mathcal{S}$ can obtain by decrypting ψ_2' with either secret key). The only complication arises when the protocol between two honest parties completes successfully. In this case, the simulator had sent some value $\hat{gamma}$ in the final flow from R of Step 4. Once the one of the party's is corrupted, the simulator will learn which key k was chosen by the ideal functionality $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ (it cannot know this value while both parties remain honest). In order to ensure that the state of the corrupt party is consistent, after learning k post-corruption, $\mathcal{S}$ will immediately run $\mathsf{NCEqv}(\hat{pk}, \hat{\gamma}, \alpha)$ to produce coins r_γ and r^* such that $(pk = \hat{pk}, sk) \leftarrow \mathsf{NCGen}(r^*)$ and $\hat{\gamma} = \gamma \leftarrow \mathsf{NCEnc}(r_\gamma; pk, k)$. The simulator will then provide r^* (and the implied sk) as internal state for S and r_γ as internal state for R, making it appear as if the simulated key exchange honestly resulted in session key k.

Simulating with a corrupt R or corrupt S: The simulation for this case is straightforward – the simulator will use its knowledge of the corrupt party's secret key to decrypt all the dual-receiver ciphertexts received by the honest party. It will then send responses in accordance with the real world protocol actions for the honest party. Successful and aborted key exchanges are simulated naturally in the ideal world by issuing corresponding requests to $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ on behalf of $\mathcal{S}$, and appropriately controlling the message delays. There is never any need for the simulator to make use of $\mathsf{IncProc}$ in this scenario (*i.e.*, when one of the parties is corrupt at the outset). The only subtlety arises when the simulator needs to form the last flow on behalf of an honest R, since it will need to know what value of k to use. The simulator can obtain k before forming the final protocol flow sent on behalf of an honest R (if need be) by issuing `setkey` to $Fkeia^{\mathsf{IncProc}}$ before sending the flow in Step 4, and "catching" the output sent to corrupt S (of course, the honest R will successfully

output key k just before the simulator has actually sent the final flow on its behalf, but this is acceptable since R always outputs key key k if it sends the final flow at all). Therefore, in all circumstances, the protocol flows can be honestly computed. Furthermore, handling a subsequent semi-adaptive corruption of the remaining honest party is trivial, since once again the simulator possesses all the random coins used to produce protocol flows sent on behalf of the honest party.

We now sketch a brief indistinguishability proof for the above simulation.

Indistinguishability of the honest/honest simulation: To see that the simulation is indistinguishable, we need to argue that the protocol flows are indistinguishable and that the outputs of the honest parties in the simulated interaction correspond to their outputs in the real interaction.

If $\mathcal{A}$ does not attempt to alter any of the protocol flows between S and R, then it is easy to see that (by the security properties of NCE) $\mathcal{S}$ yields indistinguishable interactions. This holds even if S or R is later corrupted (again, by security properties of NCE). If the protocol runs to completion for R, then R will output a random key in both the real and ideal interactions. Similarly, if the protocol runs to completion for S, then R must also have run to completion, and S will output the same random session key as R in both the real and ideal interactions. Note that the random session key that appears in the transcript will correctly match the keys output by S and/or R whenever $\mathcal{A}$ is able to decrypt the DRE, since as soon as $\mathcal{A}$ corrupts a party to get the secret key $\mathcal{S}$ can use the NCE to make it appear as though the simulated transcript resulted in a key exchange output of particular session key k.

If $\mathcal{A}$ alters any flow of the protocol at all, our construction of $\mathcal{S}$ will not allow S to output a key. This strategy can only fail if $\mathcal{A}$ can indeed come up with "valid" protocol flows that were not sent by R in this session, and yet will cause S to output a session key in the real world. Since S encrypts a nonce η_S in Step 3 of the protocol using DRE, and expects to see the same nonce in the response of Step 4 before agreeing to accept the session key, we can see that this situation only occurs if $\mathcal{A}$ can somehow send a "valid" DRE encryption containing η_S back to S. It is easy to see that, even if $\mathcal{A}$ can forward flows to other sessions

of protocol Φ_{dre} or $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ run by S or R, he can only produce such a "valid" ciphertext with negligible probability. (Notably, ciphertexts cannot be forwarded between different sessions since they contain the session id.) Therefore, the output of S is properly simulated in this case with all but negligible probability. A symmetric argument will show that the same holds true of R's output, in the event that the message from S to R in Step 3 is tampered with.

It remains only to argue the indistinguishability of the transcript (with respect to semi-adaptive corruptions) after such message tampering. This is entirely straightforward, with the only subtle case arising when $\mathcal{A}$ alters the first flow from R to S sent in Step 2. In this case $\mathcal{S}$ must invoke $\mathsf{IncProc}$, or else it would not be able to construct a protocol flow that remains indistinguishable in the face of a future corruption (of either S or R). However, we note that by invoking $\mathsf{IncProc}$, $\mathcal{S}$ performs the same operations that an honest party S would perform in Step 3 of the protocol. Furthermore, $\mathsf{IncProc}$ provides $\mathcal{S}$ with all the information needed to computing appropriate internal state for S. Thus, $\mathcal{S}$ produces indistinguishable interactions in all cases where the parties are initially honest.

Indistinguishability when one party is corrupt: Whenever one party is corrupt at the outset, the simulator conducts an entirely honest simulation, since the security properties of DRE guarantee that all protocol flows will decrypt the same way, whether using the secret key of the honest party or that of the corrupt party. Thus the real and ideal interactions are identically distributed. It is also easy to verify that the output of the honest party is correctly simulated as well.

□

Notably, although we can only realize $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ when we consider security against adaptive corruptions, *the same protocol* we use to realize $\mathcal{F}_{keia}^{\mathsf{IncProc}}$ in this setting is also a *static secure* realization of $\mathcal{F}_{ke}$. Therefore, we have achieved a strictly stronger notion of security than the natural non-interactive protocol using NI-AKE and MACs. Honest parties are always guaranteed complete deniability when the protocol succeeds, and even if the protocol aborts, deniability is maintained until some future corruption of a party occurs. It is an open question whether this notion of deniability can be further improved upon.

4 • A Feasibility Result for Realizing General Functionalities

4.1 Deniable Realizations of General Cryptographic Tasks

Now that we have considered the issue of deniable authentication, we turn our attention to more general cryptographic tasks. As per our discussion in Section 3.1, we will focus on "deniable realizations" of such tasks. Before attempting to construct deniable realizations, we proceed with a case study of security flaws related to deniability that appear in previous work, focusing on works in the *Common Reference String* model (see Section 2.2). The main result of [30] is to demonstrate the feasibility of UC-realizing general two-party and multi-party tasks in the CRS model. However, as we will see, the UC-realizations of [30] do not provide deniability in practice – a warning sign that the security modeling employed in [30] somehow fails to adhere tightly to reality.

For instance, consider the ideal functionality for Zero Knowledge (ZK) – we expect that any secure realization of that functionality should reveal no information to the adversary beyond the output of the ideal functionality (which contains only a single bit). In particular, the ideal functionality's output can easily be computed entirely without the help of the prover (or his secrets), and thus the prover should be able to completely deny his participation in zero-knowledge proof protocols since they never reveal any information that would otherwise be unobtainable to the adversary. However, we already know from the result of [73] that it is impossible to achieve such deniability for ZK in the CRS model. Indeed, we may see that the UC simulator for ZK functionality in [30] chooses a fresh CRS, and generates the simulated protocol transcripts with respect to that, instead of the published real-world CRS. Thus, if a protocol transcript makes use of the real-world CRS, it could not have been obtained via simulation (and therefore a successful prover is indeed incriminated by the transcript).

Since there is no deniability, the adversary is truly able to obtain valuable information by observing protocol interactions that would not be revealed by the ideal functionality. Thus we

have found a practical example of an actual security loss that directly results from the disparity between reality and some aspect of the security modeling inherent in the CRS technique of [30]. But what went wrong?

To gain a better understanding of the issue, it is helpful to consider the issue of deniability in light of the "real world" resources required in order to run the GUC simulator to simulate a given protocol session. If the resources required to simulate a protocol session are readily available, then we say the protocol session is *plausibly deniable* (since it is plausible that information obtained from the protocol was the result of a simulation). If the resources required to simulate are difficult or impossible to obtain, then there is no guarantee of plausible deniability (since it will be difficult to convince others that an incriminating protocol transcript was the result of a simulation). We wish to employ simulation techniques that require only minimal resources to conduct a simulation, increasing the plausibility of denials (as well as decreasing the value of any information that an adversary might obtain by attacking a secure protocol). Thus, we might use the term *fully simulatable* to refer to any plausibly deniable realizations of tasks in the GUC framework.

From this vantage point, we observe that the resource required to conduct the protocol simulations in [30] is a "trapdoor" for the CRS. In particular, the CRS must be "rigged" with such a trapdoor *a priori*. Such rigging is certainly not plausible when there is a trusted party choosing the CRS, and this is in fact the root of the deniability problem for the CRS model. Furthermore, knowledge of such a trapdoor imparts the ability to completely violate security of any protocol constructed using the techniques of [30], and thus there would be no security against any entity that was also capable of simulating protocols. (Similarly, in the "imaginary angel" model of [78], the simulator requires access to super-polynomial time functionalities that are certainly not plausibly available in the real world – and thus, the deniability problem arises there as well. Furthermore, if the "imaginary angels" of [78] were to somehow be made practical in the real world, all security would be lost.)

To take the analysis a step further, we may understand that the problem originates with the ideal functionality modeling the CRS setup. As was already discussed, in the basic UC framework of [20], no mechanism was provided for modeling state information that was shared by multiple protocol sessions. The JUC Theorem of [31] alleviated this situation somewhat by

providing a means for multiple sessions of the same protocol (or a particular fixed set of jointly designed protocols) to share state information within the confines of the existing UC modeling. However, the JUC Theorem is not concerned with the issue of *global* shared state that may directly influence the environment outside of the protocol execution. In [30], the JUC Theorem is employed in order to model the reusable nature of the CRS, but this modeling does not reflect the *global availability* of the CRS, since *the environment can observe the CRS* too.

The end result is that in the [30] modeling of a CRS, the environment can only learn about the CRS via the adversary. This means that the simulator (the "ideal adversary" that defines the capabilities of the attacker in the UC framework) has the ability to *lie to the environment* about the CRS. Surely, the ability to deceive the entire "outside world" (as represented by the environment) about the value of the CRS is quite powerful. Ironically, it is precisely because the real world adversary *obviously* lacks the ability to perpetrate such deceptions that deniability is lost. As we mentioned in Section 3.1, deniability can be lost even if the security model is "too strong" due to modeling an adversary that is more powerful than the real one. That is precisely what has happened here.

Now that we understand where the modeling went astray, we can employ the tools of the GUC framework to correct the problem – instead of preventing the environment from directly observing the CRS, we model a truly global CRS which the environment is fully entitled to access directly. That is, the simulator no longer has the power to deceive the environment about the CRS – consequently, the simulator will not be able to get any information about the CRS (such as a trapdoor) that the real-world adversary would be unable to obtain. We now investigate the consequences of this more realistic modeling.

4.2 Insufficiency of the Global CRS Model

In fact, as we will now demonstrate, a global CRS setup is *not* sufficient to GUC-realize even the basic two-party commitment functionality. Indeed, we will further elaborate the nature of this insufficiency by considering some weaknesses in the security of previously proposed constructions in the CRS model.

This section shows that the simple CRS model is insufficient for GUC-realizing the simple bit

commitment functionality $\mathcal{F}_{com}$ (even in the secure channels model). Let us elaborate.

Recall that many interesting functionalities are unrealizable in the UC framework without any setup assumption. For instance, as we mentioned earlier, it is easy to see that the ideal authentication functionality, $\mathcal{F}_{auth}$, is unrealizable in the plain model. Furthermore, many two party tasks, such as Commitment, Zero-Knowledge, Coin-Tossing, Oblivious Transfer and others cannot be realized in the UC framework by two-party protocols, even if authenticated communication is provided [23, 29, 19].

As a recourse, the common reference string (CRS) model was used to re-assert the general feasibility results of [51] in the UC framework. That is, it was shown that any "well-formed" ideal functionality can be realized in the CRS model [23, 30]. However, the formulation of the CRS model in these works postulates a setting where the reference string is given *only to the participants in the actual protocol execution.* That is, the reference string is chosen by an ideal functionality, $\mathcal{F}_{crs}$, that is dedicated to a given protocol execution. ($\mathcal{F}_{crs}$ is the ideal functionality analogous to our $\bar{\mathcal{G}}_{crs}$ shared functionality from Section 2.2.1, but as a consequence of being an ideal functionality rather than a shared functionality, it gives the reference string only to the adversary and the participants involved in one particular protocol execution.) Intuitively, this formulation means that, while the reference string need not be kept secret to guarantee security, it cannot be safely used by other protocol executions. In other words, no security guarantees are given with respect to executions that use a reference string that was obtained from another execution rather than from a dedicated instance of $\mathcal{F}_{crs}$. (The UC with joint state (JUC) theorem of [31] allows multiple executions of certain protocols to use the same instance of the CRS, but it requires all instances that use the CRS to be carefully designed to satisfy some special properties.)

In contrast, we are interested in modeling a setting where the same CRS is globally available to all parties and all protocol executions. This means that a protocol π that uses the CRS must take into account the fact that the same CRS may be used by other *arbitrary* protocols, even protocols that were specifically designed to interact badly with π. Using the GUC security model defined in Chapter 1, we define this weaker setup assumption as a *shared* ideal functionality that provides the value of the CRS not only to the parties of a given protocol execution, but rather to all parties, and even directly to the environment machine. In particular, this global CRS functionality, $\bar{\mathcal{G}}_{crs}$, exists in the system both as part of the real protocol execution and as part

of the ideal process. (The functionality $\bar{\mathcal{G}}_{crs}$ can be found in Figure 2.1 above.)

We demonstrate that $\bar{\mathcal{G}}_{crs}$ is insufficient for reproducing the general feasibility results that are known to hold in the $\mathcal{F}_{crs}$ model. Furthermore, no global setup that provides *only* publicly available information (*e.g.*, such as the "random oracle" described in Section 2.2.4) suffices to enable general feasibility results in the GUC model. To exemplify this fact, we show that no two-party protocol that uses such a publicly accessible setup as its only setup assumption GUC-realizes the ideal commitment functionality, $\mathcal{F}_{com}$ (presented in Figure 4.1). The proof follows essentially the same steps as the [23] proof of impossibility of realizing $\mathcal{F}_{com}$ in the plain model. For instance, the reason that these steps can be carried out even in the presence of $\bar{\mathcal{G}}_{crs}$ is, essentially, that the simulator must obtain the reference string from an external entity ($\bar{\mathcal{G}}_{crs}$), rather than generating the reference string by itself. We conjecture that most other impossibility results for UC security in the plain model can be extended in the same way to hold for GUC security in the presence of $\bar{\mathcal{G}}_{crs}$ and general publicly accessible setups.

Functionality $\mathcal{F}_{com}$

Commit Phase: Upon receiving a message $(\texttt{commit}, \mathsf{sid}, S, R, b)$ from party S, where $b \in \{0,1\}$, record the value b and send the message $(\texttt{receipt}, \mathsf{sid}, S, R)$ to R and the adversary. Ignore any future `commit` messages.

Reveal Phase: Upon receiving a message $(\texttt{reveal}, \mathsf{sid})$ from S: If a value b was previously recorded, then send the message $(\texttt{reveal}, \mathsf{sid}, b)$ to R and the adversary and halt. Otherwise, ignore.

Figure 4.1: The Commitment Functionality $\mathcal{F}_{com}$ (see [23])

Theorem 4.1. *Let* Pub *denote an arbitrary PPT "oracle" (Interactive Turing Machine) that returns only public information – that is, the computation performed by* Pub *is independent of the particular* pid*s which are querying it. There exists no bilateral, terminating protocol π that GUC-realizes $\mathcal{F}_{com}$ and uses only the shared functionality for* Pub*. This holds even if the communication is ideally authentic. (In particular, we note that* Pub $= \bar{\mathcal{G}}_{crs}$ *is one such public oracle.)*

Proof. Intuitively, the proof of the impossibility of UC commitments (for the plain model) described in [23] holds here as well, since a Pub-externally constrained environment $\mathcal{Z}$ is able to

obtain direct access to the oracle Pub by invoking a separate dummy party specifically to forward oracle queries and responses to Pub, thus preventing the simulator $\mathcal{S}$ from intercepting the queries and producing responses on its own (say, in order to arrange knowledge of a trapdoor). The basic proof technique is to argue that if there is a simulator which can observe commitments sent by a corrupt party and extract an appropriate input to the ideal functionality from them, then this same simulator can be used by the adversary to extract commitments sent by honest parties. Since all parties concerned (including the environment) have access to Pub, and since Pub provides the same interface to all parties (whether or not they are corrupt), it is essentially straightforward to argue this.

More formally, suppose that there exists a commitment protocol π (for a party S committing a bit b to a party R) and a simulator $\mathcal{S}$ such that $\text{EXEC}_{\mathcal{F}_{com},\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ for any adversary $\mathcal{A}$ and any Pub-externally constrained environment $\mathcal{Z}$ (here we may even allow $\mathcal{S}$ to depend on the choice of $\mathcal{A}$ and $\mathcal{Z}$). We will arrive at a contradiction.

We accomplish this by constructing a new Pub-externally constrained environment $\mathcal{Z}'$ and a new adversary $\mathcal{A}'$ such that there is *no* simulator $\mathcal{S}'$ which can satisfy $\text{EXEC}_{\mathcal{F}_{com},\mathcal{S}',\mathcal{Z}'} \approx \text{EXEC}_{\pi,\mathcal{A}',\mathcal{Z}'}$. Recall that a Pub-externally constrained environment may invoke dummy parties running $\text{IDEAL}_{\text{Pub}}$ using any unique $(\mathsf{pid}, \mathsf{sid})$, and thus may obtain the same publicly available information that any party can obtain.

Our $\mathcal{A}'$ is constructed so as to corrupt the recipient R at the beginning of the protocol. During the protocol, $\mathcal{A}'$ will run the algorithm for $\mathcal{S}$ and feeding it with oracle responses from the same Pub oracle that $\mathcal{A}'$ has access to, and using the same party and session identities for S and R in this "virtual" run of $\mathcal{S}$. Furthermore, while acting as the environment for this copy of $\mathcal{S}$, $\mathcal{A}$ will "corrupt" the party "S" in the virtual view of $\mathcal{S}$. Whenever $\mathcal{A}'$ receives protocol messages from the honest party S in the real protocol execution, it sends the same messages on behalf of the "corrupt party S" in the virtual view of $\mathcal{S}$. Whatever messages $\mathcal{S}$ would send on behalf of the "honest" virtual recipient "R", $\mathcal{A}'$ will send on behalf of the real party R (which it has previously corrupted in the real world). At some point, $\mathcal{S}$ must send the message $(\texttt{commit}, \mathsf{sid}, S, R, b')$ to the commitment functionality. At this point, the adversary $\mathcal{A}'$ will output the bit b', and halt.

We define the environment $\mathcal{Z}'$ to choose a random bit b, and provide it as the input for the honest committer S. If the adversary outputs b' such that $b' = b$, then $\mathcal{Z}'$ outputs 1 (and 0

otherwise). (Additionally, we implement any trivial interface for $\mathcal{Z}'$ to forward its Pub oracle access to $\mathcal{A}'$.) Observe that no decommitment ever occurs, and thus the view of $\mathcal{S}'$ must be independent of the choice of b (meaning that $\mathcal{S}'$ must be correct with probability 1/2). However, as $\mathcal{S}$ must produce a bit b' that matches b with all but negligible probability (since we assumed it simulates the protocol π correctly), $\mathcal{A}'$s guess b' must match b with high probability, and thus $\mathcal{Z}'$ will clearly distinguish between the guesses of $\mathcal{A}'$ and those of $\mathcal{S}'$ (which are correct with probability exactly 1/2).

□

Remark: We note that this impossibility result also implicitly rules out the use of the shared key model by the result of Section 3.2.4, since we assume the presence of authenticated channels. In other words, a "symmetric key infrastructure" (SKI) does not help. On the other hand, we will soon see that a public key infrastructure (PKI) (in combination with authenticated channels) *does* suffice to realize $\mathcal{F}_{com}$. This contrast is particularly striking in light of the result of Section 3.2.2, which states that one cannot realize adaptively secure authenticated channels using a PKI, despite the fact that SKI suffices for that task. Ultimately, if we wish to realize $\mathcal{F}_{com}$ with true adaptive security over unauthenticated channels, an SKI is *necessary* but *not sufficient*, but the combination of SKI with PKI is *sufficient* but *not necessary* (*e.g.* an ACRS setup can be used to replace the PKI).

Essentially, this impossibility result says that neither the CRS functionality $\bar{\mathcal{G}}_{crs}$, nor the Random Oracle functionality $\bar{\mathcal{G}}_{ro}$ (nor any combination thereof) can be used to realize $\mathcal{F}_{com}$ in the GUC model. Another consequence of this result is that no global setup that is implemented in a completely *non-interactive* fashion can suffice for realizing $\mathcal{F}_{com}$ (since it must inherently be public). At the other extreme, note that the $\bar{\mathcal{G}}_{ro}$ model is already already so strong that it cannot truly be realized without the use of a fully interactive trusted party, and yet it still does not suffice to avoid the impossibility result. In the next section, we study the problem of realizing $\mathcal{F}_{com}$ using setup assumptions with minimal interaction requirements.

4.3 Realizing Fully Simulatable General Computation

We now turn our attention to the problem of *constructing* fully simulatable GUC-secure protocols. That is, we would like it to be possible for a real-world adversary to simulate the effects of any attack on a protocol (in a computationally indistinguishable manner), without actually conducting the attack on the protocol (instead utilizing only the information that would be revealed by an ideally secure realization). The impossibility result of Section 4.2 implies that we cannot do this in the standard CRS model (if we correctly model the global availability of the CRS). Thus, we must consider alternatives to the CRS model if we hope to achieve our goal.

Since we must somehow avoid the impossibility result of Section 4.2 for the CRS model, we would like to find reasonable alternative global setup assumptions that allow for realizing interesting tasks. That is, we are looking for shared functionalities $\bar{\mathcal{G}}$ (as defined in Chapter 1), so that on the one hand $\bar{\mathcal{G}}$ will be implementable in reality with reasonable trust assumptions, and on the other hand we will have protocols that GUC-realize interesting functionalities and still use no setup (*i.e.*, no ideal functionalities) other than $\bar{\mathcal{G}}$. We say that such GUC-secure protocols are "fully simulatable" since the GUC-simulator for attacking the ideal protocol can, in a very practical sense, be run directly by the adversary. This allows the adversary to simulate *the same information* that can be gotten by attacking any session of the real protocol, without the need to actually perform an attack. (Of course, this simulated information is inherently useless to the adversary, since the ideal protocol attacked by the simulation is secure by definition.)

We first observe that if the system is equipped with a "fully interactive trusted party" that realizes, say, $\mathcal{F}_{mcom}$, the multi-session variant of $\mathcal{F}_{com}$, by interacting separately and privately with each session, then we can directly use the protocol of [30] to GUC-realize any "well-formed" functionality. However, we would like to find more reasonable global setup assumptions, and in particular assumptions that require less interaction from the trusted entity. (Indeed, this realization requires the trusted party to perform strictly more work than it would by directly computing the desired functionalities, *i.e.*, the trivial realization of ideal model functionalities). Although it is clear that we can achieve fully simulatable protocols by using highly interactive trusted parties to compute functionalities, it seems to be a more difficult problem to realize GUC-secure protocols using an "offline" shared functionality. Indeed, by our earlier impossibility

results, *some* degree of interaction would seem to be essential, so we begin by considering the idea of limiting the interaction to a "registration phase".

We observe that the "key registration with knowledge (KRK)" setup of [6], properly modified to serve as a shared functionality (see $\bar{\mathcal{G}}_{krk}$ in Section 2.2.3), allows us to GUC-realize any "well-formed" ideal functionality with security against static corruptions (using the techniques of that work). Although the setup phase is interactive since parties must register their public keys with registration authority, it is possible to show (with some minor modifications) that the protocol of [6] can allow the trusted party to remain "offline" for all subsequent protocol activity.

Theorem 4.2. *The [6] protocol GUC-realizes $\mathcal{F}_{zk}$, even when given access only to $\bar{\mathcal{G}}_{krk}$, as long as the party corruptions are non-adaptive and PID-wise.*

The proof of this theorem is by a natural extension of the proof in [6] to the EUC framework (which is, of course, equivalent to GUC), but surprisingly, we can achieve a much stronger goal than mere GUC security against static corruptions that requires an interactive setup.

4.3.1 Global Setup with Minimal Interaction: The Augmented CRS Model

Although it may seem that *at least* an interactive "registration phase" is required in order to avoid our earlier impossibility result, we show that something even *less interactive* will suffice. We introduce a further simplification of $\bar{\mathcal{G}}_{krk}$ that approaches the simplicity and non-interactivity of the $\bar{\mathcal{G}}_{crs}$ model, denoted $\bar{\mathcal{G}}_{acrs}$. Surprisingly, we show how to construct a protocol that GUC-realizes $\mathcal{F}_{com}$ (and thus any well-formed functionality) having access only to $\bar{\mathcal{G}}_{acrs}$. Unlike $\bar{\mathcal{G}}_{krk}$, the $\bar{\mathcal{G}}_{acrs}$ shared functionality does not *require* any interaction (much like $\mathcal{F}_{crs}$), but merely offers a one-time use interactive "key retrieval" service to those who choose to use it. Therefore, we refer to this new setup assumption as the *Augmented CRS* (ACRS) model (see Section 2.2.2 for full details). In particular, protocols realized in the ACRS model will not actually make use of the key retrieval service, since the model only allows corrupt parties to retrieve their keys. Thus, we are assured that honest parties need never communicate interactively with $\bar{\mathcal{G}}_{acrs}$ – in fact, honest parties cannot even directly distinguish between a global CRS and a global ACRS setup (yet, the CRS setup is insufficient to realize $\mathcal{F}_{com}$, whereas the ACRS setup suffices!).

Somewhat counter-intuitively, it is even crucial that uncorrupted parties in ACRS model never "bother" to obtain their secret keys from the trusted authority (since even an honest party may inadvertently execute a rogue protocol, which might expose the secret key). Similarly, it is crucial that corrupted parties have access to their secret keys, since otherwise they would be unable to conduct attack simulations. (On a side note, security is still guaranteed to honest parties who obtain their keys and use them to conduct attack simulations *provided that* they only use their keys for simulation purposes. This is a direct consequence of the "equivalence" between the information revealed by attack simulations, and the information that can be obtained via real-world attacks.) To enforce the protocol design criteria that honest parties should not require access to their secret keys, the $\bar{\mathcal{G}}_{acrs}$ functionality is defined so that it refuses to supply secret keys to honest parties. Of course, a direct realization of $\bar{\mathcal{G}}_{acrs}$ by a trusted party would not attempt to actually determine which parties are honest, yet intuitively it is easy to see that this modeling will still suffice. Namely, it is not problematic even if the real-world trusted party gives keys to honest parties, as long as they are careful to protect their own security by keeping their keys secret (and, therefore, honest parties will act just as if they do not possess their keys).

While we discuss the details of the ACRS model above in Section 2.2.2, a few additional salient points are worth mentioning in the present context.

Comparing $\bar{\mathcal{G}}_{krk}$ and $\bar{\mathcal{G}}_{acrs}$. The main difference between $\bar{\mathcal{G}}_{acrs}$ and $\bar{\mathcal{G}}_{krk}$ is that in $\bar{\mathcal{G}}_{acrs}$ there is a single public value, whereas in $\bar{\mathcal{G}}_{krk}$ an extra public value must be given per party identity. Using a paradigm analogous to the identity-based encryption of [13], we avoid the use of per-party public keys and replace them with a single *short* "master public key" (and indeed our constructions use short public keys that depend only on the security parameter). This property, combined with the fact that the parties who follow their protocols never obtain their secret keys, makes $\bar{\mathcal{G}}_{acrs}$ very close in spirit to a global CRS setup as in $\bar{\mathcal{G}}_{crs}$. In fact, in light of the far-reaching impossibility result for $\bar{\mathcal{G}}_{crs}$, $\bar{\mathcal{G}}_{acrs}$ can be regarded as a "minimum interaction" global setup.

We note that, as pointed out in [6], $\bar{\mathcal{G}}_{krk}$ can be naturally implemented by multiple "registration authorities", where no single authority needs to be fully trusted by all. (However, we once again stress that $\bar{\mathcal{G}}_{krk}$ requires *all* parties, *even those who honestly follow their protocols*,

to interactively register with a *some* authority and obtain a public key.) Similarly, although $\bar{\mathcal{G}}_{acrs}$ with a short public key would naturally seem to call for a realization by a single trusted entity, the same technique applies and several instances of $\bar{\mathcal{G}}_{acrs}$ may be run by different trusted authorities. Unlike $\bar{\mathcal{G}}_{krk}$, however, parties may participate in protocols while placing their trust in an arbitrary trusted authority, without ever having registered with *any* authority. This is extremely useful for settings where PKIs are not desirable or easy to implement, and where no single "global" authority is available (*e.g.*, see [5]).[1]

In the next section, we will prove the following result:

Theorem 4.3. *There exists a protocol that GUC-realizes $\mathcal{F}_{com}$ given access to $\bar{\mathcal{G}}_{acrs}$. Party corruptions can be adaptive (and in the non-erasure model), as long as they are PID-wise.*

Finally, we note that a GUC secure realization of $\mathcal{F}_{com}$ is indeed sufficient to GUC-realize any "well-formed" *multi-party* functionality. This may be accomplished by first using $\mathcal{F}_{com}$ to realize $\mathcal{F}_{zk}$ (as in [30]), and then using $\mathcal{F}_{zk}$ to realize the "one-to-many" Zero-Knowledge functionality, $\mathcal{F}_{zk}^{1:M}$ (via the technique of [74]). The multi-party protocol compiler from [30] can then be used to yield a UC-secure realization of any well-formed multi-party functionality in the $\mathcal{F}_{zk}^{1:M}$-hybrid model without using any shared state (thus it is also a GUC-secure realization by Corollary 1.1).

4.3.2 GUC-Realizing $\mathcal{F}_{com}$ Using the ACRS Global Setup

We now describe the construction of a protocol satisfying the conditions of Theorem 4.3, above. When combined with the compiler from [30], such a *fully simulatable* realization of $\mathcal{F}_{com}$ yields a fully simulatable realization of any well-formed two-party or multi-party functionality. Further-

[1] In fact, the protocol we will describe in Section 4.3.2 can also support a "graceful failure" approach similar to that outlined in [6], in the scenario where protocol participants do not mutually trust any single authority. That is, by using suitable "graceful" tools (in the case of our protocol, a "graceful" IBTC) , we can ensure full GUC security if trustworthy authorities are used by all parties, and ordinary stand-alone security for party P in the case where only party P's authority is trustworthy (even if party P's own authority is made completely unavailable after publishing its reference string and/or is later corrupted subsequent to the completion of the protocol!). The proof of this claim is a straightforward consequence of our protocol in Section 4.3.2, combined with the fact that our IBTC construction in Section 2.4 guarantees hiding even for maliciously chosen public keys (as long as it remains possible to verify the existence of the Σ-protocol with HVZK property for the signature scheme). Of course, all IBTCs addressed to party P should make use of the reference string published by the authority that P trusts.

more, we show that, in addition to requiring only the more minimal $\bar{\mathcal{G}}_{acrs}$ setup, our protocol achieves significantly stronger properties than the fully simulatable protocol from [6] realized in the $\bar{\mathcal{G}}_{krk}$ model. (Of course, our protocol can also be trivially modified for use in the $\bar{\mathcal{G}}_{krk}$ model, where it will enjoy the same strengthened security guarantees.)

Firstly, our protocol realizing $\mathcal{F}_{com}$ in the $\bar{\mathcal{G}}_{acrs}$ model remains secure even in the presence of adaptive corruptions (whereas the protocol of [6] does not). Intuitively, adaptive security seems to be difficult to attain in either the $\bar{\mathcal{G}}_{krk}$ or $\bar{\mathcal{G}}_{acrs}$ models, since an adaptive adversary is eventually able to learn nearly all secrets in the system (save only for the random coins of the trusted party). Since the simulator relies on these same secrets to "break" privacy of (corrupt) parties during attack simulations, it would seem that an adversary provided with access to *all* secrets should be able to violate the privacy of past protocol transcripts (allowing it to discern simulated "information-less" interactions from genuine ones). Our protocol manages to avoid this problem through the use of some additional interactivity.

Remarkably, our protocol for realizing $\mathcal{F}_{com}$ will also maintain security of past protocol executions even when the trusted party implementing $\bar{\mathcal{G}}_{acrs}$ is later corrupted (revealing the random coins used to generate the CRS, and leaving the overall system with no secrets at all)! That is, our protocol will guarantee that past transcripts of protocol interactions can *never* be used to compromise the security or deniability of honest parties *even if the trusted party is later corrupted*. Security is only lost when the trusted party acts maliciously *prior to*, or *during* protocol execution. This kind of "forward security" with respect to the trusted party further minimizes the trust assumptions required to realize $\bar{\mathcal{G}}_{acrs}$ in the real-world. For instance, an adversary cannot later coerce the trusted party into breaking the security of an honest party after the completion of the protocol. Such forward security cannot be achieved using the protocol of [6] since knowledge of the secret key allows "extraction" from past commitments, breaking privacy. Similarly, the protocol of [30] also loses all privacy of past transcripts if the trusted party implementing the CRS setup later reveals a trapdoor.

4.3.3 High-level Description of Protocol UAIBC

Our protocol for realizing $\mathcal{F}_{com}$ in the $\bar{\mathcal{G}}_{acrs}$ shared hybrid model, which we call Protocol UAIBC (for UC Adaptive Identity-Based Commitment), relies on two new techniques. First, we construct an *identity-based* trapdoor commitment (IBTC) which enjoys adaptive security. Then we provide a general transformation from any IBTC into a protocol that securely implements $\mathcal{F}_{com}$.

Constructing IBTC. In the setting of IBTC a single "master-key" is made public. Additionally, all parties can obtain a private-key that is associated to their party identifier. (Note that this setting corresponds exactly to the interface of $\bar{\mathcal{G}}_{acrs}$.) Intuitively, an IBTC is a commitment scheme with the additional property that a committer who *knows* the receiver's secret-key can *equivocate* commitments (*i.e.*, it can open up commitments to any value, breaking the binding property of the commitment). Furthermore, it should hold that an adversary that obtains the secret-keys of multiple parties, still should not be able to violate the binding property of commitments sent to parties for which it has not obtained the secret-key.

Constructions of IBTCs were previously known in the Random Oracle Model [3, 87]. In Section 2.4 we provide a conceptually simple approach to constructing an adaptively secure IBTC from any one-way function, in the standard model. Our technique relies on the use of Σ-protocols [32], in an approach based on that of [46] (and perhaps surprisingly can result in a very practical protocol). On a very high-level (and very oversimplified) the general idea is as follows: 1) let the master-key be a public-key for a signature scheme, 2) let the secret-key for a party be a signature on its party identifier, and 3) construct a commitment scheme where the reveal phase consists of a "proof" that *either* the revealed value is consistent with the value committed to, *or* the committer knows a signature on the receiver's party identifier (this "proof" must also "hide" which of these two statements actually holds). We mention that the actual instantiation of this idea is somewhat more involved, in order to guarantee adaptive security, and we provide the full details of our construction in Section 2.4.

From IBTC to GUC Commitments. Recall that a protocol for realizing $\mathcal{F}_{com}$ must intuitively satisfy two properties (in addition to the traditional binding and hiding properties of any commitment scheme): 1) it must be equivocable, and 2) it must be extractable. We show how to

transform any "equivocable" commitment scheme (such as an IBTC) into a protocol for securely realizing $\mathcal{F}_{com}$ (for single bit commitments). Previously similar types of transformations have appeared in the literature (*e.g.*, [30, 7]). Unfortunately all such transformations either require some additional *non-global* setup (and are thus not applicable in out setting), or only work in the case of static security. We now turn our focus to the protocol UAIBC, which GUC-realizes the $\mathcal{F}_{com}$ functionality via a novel transformation of an IBTC from a mere equivocable commitment (in the standard model), to an equivocable *and* extractable commitment secure against adaptive corruptions in the GUC-security model. We remark that our transformation technique can be employed by substituting *any* merely equivocable commitment scheme (such as standard public key based trapdoor commitments) in place of the IBTC in our protocol, and will yield a scheme that is both equivocable and extractable, a general approach that may prove useful in many other contexts.

We now describe some of the details and intuition for our protocol UAIBC. We will show that UAIBC realizes the $\mathcal{F}_{com}$ ideal functionality in a *fully simulatable* manner (even against adaptive adversaries in the non-erasure setting). Furthermore, the protocol UAIBC is *forward secure*, so that past protocol sessions will remain secure even if the trusted party implementing $\bar{\mathcal{G}}_{acrs}$ is later corrupted. We refer to this latter property as "strong full simulatability", since the output transcript of a GUC-simulation will be indistinguishable from a real transcript even to parties in possession of all keys (indeed, even if the parties possess all the random coins used to generate the CRS!). The protocol is given in the combined secure and authenticated channels model (formally denoted as the $\mathcal{F}_{smt}$ and $\mathcal{F}_{auth}$ hybrid models, respectively). In particular, UC-secure channels are used crucially in the proof of security of UAIBC, detailed in Section 4.3.2.

On a high-level, protocol UAIBC proceeds as follows. The committer S and receiver R first perform a coin-tossing to generate a public-key ρ for a dense crypto-system. This coin-tossing requires the receiver to use an IBTC, and has the property that if the committer is corrupted, the outcome of the coin-tossing can be set to any value. After a completed coin-tossing, the committer commits to a single bit b using an IBTC (let c denote this commitment), and additionally sends an auxiliary string e: e is either a random string in case $b = 1$, and an encryption to the decommitment information of c if $b = 0$. (We here require that the encryption scheme used has *pseudo-random ciphertexts*.) In the reveal phase, the committer is required to provide correct

decommitment information for c, and additionally reveal the value encrypted in e in case $b = 0$. We graphically illustrate the operation of this protocol in Figure 4.2.

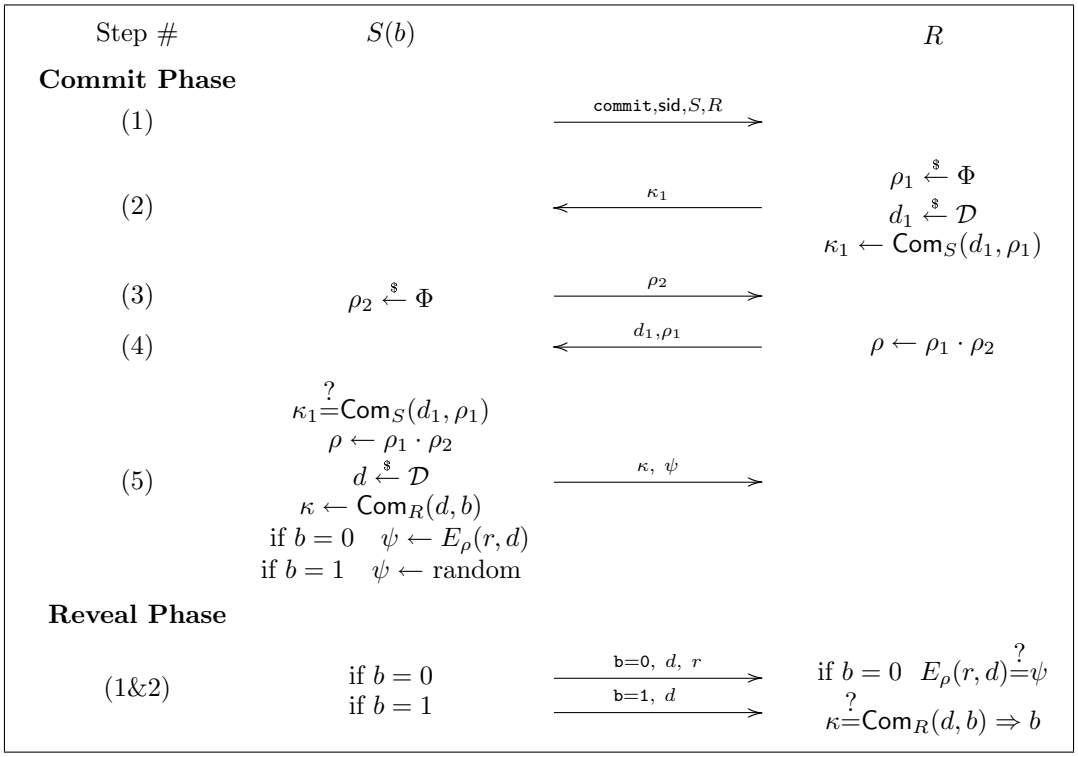

Figure 4.2: A graphical illustration of Protocol UAIBC for realizing $\mathcal{F}_{com}$, with S committing bit b to R. Note that Com is the commitment operation of an IBTC (the subscript is the identity of the intended recipient), and E_ρ is a Dense PRC secure encryption using public key ρ (the first input is the random coins fed to the encryption operation, and the second is the plaintext). Steps 2 to 4 of the **Commit** phase are essentially a coin-tossing protocol, whereas the subsequent steps are similar to the protocol of [30].

We now take a brief detour to specify the security properties of the IBTC scheme and the dense crypto-system required by our protocol, before proceeding with more in-depth discussion of the protocol.

4.3.4 Details and Design of Protocol UAIBC

A full description of the protocol UAIBC in the $\bar{\mathcal{G}}_{acrs}$ (Augmented CRS) model is shown in Figure 4.3 below.

Protocol UAIBC

Let (G, E, D) be a Dense PRC secure encryption scheme. We denote the key-space of the dense encryption scheme by Φ, and the security parameter by λ. Then Protocol UAIBC proceeds as follows, with party S committing a bit b to party R, in the shared $\bar{\mathcal{G}}_{acrs}$ hybrid model. (Note, we also assume ideal private and authenticated channels for all communications.)

Common Reference String: All parties are initialized with a CRS produced by $\bar{\mathcal{G}}_{acrs}$ (a public key for an IBTC scheme from Setup, used as an implicit parameter to Com). Note that $\bar{\mathcal{G}}_{acrs}$ is a shared subroutine of Protocol UAIBC.

Commit Phase:

1. S sends $(\mathtt{com}_0, \mathsf{sid}, S, R)$ to R.
2. R samples $\rho_1 \xleftarrow{\$} \Phi$ and a decommitment $d_1 \xleftarrow{\$} \mathcal{D}$, and records them. R then computes $\kappa_1 \leftarrow \mathsf{Com}_S(d_1, \rho_1)$, and sends $(\mathtt{com}_1, \mathsf{sid}, S, R, \kappa_1)$ to S.
3. S samples $\rho_2 \xleftarrow{\$} \Phi$, and sends the message $(\mathtt{com}_3, \mathsf{sid}, S, R, \rho_2)$ to R.
4. R computes and stores $\rho \leftarrow \rho_1 \cdot \rho_2$, and sends $(\mathtt{com}_3, \mathsf{sid}, S, R, d_1, \rho_1)$ to S.
5. S verifies that $\kappa_1 = \mathsf{Com}_S(d_1, \rho_1)$, and ignores the message if verification fails. If verification succeeds, S computes $\rho \leftarrow \rho_1 \cdot \rho_2$, then chooses $d \leftarrow \mathcal{D}$ and records it along with ρ. Additionally, if $b = 0$, S chooses and records random coins r, and then computes $\psi \leftarrow E_\rho(r, d)$; otherwise, when $b = 1$, S chooses ψ at random. S then sends $(\mathtt{com}_4, \mathsf{sid}, S, R, \kappa, \psi)$ to R.
6. R receives $(\mathtt{com}_4, \mathsf{sid}, S, R, \kappa, \psi)$ from S and records it. R then outputs $(\mathtt{receipt}, \mathsf{sid}, S, R)$.

Reveal Phase:

1. S retrieves the necessary records and, if $b = 0$, sends $(\mathtt{rev}, \mathsf{sid}, S, R, d, 0, r)$ to R; otherwise, S sends $(\mathtt{rev}, \mathsf{sid}, S, R, d, 1)$ to R.

2a. When receiving a message of the form $(\mathtt{rev}, \mathsf{sid}, S, R, d, 0, r)$, R retrieves the values of ρ, κ, and ψ corresponding to sid from its records and verifies that $E_\rho(r, d) = \psi$ and $\mathsf{Com}_R(d, 0) = \kappa$. If the verification succeeds, R outputs $(\mathtt{reveal}, \mathsf{sid}, S, R, 0)$. Otherwise, R ignores the message.

2b. When receiving a message of the form $(\mathtt{rev}, \mathsf{sid}, S, R, d, 1)$, R retrieves the value of κ corresponding to sid from its records and verifies that $\mathsf{Com}_R(d, 1) = \kappa$. If the verification succeeds, R outputs $(\mathtt{reveal}, \mathsf{sid}, S, R, 1)$. Otherwise, R ignores the message.

Figure 4.3: Protocol UAIBC for realizing $\mathcal{F}_{com}$ in the $\bar{\mathcal{G}}_{acrs}$ shared hybrid model.

Intuitively, the final flow of the commit phase of the UAIBC protocol is very similar to the UAHC protocol of [30] and the CLOS-KR protocol of [6]. The actual structure of the last flow in UAIBC is slightly different from the UAHC and CLOS-KR protocols since it uses a single ciphertext, rather than two ciphertexts as the UAHC and CLOS-KR protocols do. The only purpose of this minor change is to further simplify the presentation of the protocol and the proof, and indeed it would seem that the same change can be safely made to the UAHC and CLOS-KR protocols.

The novelty of protocol UAIBC is its use of a coin-flipping protocol to choose the public key for the encryption scheme (used when producing ψ), rather than using a globally fixed key as in [30], or public keys belonging to the individual parties as in [6]. The net result of this coin-flipping protocol is that no party (trusted or otherwise) should ever be in possession of the decryption key, unless perhaps some "cheating" occurred during the coin-flipping process (as we will discuss below).

Of course, the use of this coin-flipping protocol makes the commitment phase of the protocol interactive (whereas in [30] it was not). We remark that some interactivity is unavoidable when constructing adaptively secure GUC commitment schemes in the Augmented CRS model (see the lower bound of Section 5.1).

Simulation of the UAIBC protocol is slightly involved. On a high level, as with all UC-secure realizations of $\mathcal{F}_{com}$, the simulator must be able to extract the actual committed values being sent by corrupt parties during the commit phase, while simultaneously being capable of equivocating commitments that are sent by honest parties (for which the simulator must generate the protocol flows). On the surface, these properties seem to conflict – if a specific value can be extracted from a flow of the commitment protocol, then the flow must be binding, and yet, the simulator must also be able to equivocate flows, so they are not always binding. This conflict is particularly striking in the Augmented CRS model, where the adversary is allowed to learn all the same trapdoor information as the simulator. When no trapdoor information is available exclusively to the simulator, it is difficult to imagine how commitments can be binding for the adversary, but remain equivocable for the simulator. In other words, it would seem that if the simulator has the power to extract committed values and to equivocate commitments, an equally powerful real-world adversary should also have these capabilities. Indeed, that difficulty is precisely what

causes the UAHC protocol to fail in our setting. Fortunately, there is one crucial advantage that the simulator is allowed to have that enables us to overcome these conflicts: the simulator is allowed to choose the random coins used to generate the protocol flows of honest parties, whereas the real-world adversary has no such control (although it may learn the random coins upon later corruption of the honest party). The coin-flipping technique is a method for leveraging this advantage.

The coin-flipping protocol executed in Steps 1-4 of the UAIBC commitment protocol is designed so that the simulator can program the outcome of the coin-flip whenever party S is corrupt. By using the S's IBTC key, SK_S, to equivocate the commitment sent in Step 2, the simulator can arrange the public key ρ to be any string of its choosing. Of course, the simulator will choose a public key ρ for which it knows the corresponding secret key, ρ^{-1}. Thus, if party S is corrupt, the simulator is able to decrypt the ciphertext sent in Step 5, extracting the committed value.

To equivocate commitments sent to a corrupt party R, the simulator uses the IBTC key belonging to R, SK_R, in order to generate an equivocable commitment to be sent in the final flow (Step 5). The ciphertext sent contains a legitimate opening to 0, but will later be passed off as a random ciphertext in the event that the simulator must open to 1. While such an equivocation could be detected by someone with knowledge of ρ^{-1}, in this instance, no one (not even the simulator) can have such knowledge, as the coin-flipping process will be fair.

Proving that the simulation described above is indeed indistinguishable from the real-world interaction requires the use of a rewinding technique. We stress that the UC simulation itself does not employ any rewinding (which is of critical importance, since a UC simulator may not rewind the environment). However, the use of a rewinding technique is required whenever we must "program" the coin-flipping results in the case where the adversary corrupts party R. In particular, this scenario is encountered only in the reduction to the security of the encryption scheme used for one of the simulation indistinguishability hybrid arguments. In order to show that the encryption can be broken if the hybrid experiments can be distinguished, we must arrange for the encryption key being attacked by the reduction to appear in the outcome of the coin-flipping (even when R is corrupted).

In summary, protocol UAIBC introduces a technique for transforming trapdoor commitments (which are merely equivocable) to full-fledged (adaptive *and* forward secure!) UC commitments

(which must be both equivocable and extractable) by using a simulatable coin-flipping protocol. In particular, the simulatable coin-flipping protocol makes use of both rewinding and trapdoors to achieve simulatability. (We employ only the trapdoor based simulation technique when constructing the UC simulator itself, while relying on the ability to rewind the coin-flipping protocol *only for the proof* of the simulation's indistinguishability.) Our general approach to constructing adaptively UC secure commitments from standard trapdoor commitments is of independent interest, and indeed, the problem was implicit in many prior works (*e.g.* [30, 23, 7]) which either resort to impractical techniques (such as per-session CRS) or are forced to settle for static (non-adaptive) security.

4.3.5 Security Proof for Protocol UAIBC

In this section, we will prove that protocol UAIBC is a GUC-secure realization of $\mathcal{F}_{com}$ in the $\bar{\mathcal{G}}_{acrs}$ shared hybrid model. Before proceeding with the details of the proof, we first state and prove a general lemma regarding coin-tossing protocols that is used in our proof of security. Our lemma is analogous to the Reset Lemma of [10], but unlike the Reset Lemma it aims to guarantee not only that certain events will occur after rewindings, but that the output distribution of the entire experiment remains computationally indistinguishable.

Lemma 4.1 (Coin-tossing Lemma)**.** Consider a general "indistinguishability" task, X. The task is represented as a PPT algorithm that may take as input a system parameter Λ, a random reference parameter ρ, a bit string x, and challenge bit b. The system parameter Λ is generated by some designated PPT parameter generation algorithm. The reference parameter ρ must be drawn from an abelian group Φ (which possibly depends on the choice of Λ) with efficiently computable inverse and group operations (which we write multiplicatively). The uniform distribution on Φ must be efficiently samplable, and membership in Φ should be efficiently decidable. The task X is said to be computationally indistinguishable if every PPT distinguisher D has at most negligible advantage in the following attack game:

1. The challenger first generates a system parameter Λ (in accordance with the parameter generation algorithm). The challenger then samples $\rho \xleftarrow{\$} \Phi$, and sends the pair (Λ, ρ) to D.
2. D computes an arbitrary value x, and sends x to the challenger.
3. The challenger chooses $b \xleftarrow{\$} \{0,1\}$, computes $y \leftarrow X(\Lambda, \rho, x, b)$, and sends y to D.
4. D outputs a guess $\hat{b} \in \{0,1\}$.
5. D's advantage is defined to be $|\Pr[b = \hat{b}] - 1/2|$.

Attack Game 9: Task Indistinguishability Attack Game

Suppose that, instead of selecting the reference parameter at random from Φ, we generate the reference parameter using a simple "coin tossing" protocol that takes place between the challenger and the distinguisher. The coin tossing makes use of a string commitment Com that is computationally binding with respect to the distinguisher (this security property is formally described in Section 2.4). For the sake of generality, we allow both Com and Φ to be parameterized by Λ. We claim that the computational indistinguishability of X holds, even though ρ is now being chosen with some participation by the distinguisher itself.

More formally, for any computationally indistinguishable task X, the advantage of any PPT distinguisher D' is at most negligible in the following attack game:

1. The challenger first generates a system parameter Λ (in accordance with the parameter generation algorithm). The challenger then sends Λ to D'.
2. D' sends a commitment κ_1 to the challenger.
3. The challenger samples $\rho_2 \xleftarrow{\$} \Phi$, and sends ρ_2 to D'.
4. D' computes an arbitrary value x, along with an opening (d_1, ρ_1), and sends the triple (d_1, ρ_1, x) to the challenger.
5. The challenger verifies that $\kappa_1 = \mathsf{Com}(d_1, \rho_1)$, and that $\rho_1 \in \Phi$. The challenger then computes $\rho \xleftarrow{\$} \rho_1 \cdot \rho_2$. Finally, the challenger chooses $b \xleftarrow{\$} \{0,1\}$, computes $y \leftarrow X(\Lambda, \rho, x, b)$, and sends y to D'. If the verification step fails, the challenger sends the error message $\perp$ to D' instead.
6. D' outputs a guess $\hat{b} \in \{0,1\}$.
7. D''s advantage is defined to be $|\Pr[b = \hat{b}] - 1/2|$.

Attack Game 10: Coin-tossing Attack Game (for general tasks)

To summarize, subject to some minor and natural technical qualifications, the Coin-tossing Lemma states that: *Experiments that are computationally indistinguishable when fed with a randomly sampled reference parameter will remain indistinguishable when the reference parameter is substituted by the output of a (three round) two-party coin-tossing protocol (where the second flow of the protocol is assumed to be generated honestly).*

Proof. Suppose that D' is an efficient distinguisher for the game described in Attack Game 10 that succeeds with a non-negligible advantage α. In particular, we have that $\alpha \geq 1/p$ for some polynomial p (at infinitely many values of the security parameter). We show how to construct an efficient distinguisher D that succeeds with non-negligible advantage in Attack Game 9, contradicting the computational indistinguishability of the task. The distinguisher D operates as follows:

1. D runs a copy of D' internally, acting as the challenger for D', and begins by passing along

the value of Λ it receives in Step 1 of Attack Game 9. The value ρ that D also receives in Step 1 is recorded for later use.

2. After receiving a commitment κ_1 from the internal copy of D', D samples $\rho_2^* \xleftarrow{\$} \Phi$ and sends it to D'.

3. If D' does not respond with a valid opening of κ_1, then D outputs 0 and halts. Otherwise, if D' responds with a value x^* and a valid opening of κ_1 to the value ρ_1, then D proceeds as follows:

   $\omega \xleftarrow{\$} \{1, \ldots, p\}$
   for $i \leftarrow 1$ to p do
      if $i = \omega$ // *Pray ω is the "right" guess*
         then $\rho_2 \leftarrow \rho \cdot \rho_1^{-1}$
         else $\rho_2 \xleftarrow{\$} \Phi$
      rewind D' back to Step 3 of Attack Game 10 and send ρ_2 to D'
      if D' responds with x and a valid opening of κ_1 to the value ρ_1', then
         if $i \neq \omega$ then output 0 and halt // *Prayers unanswered*
         if $\rho_1' \neq \rho_1$ then output 0 and halt // *Commitment broken!*
         D passes x to its challenger in Step 2 of Attack Game 9
         D receives a response y from its challenger, and forwards y to D'
         when D' outputs $\hat{b}$, D outputs $\hat{b}$ and halts
   output 0 and halt // *rewinding failed to produce second opening*

We claim that this D will have advantage at least $1/4p^2$ at game Attack Game 9. To see this, will consider a series of modifications to Attack Game 10.

Game G_0. This game is modified version of Attack Game 10 that plays out identically unless the verification in Step 5 succeeds. In this case, instead of simply sending the value y, the challenger will rewind D' to Step 3 and feed it with another (freshly sampled) value of ρ_2, as many times as it takes to get a second run with a successful opening of the commitment. That is, after rewinding, the game proceeds normally to Step 5, only now if the verification

fails, we will rewind D' to Step 2 again and repeat until the verification in Step 5 has succeeded for a second time, after which the game continues as in the original version.

What is the advantage of D' in G_0? If the advantage of D' was α in Attack Game 10, then the advantage of D' at G_0 is also α. To see why, let C and C' be random variables representing the value of ρ_2 chosen on the first and last runs (respectively), and let R be the random variable representing all the random coins used in Step 1 and the random tape of D' (note that the value of R is unaffected by any number of rewindings). Let $\mathsf{Succ}(r, \rho_2)$ be a predicate denoting whether the verification in Step 5 succeeds when $R = r$ and $C = \rho_2$ (*i.e.*, it indicates whether D' gives a valid opening of the commitment with those outcomes). Together, R and C represent all the random coins used in Attack Game 10. We will show that, for any fixed choice of R, the distribution of C' is the same as the distribution of C, and therefore distribution of the modified experiment is unchanged from that of the original. Fixing our choice of R and expanding the distribution of C' by total probability theorem yields:

$$\begin{aligned}
\Pr[C' = \rho_2 \mid R = r] \quad = \quad & \Pr[C' = \rho_2 \mid R = r, \mathsf{Succ}(r, C)] \cdot \Pr[\mathsf{Succ}(r, C)] \\
& + \Pr[C' = \rho_2 \mid R = r, \neg\mathsf{Succ}(r, C)] \cdot \Pr[\neg\mathsf{Succ}(r, C)] \\
= \quad & \Pr[C = \rho_2 \mid R = r, \mathsf{Succ}(r, C)] \cdot \Pr[\mathsf{Succ}(r, C)] \\
& + \Pr[C = \rho_2 \mid R = r, \neg\mathsf{Succ}(r, C)] \cdot \Pr[\neg\mathsf{Succ}(r, C)]
\end{aligned}$$

The equality of the first terms in the sums follows from simply observing that $C' = C$ in the event that D' fails to open the commitment on the first run, since no rewinding occurs in that case (the first run is the last run). The equality of the second terms follows by observing that, conditioned on $\mathsf{Succ}(r, C)$, the variable C' is uniformly distributed among the values of ρ_2 such that $\mathsf{Succ}(r, \rho_2)$ is true, since in this case we will "rewind" by feeding uniformly random choices of ρ_2 into the experiment until $\mathsf{Succ}(r, \rho_2)$ holds. (Of course, C is also uniformly distributed among those same values when conditioned on $\mathsf{Succ}(r, C)$.) Applying total probability theorem again to the last line in the equation above, we obtain: $\Pr[C' = \rho_2 \mid R = r] = \Pr[C = \rho_2 \mid R = r]$. Thus, the distributions of the pair (R, C)

and the pair (R, C') are identical. We conclude that the output distribution of the original experiment in Attack Game 10 and the output distribution in Game G_0 are the same.

Game G_1. This is a minor modification of Game G_0. In the event that rewindings occur, G_1 is identical to G_0 for the first p rewindings. After p rewindings have occurred, G_1 proceeds from Step 3 as in the original version of Attack Game 10, without making any further attempts to rewind. G_1 is otherwise identical to G_0.

To analyze what happens to G_0 if we bound the number of rewindings by p, we first calculate the expected number of rewindings in G_0. Let N denote the number of rewindings that occur in a run of G_0. Using our previously established notation, we will analyze the $\mathrm{E}[N]$ by first fixing the choice of R. We can then compute the expected number of rewindings $\mathrm{E}[N \mid R = r]$ with a simple application of total probability theorem (combined with the linearity property of expectation):

$$\begin{aligned}
\mathrm{E}[N \mid R = r] &= \mathrm{E}[N \mid R = r, \mathsf{Succ}(r, C)] \cdot \Pr[\mathsf{Succ}(r, C)] \\
&\quad + \mathrm{E}[N \mid R = r, \neg\mathsf{Succ}(r, C)] \cdot \Pr[\neg\mathsf{Succ}(r, C)] \\
&= (1/\Pr[\mathsf{Succ}(r, C)]) \cdot \Pr[\mathsf{Succ}(r, C)] \\
&\quad + 0 \cdot \Pr[\neg\mathsf{Succ}(r, C)] \\
&= 1 \qquad (\text{ for } \Pr[\mathsf{Succ}(r, C)] \neq 0 \text{ })
\end{aligned}$$

The second equality follows by observing that, for a fixed value of R, the probability that any of the rewindings succeeds is exactly the same as $\Pr[\mathsf{Succ}(r, C)]$ (the initial probability with which a rewinding occurs). Of course, conditioned on $\neg\mathsf{Succ}(r, C)$, there are no rewindings at all. In case $\Pr[\mathsf{Succ}(r, C)] = 0$, we will have that $\mathrm{E}[N \mid R = r] = 0$, since no rewindings can occur. For any fixed choice of r we have shown that $\mathrm{E}[N \mid R = r] \leq 1$, and therefore, we may conclude that $\mathrm{E}[N] \leq 1$.

Now we are in a position to analyze the advantage of D' in G_1. Since the expected number of rewinding attempts in G_0 is at most 1, the overall probability that there will be more than p rewinding attempts is at most $1/p$, by Markov's inequality. Therefore, with all but

probability $1/p$, the rewinding procedure of G_1 perfectly simulates G_0, wherein D' has the same advantage α as in the original setting of Attack Game 10. In the event that G_1 differs from G_0, the distinguisher D' learns no information about b, so the probability that $\hat{b} = b$ is $1/2$ (by independence). Therefore, the overall advantage of D' in Game G_1 is at least $\alpha - (1/2)(1/p) \geq 1/p - 1/2p = 1/2p$.

Game G_2. This is the same as Game G_1, except that whenever a second opening of the commitment is obtained, it is checked against the original opening. If the commitment has been opened in two different ways, the game is halted. Since opening the commitment in two different ways must occur with negligible probability (by the binding property of the commitment scheme), for sufficiently large choices of the security parameter, this abort happens with probability at most $1/4p$. Therefore, the advantage of D' in G_2 differs from that of G_1 by at most $1/4p$, so we have that D' has advantage at least $1/2p - 1/4p = 1/4p$ in Game G_2.

Game G_3. Finally, we modify Game G_2 to exactly match the procedure we described above for constructed D from D'. Namely, G_3 differs from G_2 in that if we must rewind, we first select a uniformly random "guess" at one of the p possible rewindings, and if the "guess" does not correspond to the rewinding on which the second opening of the commitment occurs, the game is halted (and the advantage is computed as if D' had output 0). Since the "guess" for the rewinding instance is accurate with probability $1/p$, the experiment plays out as in Game G_2 with probability at least $1/p$. In the event that a halt occurs, we will have that $\hat{b} = b$ with probability $1/2$ (again, by independence). Therefore, the overall advantage of D' in Game G_3 will be at least $(1/p)(1/4p) = 1/4p^2$.

This concludes our analysis of the advantage of D in our construction, showing that D indeed has a non-negligible advantage if distinguisher D' can succeed at Attack Game 10 with non-negligible advantage. □

Proof of Theorem 4.3. We now prove the security of protocol UAIBC, by showing that it is a GUC-secure realization of $\mathcal{F}_{com}$. We describe a technique for simulating the protocol flows of UAIBC in the ideal-world with $\mathcal{F}_{com}$, and give a proof that the simulation in the ideal-world

setting is indistinguishable from a real-world execution of UAIBC, even if the adversary is allowed to corrupt the trusted party implementing $\bar{\mathcal{G}}_{acrs}$ at the end of the game. Recall that we need only prove that UAIBC will $\bar{\mathcal{G}}_{acrs}$-EUC-emulate $\mathcal{F}_{com}$, and thus we will be proving with respect to a $\bar{\mathcal{G}}_{acrs}$-externally constrained environment $\mathcal{Z}$. To model this, we may imagine that $\mathcal{Z}$ is provided with SK_{P_a}, for any corrupt party P_a, at the moment of corruption. (In reductions to the security of the IBTC, we make an oracle query to obtain the key SK_{P_a}, which, as can be seen below, will never correspond to the key under attack by the reduction.)

Initialization - All parties are initialized with a copy of the common reference string PK published by $\bar{\mathcal{G}}_{acrs}^{\mathsf{Setup},\mathsf{Extract}}$ during its honestly executed global initialization step. That is, the simulation we describe below merely expects the initialization step to be completed prior to the start of the simulation (no special setup is required on behalf of the simulator itself).

Simulating Communication with $\mathcal{Z}$ - $\mathcal{S}$ simply forwards communications between $\mathcal{A}$ and $\mathcal{Z}$.

Simulating the commit phase for honest parties (Strategy 1) - Since we are in the secure channels model, the simulator need only send null messages of appropriate length for each of the messages passed in UAIBC. Nothing more need be done unless one of the parties is corrupted at some later time. If the sender is later corrupted, the simulator receives the sender's input and can construct internal state for the sender representing an honestly generated transcript. If the recipient is later corrupted (prior to the reveal phase, which would again provide the simulator with the correct input) then the simulator must follow the strategy we now describe for the case of a recipient corrupted prior to the sender.

Simulating the commit and reveal phases for corrupt recipients (Strategy 2) - If the recipient is corrupted at any point during the commit phase while the sender is still honest, the simulator conducts the following strategy. We note that, once again, we depend critically on the secure channels model to allow the simulator to choose the content of the prior transcript only *after* corrupting the recipient in the ideal model. Since only the recipient has been corrupted, the simulator must be able to complete the commit phase of

the protocol on behalf of the sender without actually knowing the committed bit b (which $\mathcal{F}_{com}$ will not disclose until the reveal phase is initiated). Thus, the simulator needs to be able equivocate the commitment so that it can be opened to the proper value b in the subsequent reveal phase. Before generating any of the protocol flows, $\mathcal{S}$ issues a `retrieve` request to $\bar{\mathcal{G}}_{acrs}^{\mathsf{Setup},\mathsf{Extract}}$ on behalf of the corrupt R in order to obtain SK_R. All flows will be honestly generated except the final message sent by S in Step 5 of the commit phase of the protocol. $\mathcal{S}$ computes $(\hat{\kappa}, \alpha) \leftarrow \mathsf{ECom}(R; SK_R)$ and $d^{(\hat{b})} \leftarrow \mathsf{Eqv}(R, SK_R, \hat{\kappa}, \alpha, \hat{b})$ for $\hat{b} = 0, 1$ and stores the results. Instead of the honest flow, $\mathcal{S}$ sends $(\mathtt{com}_4, sid, S, R, \hat{\kappa}, \hat{\psi})$ where $\hat{e}$ is the encryption of $d^{(0)}$ (computed exactly as though committing to $b = 0$ in Step 5).

When performing the reveal phase, $\mathcal{S}$ can simply open to $b = 0$ using the decommitment $d^{(0)}$ honestly (including opening $\hat{\psi}$ correctly). If $\mathcal{S}$ must open to $b = 1$ it can use the decommitment $d^{(1)}$, and by the pseudo-random property of the encryption scheme, it can "pretend" that $\hat{\psi}$ was chosen at random.

Simulating the commit and reveal phases for corrupt senders (Strategy 3) - If the sender is corrupted while the recipient is honest during the commit phase, the simulator will need to learn the committed bit b from the sender in order to correctly provide the corresponding commit phase input to $\mathcal{F}_{com}$. Once again, we note that the simulator need not generate any actual protocol flows until the actual time of corruption, so the following strategy may be employed irrespective of the particular step in the protocol of the commit phase at which S is corrupted. Upon the initial corruption of S, $\mathcal{S}$ will issue a `retrieve` request to $\bar{\mathcal{G}}_{acrs}^{\mathsf{Setup},\mathsf{Extract}}$ on S's behalf in order to retrieve SK_S. The simulation strategy will replace the flows sent by R in Steps 2 and 4 with specially "rigged" flows. $\mathcal{S}$ first generates and stores a valid key-pair for the dense encryption scheme $(\overline{\rho}, \overline{\rho^{-1}}) \leftarrow G(1^\lambda)$. For Step 2, $\mathcal{S}$ computes and stores $(\hat{\kappa_1}, \alpha) \leftarrow \mathsf{ECom}(S, SK_S)$, sending $\hat{\kappa_1}$ in place of the honestly computed κ_1 (on behalf of the honest party R). In Step 4, $\mathcal{S}$ computes and stores $\hat{\rho_1} \leftarrow \overline{\rho} \cdot \rho_2^{-1}$ and $\hat{d_1} \leftarrow \mathsf{Eqv}(S, SK_S, \hat{\rho_1})$. $\mathcal{S}$ replaces the legitimate decommitment d_1 that would normally be sent in Step 4 with $\hat{d_1}$, and the result is that $\kappa_1 = \mathsf{Com}_S(\hat{d_1}, \hat{\rho_1})$. This procedure ensures that $\mathcal{S}$ knows the decryption key for the ciphertext ψ sent by S in Step 5, and can thus decrypt ψ if $b = 0$ and it was legitimately formed. If $\mathcal{S}$ is able to decrypt ψ

to obtain a valid d such that $\kappa = \mathsf{Com}_R(d, 0)$, then $\mathcal{S}$ proceeds by invoking $\mathcal{F}_{com}$ as if S had committed to 0, otherwise it invokes it with a commitment to 1 (since S cannot possibly produce a valid opening of the commitment to 0 in this case).

Notice that in this scenario, the simulator will not need to send any message during the reveal phase, since the sender is corrupted and is the only party sending messages during a reveal operation. The simulation will only fail (and halt) if a valid reveal phase message is sent from S (who is corrupt, and controlled by $\mathcal{A}$) which opens to a different bit b than the simulator chose when invoking $\mathcal{F}_{com}$ at the end of the commit phase.

Message delivery - $\mathcal{S}$ will merely deliver messages from $\mathcal{F}_{com}$ whenever $\mathcal{A}$ delivers the corresponding protocol messages in the simulated UAIBC interaction, in the obvious manner.

Given the above simulation strategies, we must prove that the environment $\mathcal{Z}$'s interactions with $\mathcal{S}$ will be indistinguishable from $\mathcal{Z}$'s interactions with $\mathcal{A}$ in the real-world, even if $\mathcal{A}$ is allowed to corrupt the trusted party $\mathsf{T}^{\bar{\mathcal{G}}_{acrs}}$ at the end of the experiment. To show this, we consider the following hybrid experiments.

I_0 **- Real-world interaction.** Protocol UAIBC running in the "real-world" (in this case, the $\bar{\mathcal{G}}_{acrs}$-Relativized setting), interacting with the real-world adversary $\mathcal{A}$.

I_1 **- Employing the commitment part of Strategy 2.** The interaction proceeds exactly as in the real-world, except that wherever Step 5 of the protocol is executed with a corrupt receiver R, the honest sender S computes $(\hat{\kappa}, \alpha) \leftarrow \mathsf{ECom}(R, SK_R)$ (for now, we will simply imagine that that S is somehow provided with SK_R). S then computes the decommitment as $d \leftarrow \mathsf{Eqv}(R, SK_R, \hat{\kappa}, \alpha, b)$. The remainder of the protocol is then conducted as usual. In the event that S is later corrupted, its internal state is constructed to be consistent with a random selection process for the same value of d.

Lemma 4.2. *I_1 is computationally indistinguishable from I_0.*

Proof. The proof is by reduction to the equivocability property of $IBTC$s, since the only difference from I_0 is the use of an equivocable commitment. We observe that if there is an adversary $\mathcal{A}$ such that $\mathcal{Z}$ can distinguish interactions in I_0 from I_1, there is a (simple)

reduction which breaks the equivocability property of the $IBTC$. This indistinguishability holds even for adversaries in the strong fully simulatable setting, since the reduction is given MSK (containing the random coins used by Setup) as part of its input. Thus, it is easy for the reduction to properly simulate even the corruption of the trusted party. We omit the trivial details of the reduction. □

I_2 **- Employing only Strategy 2.** This interaction is the same as I_1, only S will perform Step 5 completely as described in Strategy 1. That is, in addition to the change made in I_1, the only difference between I_2 and I_1 is the computation of the value of ψ. In the event that the honest party commits a bit $b = 1$, the value of ψ now contains $E_\rho(r, d^{(0)})$, whereas it would have been random in I_1.

Lemma 4.3. *I_2 is computationally indistinguishable from I_1.*

Proof. The proof is by reduction to the pseudo-random property of ciphertexts from E. Unfortunately, it is not obvious how to build a reduction attacking the security of a particular public key ρ^*, as we must do in order to achieve the reduction. In fact, we must use a "rewinding" technique in order to accomplish this. While it is critical that UC secure protocols be straight-line *simulatable* (*i.e.*, they may not use rewinding techniques for simulation purposes, see [66] for details), we stress that here we are using the rewinding only in the proof of indistinguishability for the simulator. The simulator itself does not perform any rewinding.

Therefore, our reduction makes use of Lemma 4.1 (the Coin-tossing Lemma) to abstract away the complicated analysis of the rewinding procedure. Observe that the first three flows of UAIBC are a simple coin-tossing protocol, satisfying all the conditions of Lemma 4.1, provided that the GUC experiment we are running does not require access to S's trapdoor (which is relevant for the binding property of the commitment used in the coin-tossing phase) until after the coin-tossing has completed. Since S is an honest party in this scenario, we can be sure that the neither $\mathcal{A}$ nor $\mathcal{Z}$ (nor, of course, the simulator itself) will expect access to S's trapdoor, and the conditions of the Coin-tossing Lemma are thus satisfied. Accordingly, we may substitute the coin-flipped public key ρ with any randomly generated

public key (as if the coin-flipping phase of the protocol had never happened, in either the real or ideal/simulated views). In particular, we may substitute a challenge key ρ^* that our reduction to the security of E is attempting to break. If the resulting versions of I_2 and I_1 are indistinguishable (when using such a random challenge key ρ^*), then we know that the unmodified versions of I_2 and I_1 (where ρ is the outcome of the coin-flipping protocol) are also indistinguishable.

In fact, it is easy to see that I_2 and I_1 must indeed be indistinguishable if the public key ρ^* (challenged to the reduction) is used instead of the coin-flipping protocol output – after all, the reduction to the security of E is now a direct one: the only change between I_2 and I_1 is the substitution of a ciphertext encrypted under ρ^* for a random string. Indistinguishability of this substitution is precisely the guarantee provided by Dense PRC encryption. Given that I_2 and I_1 are indistinguishable when the parties are fed with a random key ρ^*, we may apply the Coin-tossing Lemma to obtain indistinguishability of the full-fledged versions of I_2 and I_1 (which use the coin-tossing protocol).

Once again, we observe that providing the adversary with MSK at the end of the game does not help the adversary to distinguish ciphertexts, since it is too late to use it to equivocate any of the commitments, and thus the adversary cannot "rig" any public keys. □

I_3 **- Adding Strategy 3.** This interaction is the same as I_2 only now R also employs Strategy 2. We first note that it is not a problem that the simulator is revealing commitments to $\hat{\kappa_1}$ instead of κ_1 since the distributions are indistinguishable by the "dense" public key property of the encryption scheme. The only remaining difference is the use of an equivocable commitment, and thus we prove the indistinguishability of I_3 and I_2 using the same technique as Lemma 4.2. Note that, since the indistinguishability of equivocations holds even against distinguishers that know MSK, the simulation remains indistinguishable even if the trusted party is corrupted after the protocol.

I_4 **- Adding Strategy 1.** This interaction is the same as I_3, only we now employ Strategy 1 as well. This is, in fact, just a conceptual change to I_3, since Strategy 1 merely delays the actual computation of protocol flows until they are observable by $\mathcal{A}$. Indistinguishability

follows immediately in the secure channels model.

I_5 **- Simulated interaction.** This is the same as I_5 unless the simulation fails and halts, which can only occur if a reveal phase successfully opens a commit phase to for the bit $b' \neq b$, where b is the bit the simulation invoked $\mathcal{F}_{com}$ with after the commit phase. This can occur in one of two ways. 1) The simulator was unable to decrypt ψ to yield a valid d in Step 5, yet the reveal phase shows that ψ contains a correct encryption of a valid d. This would violate the correctness property of the encryption scheme, so this case never occurs. 2) The simulator decrypted ψ, yielding a valid d such that $\kappa = \mathsf{Com}_R(d, 0)$, but the reveal phase showed a decommitment d' such that $\kappa = \mathsf{Com}_R(d', 1)$. This breaks the binding property of the $IBTC$, and a reduction to this property can be built in the obvious way. We observe that the binding property is broken at "runtime", so the reduction will *not* need to deal with corruption of the trusted party in the "strong fully simulatable setting" (where we achieve forward security with respect to the corruption of the trusted party), which is critical since the reduction will not know MSK. Thus, I_5 must be indistinguishable from I_4, even if the trusted party is corrupted after completion of the protocol.

□

5 • Efficient Protocols for Commitments and Zero-Knowledge Proofs

As we have already argued, from the security and functionality perspectives, the GUC secure protocols of Chapter 4 realized in the ACRS model appear to be strictly superior to UC secure protocols realized in the CRS model of [30]. This raises the natural question of what price must be paid for such gains, particularly in terms of efficiency. Unfortunately, the constructions in Chapter 4 are very inefficient: the commitment scheme committed to the message in a bit-by-bit manner, while the zero-knowledge (ZK) proof for any NP-relation R was implemented using the generic Cook-Levin reduction to a canonical NP-complete problem via the generic technique of [30] for transforming a realization of $\mathcal{F}_{com}$ into a realization of $\mathcal{F}_{zk}$. Thus, now that the feasibility of GUC secure computation has been established, it is natural to ask if one can build *efficient*, GUC-secure commitment and ZK proofs in the ACRS (or PKI) model. In this chapter, we provide such efficient GUC-secure commitment and ZK proofs which are secure against adaptive corruptions, therefore making the ARCS model an attractive alternative to the CRS model on nearly all fronts.

The only drawback of our solution is that we rely on *data erasures*, which is not the case for many efficient UC protocols, such as that of Damgard and Nielsen [35]. In practice, although adaptive security (or at least forward security) is of *critical* concern (particularly in the context of deniability),[1] we believe that the assumption of data erasures is very realistic, especially if the data to be erased is highly ephemeral in nature. In particular, this assumption is widely used in practice (for example, for analyzing most key exchange protocols, such as Diffie-Hellman), and was already used in several works on UC security as well (*e.g.*, [28, 50, 68, 26], although there the use of erasures was hidden deep in the paper). Of course, in light of the feasibility results of

[1] We remark that adaptive security with erasures trivially implies static security, and is usually much harder to achieve than the latter.

Chapter 4 (which have feature adaptive security in the non-erasure model) we may still hope that future research will remove this restriction entirely even from efficient GUC-secure protocols.

In the following sections we present a simple "protocol compiler" transforming any efficient "dense" Ω-protocol proof for an NP-relation R (see Section 2.3.1) into an efficient, constant-round GUC-secure ZK proof (GUC ZK) for R. The notion of Ω-protocols was originally introduced by Garay, MacKenzie and Yang [50]. Briefly, Ω-protocols are simply Σ-protocols with an extra property that one can generate a public reference parameter ρ together with a trapdoor information τ, such that knowledge of τ allows one to extract the witness from any valid conversation between a prover and verifier who use ρ as a reference parameter (as opposed to the usual special soundness property, where one needs two different transcripts with the same first flow). In [50, 68], Ω-protocols were used for the similar task of building UC-secure ZK proofs in the CRS model (which means, of course, that they are not GUC-secure protocols as per the main impossibility result of Chapter 4). As a result of the more stringent requirements of GUC-security, our compiler is *considerably* more involved than the compiler of [50, 68] (which also used erasures). For example, in the GUC setting the simulator is not allowed to know a trapdoor τ for any globally published reference parameter ρ, so we have to sample the reference parameter ρ using the ACRS model with a coin-flipping protocol (in a similar fashion to the approach we used in the protocol of Section 4.3.2). As a result of this need to generate ρ via coin-flipping, our compiler requires Ω-protocols whose reference parameters are "dense" (*i.e.*, indistinguishable from random), and therefore none of the previous Ω-protocols of [50, 68] is suitable for our purpose.

Therefore, of independent interest, we will present several novel constructions of *dense* Ω-protocols. First, we show how to build a direct, but only semi-efficient dense Ω-protocol for any NP relation R from any Σ-protocol for R. Although this Ω-protocol uses the cut-and-choose technique (somewhat similar to the technique of Pass [73], but in a very different setting), it is very general and yields a much more efficient GUC ZK protocol than the technique of [30] (employed in our feasibility result), which uses a generic Cook-Levin reduction. Next, we show a *very efficient*, number-theoretic based dense Ω-protocol for proving the knowledge of discrete log representation. Once again, this Ω-protocol had to use some interesting additional tools to get beyond the "non-dense" Ω-protocol of [50], such as the "projective Paillier encryption" of Cramer and Shoup [33]. As a result, we get a semi-efficient GUC ZK for any R having an efficient

Σ-protocol, and a very efficient GUC ZK for proving the knowledge of discrete log representation.

Furthermore, using the same techniques for constructing efficient ZK, we proceed to build efficient *constant-rate* (and constant-round) GUC-secure commitments in the ACRS model. In spirit, this result is similar to the result of Damgard and Nielsen [35], who constructed the first constant-rate UC-secure commitments in CRS model. However, our security requirements are more stringent, and, in fact, it seems hopeless to adapt the protocol of [35] to the GUC framework. Instead, we observe that GUC commitments can easily be constructed using our techniques for GUC ZK, provided that we can build an efficient Ω-protocol for a particular relation R on *identity-based trapdoor commitments* (IBTCs). See Section 2.4 for further discussion of IBTCs. Intuitively, in this Ω-protocol a prover needs to show that he knows the message being committed in the value κ, with respect to a particular identity. In fact, if one can build an IBTC scheme where the required relation R would simply use a proof of knowledge for a discrete log representation, our previous GUC ZK protocol would complete the job. Unfortunately, the IBTCs we constructed in Section 2.4 are of a much more complicated form. Therefore, we will now build a specialized IBTC scheme (of independent interest) which is based on Water's signature scheme [84]. The resulting IBTC not only has the needed form for its relation R, but is also much simpler and more efficient than prior IBTCs built in the standard model. Combining these results, we finally build the required GUC commitment.

Finally, we outline a technique for reducing the round complexity of our protocols (in fact, for achieve *optimal* round complexity) by using the random oracle (RO) model in conjunction with the GUC framework (see Section 2.2.4). Given that we previously proved in Section 4.2 that no completely public setup model *including the random oracle model* can be used to realize GUC commitment, it is very surprising to find that random oracles are still useful in the GUC framework, even in the presence of the "more powerful" ACRS setup (which *is* sufficient to realize GUC commitment). Indeed, the RO is simply modeled as a shared functionality available both in the real and in the ideal model. As such, the simulator cannot "reprogram" the RO. Even more counter-intuitively, the simulator cannot even "extract" the random oracle queries used by the real-model attacker! This is because, without loss of generality, we may assume that all such queries are made by the environment (which the simulator cannot control), and that the outputs resulting from the oracle queries are sent to the adversary by the environment instead of the

oracle. Correspondingly, the RO model seems very restricted in the GUC framework. However, we still show that one *can* meaningfully use it *in conjunction with* the ACRS model, because we *are* allowed to extract and reprogram the RO in the proof of security. In particular, by applying the Fiat-Shamir heuristic to our GUC ZK protocols, we obtain an efficient, two-message, straight-line extractable and simulatable (in fact, GUC-secure!) ZK proof for any relation R having an efficient dense Ω-protocol (see above for examples of such Ω-protocols). Moreover, in this protocol one only needs to erase some short data during a *local computation* (*i.e.*, no sensitive data needs to be stored while waiting for network traffic).[2] This makes the need for data erasures even more minimal, since the data to be erased is completely ephemeral.

5.1 A Lower Bound for Round Complexity

One important measure of efficiency for cryptographic protocols is round complexity – protocols that require multiple messages to be sent back and forth incur a proportional overhead due to network latency. Clearly, it is very important for practical reasons to design protocols with *constant* round complexity – and, indeed, all the protocols we have previously described already feature this very desirable property. However, it would be even better to construct protocols with *optimal* round complexity. Of course, in the setting of stand-alone security it is clear that the "commit phase" of commitments is nearly always implemented non-interactively (*i.e.*, using a single message sent by the committer S to the recipient R), and we might wonder if is possible to have GUC-secure commitments that are similarly non-interactive. Furthermore, we might even hope to obtain GUC-secure Non-Interactive Zero-Knowledge (NIZK) proofs, which are known to exist in the CRS model with stand-alone security.

Unfortunately, as we will now demonstrate, GUC secure protocols for commitment and zero-knowledge inherently require some extra interactivity (when they are realized using most natural global setups such as a PKI or ACRS). That is, we will show that the "commit phase" of any GUC-secure commitment scheme requires more than one message, and that GUC-secure NIZK proofs are impossible. This lower bound holds only with respect to protocols that have forward

[2] Of course, we can get a less efficient GUC ZK protocol with these properties, which does *not* rely on data erasures at all, by applying the Fiat-Shamir heuristics to the inefficient protocol of Section 4.3.2.

security (where it holds even if we allow erasures). Indeed, we strongly conjecture that there exist GUC-secure non-interactive protocols realizing these tasks in the static corruption model. However, given the comparatively weak security guarantees of the static corruption model, for most applications it seems preferable to employ interactive protocols while retaining adaptive security.

We begin by proving an impossibility result for non-interactive commitment schemes with forward security in the GUC framework (see Section 2.1 for a description of forward security).

Theorem 5.1. *We say that an "oracle" (or Interactive Turing Machine) is* monotonically consistent *if it always returns the same response to party P when queried repeatedly on the same input by party P (even across separate sessions), except that it may choose not to respond to some queries when P is honest (otherwise, consistency holds independently of P's corruption status). Let* Priv *denote any PPT monotonically consistent oracle (whose outputs may depend on the* pid *of the querying party, but not the* sid*).*

There exists no non-interactive (single message), terminating protocol π that GUC-realizes $\mathcal{F}_{com}$ with forward security (even in the erasure model), using only the shared functionality for Priv*. This holds even if the communication is ideally authentic. (In particular, we note that* $\text{Priv} = \bar{\mathcal{G}}_{acrs}$ *and* $\text{Priv} = \bar{\mathcal{G}}_{krk}$ *are efficient monotonically consistent oracles, even if they are combined with the random oracle shared functionality $\bar{\mathcal{G}}_{ro}$.)*

Proof of Theorem 5.1. Suppose there exists a non-interactive protocol π GUC-realizing $\mathcal{F}_{com}$ in the Priv shared hybrid model. Then, in particular, there must be a simulator $\mathcal{S}$ such that $\text{EXEC}_{\mathcal{F}_{com},\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ for a particular adversary $\mathcal{A}$ and Priv-externally constrained environment $\mathcal{Z}$, which are constructed as follows.

Let $\mathcal{A}$ be a "dummy adversary" that simply forwards protocol flows between corrupt parties and the environment (*i.e.*, when $\mathcal{A}$ receives a message from $\mathcal{Z}$, it will send the message on behalf of some specified corrupt party; similarly, whenever a corrupt party receives a message, $\mathcal{A}$ simply forwards it to $\mathcal{Z}$). Let $\mathcal{Z}$ operate by first corrupting party S (the committer), then choosing a random bit $b \xleftarrow{\$} \{0,1\}$ and running the commit phase of π on behalf of S in order to obtain commitment κ. Wherever π makes oracle queries to Priv, $\mathcal{Z}$ issues the same queries on behalf of S (relying on monotonic consistency to be sure that it will obtain at least the same information

as an honest S would). $\mathcal{Z}$ sends κ to $\mathcal{A}$, and waits for party R to output $(\texttt{receipt}, \ldots)$. Next, $\mathcal{Z}$ runs the reveal phase of π on behalf of S (again, issuing queries to Priv where necessary) and forwards the corresponding messages through $\mathcal{A}$. Finally, $\mathcal{Z}$ waits for R to output $(\texttt{reveal}, \mathsf{sid}, \hat{b})$ and if $b = \hat{b}$ then $\mathcal{Z}$ outputs 1; otherwise, $\mathcal{Z}$ outputs 0.

Clearly, if the GUC experiments above must remain indistinguishable, $\mathcal{S}$ must cause R to output $\hat{b} = b$ with overwhelming probability. Since $\mathcal{S}$ is interacting with $\mathcal{F}_{com}$, it must specify the value of $\hat{b}$ to $\mathcal{F}_{com}$ *prior to* the point where R outputs $(\texttt{receipt}, \ldots)$, which always occurs before $\mathcal{Z}$ has initiated the reveal phase of π. That is, when $\mathcal{A}$ feeds $\mathcal{S}$ with an honestly generated commitment κ for a bit b, $\mathcal{S}$ will immediately compute a bit $\hat{b}$ such that $\hat{b} = b$ with overwhelming probability. Therefore, $\mathcal{S}$ acts like an "extractor" for commitments. However, we stress that while computing $\hat{b}$, $\mathcal{S}$ expects to have access to the oracle Priv – and, in particular, we note that party S is *corrupt* so that $\mathcal{S}$ may ask queries for S which would not be answered when S is honest (we will see how this matters shortly).

Intuitively, we have just shown that $\mathcal{S}$ can be used to extract a commitment sent by honest parties, violating the natural "hiding" property of the commitment, although this extractor requires access to the private oracle on behalf of the committer. Indeed, this "extractor" requires access to the private oracle for a *corrupt* committer, and therefore one might think this extractor is potentially "harmless" since it only violates the security of honest parties *after* they become corrupt. However, security against adaptive corruptions requires that *past transcripts* sent by honest parties who later become corrupt remain indistinguishable from *simulated transcripts* (which were created while the party was still honest). Of course, the simulator does not know the inputs of honest parties, so simulated commitments must be *independent* of the actual bit being committed to – and herein lies the contradiction. If there is an extractor that can open honest commitments to reveal the committed bit with overwhelming probability (when the committing party has later become corrupt), then this extractor distinguishes honest commitments from simulated commitments (where the extractor can only be correct/incorrect with probability 1/2 for a commitment to a random bit, assuming it even generates an output).

More formally, we will show that the existence of the simulator $\mathcal{S}$ above contradicts the security of π against adaptive corruptions, by creating a particular environment $\mathcal{Z}'$ which succeeds in distinguishing $\text{EXEC}_{\mathcal{F}_{com},\mathcal{S}',\mathcal{Z}'}$ from $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}'}$ after an adaptive corruption operation for

any simulator $\mathcal{S}'$ (as before, $\mathcal{A}$ is just a "dummy adversary"). As a notational convenience, we will write $\mathcal{S}^{\text{Priv}}(S, \kappa)$ to denote the output bit $\hat{b}$ produced by the simulation above, when running on (honestly generated) commitment κ sent by S – recalling that $\mathcal{S}$ can only be run when S is corrupt.

Our $\mathcal{Z}'$ proceeds by corrupting R at the outset, and then choosing random a bit $b \xleftarrow{\$} \{0,1\}$, which it gives as input to the honest party S. It then expects to obtain κ (the output of the commit phase) from the adversary. After receiving κ, $\mathcal{Z}'$ instructs the honest party to reveal b, completing the protocol. In accordance with the forward security corruption model, $\mathcal{Z}'$ is now allowed to corrupt S, which will enable to $\mathcal{Z}'$ to obtain complete access to Priv for S. Once this access has been obtained, $\mathcal{Z}'$ is free to compute $\hat{b} \leftarrow \mathcal{S}^{\text{Priv}}(S, \kappa)$. In the real world experiment (where protocol π is being attacked by the dummy adversary), the distribution of κ is exactly identical to its distribution in the original setting above where $\mathcal{S}$ outputs $\hat{b} = b$ with overwhelming probability. On the other hand, in the ideal world experiment (where $\mathcal{F}_{com}$ is being attacked by $\mathcal{S}'$), we know that $\mathcal{S}'$ must produce κ independently of the bit b (since b is the hidden input of the honest party, sent only to $\mathcal{F}_{com}$ which hides it from $\mathcal{S}$ information theoretically). This means that in the ideal world, we must have that $\hat{b} = b$ with probability at most $1/2$, since the entire computation of $\hat{b}$ is independent of b! Therefore, $\mathcal{Z}'$ can distinguish between the real world and ideal world experiments with probability at least $1/2 - \mathsf{negl}(\lambda)$, contradicting our assumption that π is GUC-secure. □

Remark: The class of shared functionalities modeled by Priv is very large indeed, making this impossibility result quite strong. Not only do *all* the global setups described in Section 2.2 ($\bar{\mathcal{G}}_{crs}$, $\bar{\mathcal{G}}_{acrs}$, $\bar{\mathcal{G}}_{krk}$, and $\bar{\mathcal{G}}_{ro}$) fit the modeling requirements of Priv, so would most natural formulations of a shared functionality for *one-time* shared key setups. Notice, whereas our earlier impossibility result in Section 3.2.2 *implicitly* held even in the presence of shared key setup, since authenticated channels are sufficient to realize such setup (see Section 3.2.4), the same is not automatically true here. Indeed, our realization of shared key setup based on authenticated channels is *interactive*, and therefore one might otherwise think shared key setups can suffice to realize non-interactive commitments with forward security. The ability to model shared key setup as a shared functionality satisfying the aforementioned Priv oracle shows that this is not so. In fact, even if we

combine a shared key setup with all our other setup functionalities ($\bar{\mathcal{G}}_{crs}$, $\bar{\mathcal{G}}_{acrs}$, $\bar{\mathcal{G}}_{krk}$, and $\bar{\mathcal{G}}_{ro}$) into a single all-encompassing global setup, our impossibility result still holds.

Next, we will prove that the same impossibility extends to NIZK proofs for many natural NP-relations. More formally, we describe the ideal Zero-Knowledge functionality for relation R, $\mathcal{F}_{zk}^{R}$, is described in Figure 5.1.[3] Our impossibility result shows that it is impossible to have forward secure non-interactive GUC-realizations of $\mathcal{F}_{zk}^{R}$ for non-trivial relations R (that are not already trivialized by the shared functionality for the global setup[4]).

Functionality $\mathcal{F}_{zk}^{R}$

$\mathcal{F}_{zk}$, parameterized by a binary relation R and running with a prover P, a verifier V, and an adversary $\mathcal{S}$, proceeds as follows upon receipt of a message $(\texttt{ZK-prover}, sid, P, V, x, w)$ from the prover P:

If $(x, w) \in R$, then send $(\texttt{ZK-proof}, sid, P, V, x)$ to V and $\mathcal{S}$, and halt. Otherwise halt.

Figure 5.1: The Zero-Knowledge Functionality for Relation R

Theorem 5.2. *We say that an "oracle" (or Interactive Turing Machine) is* monotonically consistent *if it always returns the same response to party P when queried repeatedly on the same input by party P (even across separate sessions), except that it may choose not to respond to some queries when P is honest (otherwise, consistency holds independently of P's corruption status). Let* Priv *denote any PPT monotonically consistent oracle (whose outputs may depend on the* pid *of the querying party, but not the* sid*).*

Further, we say that an NP-relation R defining some language L is non-trivial *if we believe*

[3] Technically, the relation R might be determined by system parameters, which form part of a CRS. Here, we note that the same CRS *must* be used in both the "ideal" and "real" settings (*e.g.*, using the $\bar{\mathcal{G}}_{crs}$ modeling).

[4] Of course, it is easy to see how one might achieve non-interactive proofs for certain languages related to the global setup. For example, if the global setup is a PKI that uses key registration with knowledge, parties can trivially prove the statement that their public keys are "well-formed" (without even communicating at all!) since the global setup already asserts the verity of this statement on their behalf. Therefore, our impossibility result does not *necessarily* extend to cases where the relation R to be proved is determined by system parameters, but we will only make use of relations that are not trivialized by the presence of the system parameters (where the impossibility result still holds).

*that no PPT algorithm efficiently decides membership in L (*i.e.*, $L \notin BPP$). In particular, R is* non-trivial with respect to oracle Priv *if there is no PPT algorithm for efficiently deciding membership in L even when given oracle access to* Priv *(for arbitrary party identifiers, and even with all parties being marked as corrupt).*

There exists no non-interactive (single message), terminating protocol π that GUC-realizes $\mathcal{F}_{zk}^{R}$ with forward security (even in the erasure model), using only the shared functionality for Priv*, for any NP-relation R that is non-trivial with respect to* Priv*. This holds even if the communication is ideally authentic. (In particular, we note that* Priv $= \bar{\mathcal{G}}_{acrs}$ *and* Priv $= \bar{\mathcal{G}}_{krk}$ *are efficient monotonically consistent oracles, even if they are combined with the random oracle shared functionality $\bar{\mathcal{G}}_{ro}$.)*

Proof of 5.2. The proof is entirely analogous to the proof of Theorem 5.1, and therefore we will only sketch it at a high level and direct the reader to the previous proof for further details. Here will call the prover P and the verifier V.

Assuming there is a non-interactive and GUCsecure realization of $\mathcal{F}_{zk}^{R}$, we first follow the approach of Theorem 5.2 in order show that (using a similar shorthand notation) there exists an extracting simulator $\mathcal{S}^{\text{Priv}}(P, x, \psi)$. For any $x \in L$, this extracting simulator is capable of computing a witness w such that $(x, w) \in R$ if ψ is an honestly generated non-interactive proof according to protocol π. However, $\mathcal{S}^{\text{Priv}}(P, x, \psi)$ expects to be run *after* the corruption of P, and it we are guaranteed that it will succeed in extracting a valid witness w (from any honestly generated proof ψ) with overwhelming probability in that scenario.

Then we construct an environment $\mathcal{Z}'$ which, parameterized by any $(x, w) \in R$, first feeds (x, w) to an honest prover P, and then obtains the resulting protocol flow ψ. Note that ψ is the protocol flow that is either observed by the dummy adversary running in the real world experiment, or is being "faked" by some simulator in the ideal model. The environment then corrupts the honest prover (after completion of the proof protocol), and runs $\mathcal{S}^{\text{Priv}}(P, x, \psi)$ to obtain w. In particular, since w must be valid with overwhelming probability in the real world, it must also be valid with overwhelming probability in the ideal world running with *some* (efficient) simulator $\mathcal{S}'$ (or else the environment can successfully distinguish the two experiments, contradicting the claimed GUC-security of the protocol). However, the value of w is information

theoretically hidden from $\mathcal{S}'$ by $\mathcal{F}_{zk}^{R}$, so its clear that $\mathcal{S}'$ must output ψ given only x and access to the Priv oracle (in particular, while V is corrupt and P is honest).

To conclude the proof, we show how to obtain a witness w for statement x using only a Priv oracle, contradicting the non-triviality of L with respect to Priv. Given any statement x, we first pick some party P to act as the prover, and V to act as the verifier. Then we run $\mathcal{S}'^{\text{Priv}}(x)$ to produce a "fake" proof ψ. Finally, we run $\mathcal{S}^{\text{Priv}}(P, x, \psi)$ to obtain w such that $(x, w) \in R$. Since this entire procedure produces a valid witness w for any $x \in L$ while using only oracle access to Priv, we have successfully contradicted the non-triviality of L with respect to Priv. □

The lesson we learn from these impossibility results is that we *must* have some interactivity in protocols for many natural tasks if we wish to achieve forward security in the GUC-framework (with reasonable setup assumptions). That is, there is a lower bound of at least two messages required for protocols that directly imply GUC-secure commitments or zero-knowledge (without adding further interactivity). This raises the question of whether or not we can achieve *round optimal* commitments and zero-knowledge proofs, and how many messages are required in the optimal case. In Section 5.5, we will show how to realize both GUC-secure commitments *and* zero-knowledge protocols for any NP-relation R using only two messages *in the random oracle model*. Therefore, we can meet the lower bound in the random oracle model. It remains an open question if the 4 message commitment protocol of Section 4.3.2 can be further improved in the plain ACRS model (without using random oracles). Similarly, it is an open question if one can improve upon the 6 message protocol for general zero-knowledge proofs implied by the commitment scheme of Section 4.3.2 (using the technique from [30] for construct $\mathcal{F}_{zk}$ from $\mathcal{F}_{com}$) without resorting to random oracles.

5.2 Efficient GUC Zero-Knowledge in the ACRS Model

We now describe a general transformation from any dense Ω-protocol Π for a relation R to a GUC-secure zero-knowledge proof for the relation R in the ACRS ($\bar{\mathcal{G}}_{acrs}$) model.[5] In our construction, we will use an identity-based trapdoor commitment (IBTC) scheme (see Section 2.4).

[5]Recall that, by the results of Section 2.3.1 and Section 2.3.2, it is possible to construct these dense Ω-protocols for any NP-relation under reasonable assumptions.

Commitments in this scheme are written $\mathsf{Com}_{ID}(d, m)$, where ID is identity of the recipient, d is some random coins, and m is the committed message. We also denote by Φ the space of extended reference parameters for Π.

The ACRS Setup is instantiated using the Setup function of the IBTC. In addition, any system parameters Λ for the Ω-protocol are placed in the public value of the augmented CRS, alongside the output of Setup. Note that there is no trapdoor associated with the system parameter for the Ω-protocol, so this system parameter is essentially a "standard" CRS. A critical difference between our approach and that of Garay et al [50] is that the reference parameter for the Ω-protocol is not placed in the CRS; rather, a fresh reference parameter ρ is generated with every run of the protocol, using the three-move "coin-tossing" protocol introduced in Section 4.3.2 (which is where we need the IBTC).

We describe our protocol for realizing $\mathcal{F}_{zk}^{R}$ in Figure 5.2 below.

Protocol DOZK

Running between a prover P (with private input (x, w)) and a verifier V (who are both initialized with knowledge of a common reference string $\mu = PK$ that is a public key for an IBTC scheme) the protocol DOZK proceeds as follows:

1. V computes $\rho_1 \xleftarrow{\$} \Phi$, forms a commitment $\kappa_1 = \mathsf{Com}_P(d_1, \rho_1)$, and sends κ_1 to P.
2. P computes $\rho_2 \xleftarrow{\$} \Phi$ and sends ρ_2 to V.
3. V first verifies that $\rho_2 \in \Phi$, and then sends the opening (d_1, ρ_1) to P.
4. P verifies that (d_1, ρ_1) is a valid opening of κ_1, and that $\rho_1 \in \Phi$.
   Both P and V locally compute $\rho \leftarrow \rho_1 \cdot \rho_2$.
5. P initiates the dense Ω-protocol Π, in the role of prover, using its witness w for x. P computes the first message a of that protocol, forms the commitment $\kappa' = \mathsf{Com}_V(d', a)$, and sends κ' to V.
6. V sends P a challenge c for protocol Π.
7. P computes a response z to V's challenge c, and sends (d', a, z) to V.
   P then **erases** the random coins used by Π.
8. V verifies that (d', a) is a valid opening of κ' and that (a, c, z) is an accepting conversation for Π.

Figure 5.2: Protocol DOZK for realizing $\mathcal{F}_{zk}^{R}$ in the $\bar{\mathcal{G}}_{acrs}$ shared hybrid model, using any dense Ω-protocol Π for the relation R.

Theorem 5.3. *The protocol described above GUC-emulates the $\mathcal{F}_{zk}^{R}$ functionality in the secure-channels model, with security against adaptive corruptions (with erasures).*

Proof of Theorem 5.3. We first observe that the protocol above only makes use of a single shared

functionality, $\bar{\mathcal{G}}_{acrs}$. Therefore, we are free to make use of the equivalence theorem and EUC model (see Chapter 1. This allows us to prove the GUC security of the protocol using the standard techniques of the UC framework, with only a single (but crucial) modification – we will allow the environment access to the shared functionality.

Let $\mathcal{A}$ be any PPT adversary attacking the above protocol. We describe an ideal adversary $\mathcal{S}$ attacking the ideal protocol for $\mathcal{F}_{zk}^{R}$ that is indistinguishable from $\mathcal{A}$ to any distinguishing environment $\mathcal{Z}$, in the presence of a shared setup $\bar{\mathcal{G}}_{acrs}$. In standard fashion, $\mathcal{S}$ will run a copy of $\mathcal{A}$ internally. We now formally describe how $\mathcal{S}$ interacts with its internal copy of $\mathcal{A}$.

First, we give the remaining details of the simulator, which are essentially standard fare in the UC literature.

Initialization. All parties are assumed to be initialized with a copy of the common reference string PK published by $\bar{\mathcal{G}}_{acrs}$ during its global initialization phase. If the parties have not already been so initialized, $\mathcal{S}$ activates a copy of the $\bar{\mathcal{G}}_{acrs}$ shared functionality, which then proceeds with the initialization. (Notice, an external copy of the globally shared $\bar{\mathcal{G}}_{acrs}$ functionality is actually being invoked by $\mathcal{S}$, and $\mathcal{S}$ does not attempt to initialize any parties directly.)

Simulating communication with $\mathcal{Z}$. $\mathcal{S}$ simply forwards all communications between its internal copy of $\mathcal{A}$ and $\mathcal{Z}$.

Simulating communication with $\bar{\mathcal{G}}_{acrs}$. $\mathcal{S}$ simply forwards all communications between its internal copy of $\mathcal{A}$ and $\bar{\mathcal{G}}_{acrs}$.

Simulating a proof between two honest parties, P and V. Since we are in the secure channels model, $\mathcal{S}$ simply notifies $\mathcal{A}$ that communications (with messages of appropriate length for a proof protocol) have taken place between P and V. If $\mathcal{A}$ blocks any communications, $\mathcal{S}$ blocks V from receiving the output of $\mathcal{F}_{zk}^{R}$.

If either P or V is corrupted during the execution of the protocol, or subsequent to its completion, the protocol transcript preceding the corruption event is generated using the corresponding technique described below (including provisions for the internal state of the corrupt party). Intuitively, this retroactive generation of protocol flows this is made possible by the non-committing encryption (NCE) scheme used to implement secure channels.

Simulating a proof between an honest P and corrupt V. The following simulation strategy is employed whenever P is honest and V is corrupted at any point prior to, or during, the execution of the protocol.

$\mathcal{S}$, upon notification from $\mathcal{F}_{zk}^{R}$ of a successful proof from P of statement x, proceeds as follows.

First, acting on behalf of the corrupt party V, $\mathcal{S}$ obtains the trapdoor SK_V from $\bar{\mathcal{G}}_{acrs}$.

Next, $\mathcal{S}$ runs the coin-tossing phase of the protocol with the corrupt party V (being controlled by $\mathcal{S}$'s internal copy of $\mathcal{A}$) normally. Upon completion of the coin-tossing phase at Step 5, rather than sending a commitment to the first message sent by Π (which would require the witness w as an input) as per the protocol specification, $\mathcal{S}$ obeys the following procedure for the next 3 steps of the protocol:

5. $\mathcal{S}$ computes $(\hat{\kappa}', \alpha) \leftarrow \mathsf{ECom}(V, SK_V)$. $\mathcal{S}$ then sends the equivocable commitment $\hat{\kappa}'$ to the corrupt verifier V (which is part of $\mathcal{S}$'s internal simulation of $\mathcal{A}$).
6. $\mathcal{S}$ receives a challenge c from the corrupt verifier V.
7. $\mathcal{S}$ runs the HVZK simulator ZKSim for protocol Π on input (Λ, ρ, x, c), obtaining messages a and z. $\mathcal{S}$ then equivocates $\hat{\kappa}'$, by computing $d' \leftarrow \mathsf{Eqv}(V, SK_V, \hat{\kappa}', \alpha, a)$, and sends d', a, z to the corrupt verifier V.

Observe that this simulation is done entirely in a straight-line fashion, and requires only the trapdoor SK_V belonging to corrupt party V.

If P is also corrupted at some point during this simulation, $\mathcal{S}$ must generate P's internal state information and provide it to $\mathcal{A}$. If P is corrupted prior to Step 5, then $\mathcal{S}$ can easily provide the random coins used by P in all previous steps of the protocol (since those are simply executed by $\mathcal{S}$ honestly). A corruption after Step 5 but before Step 7 is handled by creating an honest run of protocol Π using witness w (which was revealed to $\mathcal{S}$ immediately upon the corruption of P), and computing the internal value d' via $d' \leftarrow \mathsf{Eqv}(V, SK_V, a)$. κ', where a is now the honestly generated first message of Π. Finally, if corruption of P occurs after Step 7 of the simulation, the internal state is easily generated to be consistent with observed protocol flows, since they already contain all relevant random coins, given the erasure that occurs at the end of Step 7.

Intuitively, the faithfulness of this simulation follows from the equivocability and binding properties of commitments, and the HVZK and dense reference parameters properties of the

Ω-protocol Π. We stress that while the *proof* of this requires a rewinding argument (specifically, see Lemma 4.1), the simulation itself is straight-line.

Simulating a proof between a corrupt P and honest V. The following simulation strategy is employed whenever V is honest, and P is corrupted at any point prior to or during the execution of the protocol.

First, acting on behalf of the corrupt party P, $\mathcal{S}$ obtains the trapdoor SK_P from $\bar{\mathcal{G}}_{acrs}$.

Then $\mathcal{S}$ generates a pair (ρ, τ) using the *RefGen* algorithm for Π, and "rigs" the coin-tossing phase of the protocol by playing the role of V (communicating with the internal simulation of the corrupt party P) and modifying the initial steps of the protocol as follows:

1. $\mathcal{S}$ computes $(\hat{\kappa}_1, \alpha) \leftarrow \mathsf{ECom}(P, SK_P)$, and sends $\hat{\kappa}_1$ to P.
2. P replies by sending some string ρ_2 to V.
3. $\mathcal{S}$ computes $\rho_1 \leftarrow \rho \cdot \rho_2^{-1}$, and $d_1 \leftarrow \mathsf{Eqv}(P, SK_P, \hat{\kappa}_1, \alpha, \rho_1)$.
   $\mathcal{S}$ first verifies that $\rho_2 \in \Phi$. Then $\mathcal{S}$ sends the opening (d_1, ρ_1) to P.

The remainder of the protocol is simulated honestly.

Observe that the outcome of this coin-flipping phase will be the same ρ generated by $\mathcal{S}$ at the start of the protocol (along with its corresponding trapdoor information τ). If and when the verifier accepts, $\mathcal{S}$ runs the trapdoor extractor $\mathcal{E}_{\mathrm{td}}$ for Π on input $(\Lambda, \tau, x, a, c, z)$ to obtain a witness w for x. $\mathcal{S}$ then sends the pair (x, w) to the ideal functionality $\mathcal{F}_{zk}^R$ on behalf of the corrupt prover P.

In the event that V is also corrupted at any point prior to completion of the protocol, $\mathcal{S}$ simply produces internal state for V consistent with the visible random coins in the transcript (none of the verifier's random coins are hidden by the honest protocol).

Intuitively, the faithfulness of this simulation follows from the equivocability and binding properties of commitments, and the trapdoor soundness and dense reference parameters properties of the Ω-protocol Π. Again, we stress that while the *proof* of this requires a rewinding argument (embodied by our "coin-tossing lemma", Lemma 4.1, from Section 4.3.5), the simulation itself is straight-line.

Now that we have fully described the behavior of $\mathcal{S}$, it remains to prove that $\mathcal{S}$ interacting with $\mathcal{F}_{zk}^R$ (the ideal world interaction) is indistinguishable from $\mathcal{A}$ interacting with the protocol

(the real-world interaction), from the standpoint of any environment $\mathcal{Z}$ with access to $\bar{\mathcal{G}}_{acrs}$. We stress that even $\mathcal{Z}$ cannot obtain trapdoor information from $\bar{\mathcal{G}}_{acrs}$ for any honest parties, since $\bar{\mathcal{G}}_{acrs}$ will not respond to requests for such trapdoors.

We structure our indistinguishability proof as a sequence of games, starting with the unaltered real-world interaction and proceeding by steps towards the ideal world interaction.

I_0 **- Real-world interaction** The original protocol runs with adversary $\mathcal{A}$.

I_1 **- Simulating interactions between two honest parties** This interaction is the same as I_0, only the computation of the actual protocol messages between two honest parties is delayed until one of them becomes corrupted (at which point, $\mathcal{A}$ expects to learn the corrupted party's history via examination of its internal state).

Given that we are in the secure channels model (which implies that any messages sent between honest parties remains entirely private until one of them is corrupted) this is only a conceptual change to I_0, so the distributions of these two games are trivially identical.

I_2 **- Modifying** (κ', d') **sent to corrupt verifier** When the verifier is corrupt but the prover is honest, we have the honest prover replace the commitment κ' to be sent in Step 5 of the protocol with an equivocable commitment opened to the same value. That is, we provide the honest prover with the trapdoor information SK_V of the corrupt verifier, and we modify Steps 5-7 of the protocol as follows:

5. P starts the Ω-protocol Π for relation R, with common input (Λ, ρ, x). As usual, P plays the role of the prover in Π and computes the first message a.
   P computes $(\hat{\kappa}', \alpha) \leftarrow \mathsf{ECom}(V, SK_V)$.
   P then sends the equivocable commitment $\hat{\kappa}'$ to the corrupt verifier V.
6. P receives a challenge c from the corrupt verifier V.
7. P computes a response z to V's challenge c. P computes $d' \leftarrow \mathsf{Eqv}(V, SK_V, \hat{\kappa}', \alpha, a)$, and sends d', a, z to the corrupt verifier V.

I_3 **- Modifying** (a, z) **sent to a corrupt verifier** Once again, this change affects only the scenario where the prover is honest and the verifier is corrupt.

This interaction is the same as I_2, only the values of a, z sent by the prover are generated using the HVZK Simulator for Π, rather than using Π directly.

That is, modify Step 7 of the protocol as follows:

7. P runs the HVZK simulator ZKSim for protocol Π on input (Λ, ρ, x, c), obtaining simulated messages a and z (this values are used instead of those that would have been generated via Π).
   P computes $d' \leftarrow \mathsf{Eqv}(V, SK_V, \hat{\kappa}', \alpha, a)$, and sends d', a, z to the corrupt verifier V.

I_4 **- Modifying the coin-toss commitment for corrupt provers** This interaction is exactly the same as I_3 in case the verifier is corrupt. However, in the event that the prover is corrupt, we modify the coin-flipping stage of the protocol to replace the commitment sent by the honest verifier with an equivocable commitment opened to the same value.

That is, we provide the honest verifier with the trapdoor information SK_P of the corrupt prover, and we modify Steps 1-3 of the protocol as follows:

1. V computes $\rho_1 \xleftarrow{\$} \Phi$.
   V computes $(\hat{\kappa}_1, \alpha) \leftarrow \mathsf{ECom}(P, SK_P)$, and sends $\hat{\kappa}_1$ to P.
2. P replies by sending some string ρ_2 to V.
3. V computes $d_1 \leftarrow \mathsf{Eqv}(P, SK_P, \hat{\kappa}_1, \alpha, \rho_1)$.
   V first verifies that $\rho_2 \in \Phi$. Then V sends the opening (d_1, ρ_1) to P.

I_5 **- Rigging the coin-flipping for corrupt provers** This interaction is the same I_4, only in the case where the prover is corrupt we further modify the coin-flipping phase of the protocol by changing the honest verifier's opening in Step 3 in order to "rig" the outcome of the coin-flipping to a pre-specified choice of reference parameter ρ.

Specifically, we make the following change:

3. P generates a pair $(\rho, \tau) \leftarrow \mathit{RefGen}(\Lambda)$, and sets $\rho_1 = \rho \cdot \rho_2^{-1}$ (rather than choosing ρ_1 at random).
   V computes $d_1 \leftarrow \mathsf{Eqv}(P, SK_P, \hat{\kappa}_1, \alpha, \rho_1)$.
   V first verifies that $\rho_2 \in \Phi$. Then V sends the opening (d_1, ρ_1) to P.

I_6 **- The ideal world** The most significant difference between I_5 and the final, simulated interaction in the ideal world is that the simulator uses the rigged coin-flipping technique to "trapdoor extract" a witness when the prover is corrupt – and then the honest verifier's output is taken from the $\mathcal{F}_{zk}^{R}$ functionality, rather than a direct verification of the protocol messages. There is a minor difference when the verifier is corrupt – now it is the simulator who generates the protocol messages of the honest prover, rather than the prover itself. There are also corresponding changes in message delivery, none of which are ultimately visible to the environment or the (now, internally simulated) adversary.

Provided that the trapdoor extraction procedure produces a valid witness whenever the corrupt prover succeeds in convincing an honest verifier, I_5 and I_6 are identically distributed.

Claim 5.4. I_2 is indistinguishable from I_1.

Proof. The proof follows directly from the equivocability property of IBTCs. Namely, any environment that can distinguish I_2 from I_3 can easily be made to distinguish equivocable commitments from honest commitments to the same value by simply plugging in the challenge commitments appropriately (and using $\bar{\mathcal{G}}_{acrs}$ to provide the same public setup parameters as the challenger's IBTC system). □

Claim 5.5. I_3 is indistinguishable from I_2.

Proof. The proof follows by observing that if I_3 can be distinguished from I_2, then either (a) extended reference parameters can be distinguished from reference parameters, or (b) real conversations for Π can be distinguished from simulated conversations. The reduction follows by an application of Lemma 4.1, which allows us to "rig" the coin-tossing phase of the protocol (which is taking place in either interaction I_3 or I_2) to yield the same ρ specified by the challenger in the HVZK attack game. (Observe that we require the dense reference parameter property of the Ω-protocol, to satisfy the requirements of the Lemma.) In particular, Lemma 4.1 assures us that any successful attacker distinguishing our protocol with its coin-tossing phase can be replaced by an equivalent attacker who distinguishes a modified version of the protocol that replaces the coin-tossing phase (and its resultant choice of ρ) by directly sampling ρ instead. This means that we can now ensure that the challenge reference parameter is being used for the Ω protocol (since

we may substitute the random challenge parameter in place of the coin-tossing). Once we have substituted the in the proper reference parameter ρ from the challenger, we may simply send (x, w, c) to the HVZK game challenger (where w is taken from the input to the honest prover P in our protocol interaction), and replace the honest prover P's choice of (a, c) by the response from the challenger. Distinguishing I_3 from I_2 now corresponds precisely to guessing the HVZK challenge bit b. □

Claim 5.6. I_4 is indistinguishable from I_3.

Proof. This is a straightforward reduction to equivocability of IBTCs, as before. □

Claim 5.7. I_5 is indistinguishable from I_4.

Proof. We begin by considering a modified interaction I_4 where V computes ρ_1 by first selecting ρ uniformly at random, and then computing $\rho_1 \leftarrow \rho \cdot \rho_2^{-1}$. It is easy to see that the distribution of ρ_1 is unaltered, and thus we have made only a conceptual change to I_4.

Given this new view of I_4, it is easy to see that if we modify I_4 as per game I_5, the *only difference* is that the value ρ used by V is no longer random, but is instead chosen according to *RefGen*. From here, it is straightforward to reduce the distinguishing game to the Dense Reference Parameter property of the Ω-protocol. □

Claim 5.8. I_6 is indistinguishable from I_5.

Proof. This proof is by reduction to trapdoor extractability property of the Ω-protocol. Recall that the "rigging" of the reference string is already taken care of by the technique of I_5 (so we may easily arrange for the same ρ selected by the challenger in the extraction attack game of the Ω-protocol). The trapdoor soundness property for Π guarantees that we get a witness with overwhelming probability. □

Combining the preceding claims yields the desired proof that the real interaction I_0 is indistinguishable from the ideal interaction I_6.

□

5.3 Efficient GUC Commitments in the ACRS Model

In this section we describe an efficientGUC-secure realization of a multi-bit message variant of $\mathcal{F}_{com}$ (*i.e.*, rather than committing to a single bit, it is possible to commit to λ bits simultaneously). The realization is secure against adaptive corruptions, but only in the erasure model. Our protocol for efficiently GUC-realizing $\mathcal{F}_{com}$ is based on a modification to the protocol for efficient GUC zero-knowledge proofs from Section 5.2. This time, we will make use of a dense Ω-protocol specifically for the IBTC opening relation; here, a witness for a commitment κ with respect to an identity *ID* is a valid opening (d, m) (*i.e.*, $\mathsf{Com}_{ID}(d, m) = \kappa$). Instead of trapdoor soundness, we only require partial trapdoor soundness with respect to the function $f(d, m) := m$.

Of course, our efficient GUC-secure commitment protocol has two phases. The commit phase is essentially *identical* to the ZK protocol in Section 5.2, except that Step 5 has been modified. The reveal phase is entirely new.

Theorem 5.9. *The protocol described above GUC-emulates the $\mathcal{F}_{com}$ functionality in the secure-channels model, with security against adaptive corruptions (with erasures).*

Proof of Theorem 5.9. The proof is analogous to that of our zero-knowledge protocol, but entails some minor changes that include the partial trapdoor soundness requirement for Π. Rather than recapitulate all the details of the proof in Section 5.2, we will merely sketch changes to the proof here.

The main difference is that a slightly more specialized argument is needed to prove that I_6 is indistinguishable from I_5. In I_6, the simulator now uses the trapdoor extractor $\mathcal{E}_{\mathrm{td}}$ during the commit phase (when S is corrupted) to extract a value m to pass to the ideal functionality. Later, during the reveal phase, S may open the commitment κ inconsistently as $(\hat{d}, \hat{m})$, where $\hat{m} \neq m$; we want to argue that this happens with only negligible probability, using the partial trapdoor soundness property for Π, relative to the function $f(d, m) := m$. Suppose to the contrary that the adversary succeeds in making such an inconsistent opening with non-negligible probability, even though R accepted the conversation (a, c, z) in the Ω-protocol. Then, relying upon the binding property of the IBTC scheme (applied to the commitment κ'), we can rewind the adversary to get a second accepting conversation (a, c', z'), where $c' \neq c$, also with non-negligible probability

Protocol DOC

Protocol DOC proceeds as follows, with party S committing a message m to party R (in the shared $\bar{\mathcal{G}}_{acrs}$ hybrid model):

Commit Phase:

1. R computes $\rho_1 \xleftarrow{\$} \Phi$, forms a commitment $\kappa_1 = \mathsf{Com}_S(d_1, \rho_1)$, and sends κ_1 to S.
2. S computes $\rho_2 \xleftarrow{\$} \Phi$ and sends ρ_2 to R.
3. R first verifies that $\rho_2 \in \Phi$, and then sends the opening (d_1, ρ_1) to S.
4. S verifies that (d_1, ρ_1) is a valid opening of κ_1, and that $\rho_1 \in \Phi$.
   Both S and R locally compute $\rho \leftarrow \rho_1 \cdot \rho_2$.
5. S generates a commitment $\kappa = \mathsf{Com}_R(d, m)$, and then initiates the Ω-protocol Π, in the role of prover, using its witness (d, m).
   S computes the first message a of that protocol, forms the commitment $\kappa' = \mathsf{Com}_R(d', a)$, and sends κ and κ' to R.
6. R sends S a challenge c for protocol Π.
7. S computes a response z to R's challenge c, and sends (d', a, z) to R.
   S then **erases** the random coins used by Π.
8. R verifies that (d', a) is a valid opening of κ' and that (a, c, z) is an accepting conversation for Π, and then outputs $(\texttt{receipt}, \mathsf{sid}, S, R)$ if so.

Reveal Phase:

1. S sends the opening (d, m) to R, who verifies that (d, m) is a valid opening of κ and outputs $(\texttt{reveal}, \mathsf{sid}, S, R, m)$ if so.

Figure 5.3: Protocol DOC for realizing $\mathcal{F}_{com}$ in the $\bar{\mathcal{G}}_{acrs}$ shared hybrid model, given a partial trapdoor sound dense Ω-protocol Π for proving knowledge of an IBTC decommitment. In particular, partial trapdoor soundness must hold with respect to the function $f(d, m) = m$ (*i.e.*, it is enough for the trapdoor extraction procedure to output the committed value, rather than the entire opening of the commitment).

(this follows directly from the Reset Lemma of [10]). The partial trapdoor soundness property will guarantee that the rewinding extractor $\mathcal{E}_{\text{rw}}$, applied to these two conversations, will yield an opening of κ of the form (d, m). Now we have two openings of κ, $(\hat{d}, \hat{m})$ and (d, m), where $\hat{m} \neq m$, which breaks the binding property of the IBTC scheme — a contradiction.

□

5.4 Efficient Number-theoretic Instantiations

Here we outline a collection of related number-theoretic constructions that can be used to instantiate the protocols of Section 5.2 and Section 5.3.

While the generic Ω-protocol construction of Section 2.3.2 will certainly yield much more efficient protocols than those of Chapter 4 (at least for languages with efficient Σ-protocols) we would like to get an even more efficient protocol that avoids the cut-and-choose paradigm altogether. In this section, we briefly show how we can obtain such a protocol for GUC-secure commitments. Unlike the commitment scheme in Section 4.3.2, which could only commit single bits, the GUC commitment scheme described in Section 5.3 can be used to commit to values in a much larger set. Moreover, because of the special algebraic structure of the scheme, the GUC commitment protocol of Section 5.3 can be instantiated in such a way as to enable combination with other, well-known protocols for proving properties on committed values (*e.g.*, the that product of two committed integers is equal to a third committed integer).

To achieve these goals, we need an IBTC scheme that supports an efficient Ω-protocol. As outlined in Section 2.4 (based on a variation of an idea in [46]), to build an IBTC scheme one can use any secure signature scheme, along with an Augmented Σ-protocol for proof of knowledge of a signature on a given message. Here, the "message" to be signed will be an identity ID. Assuming the Augmented Σ-protocol is HVZK, we can turn it into a commitment scheme, as follows. For a conversation (a, c, z), the commitment is a, the value committed to is c, and the decommitment is z. To commit to a value c, one runs the HVZK simulator. The trapdoor for a given ID is a signature on ID, and using this signature, one can generate equivocable commitments just by running the actual Σ-protocol.

For our purposes, we suggest using the Waters' signature scheme [84]. Let $\mathbb{G}$ and $\mathbb{H}$ be

groups of prime order q, let $e : \mathbb{G} \to \mathbb{H}$ be an efficiently computable, non-degenerate bilinear map, and let $\mathbb{G}^* := \mathbb{G} \setminus \{1\}$. A public reference parameter consists of random group elements $\mathbf{g}_1, \mathbf{g}_2, \mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_k \in \mathbb{G}$, a description of a collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^k$, and a group element $\mathbf{h}_1$. A signature on a message m is a pair $(\mathbf{s}_1, \mathbf{s}_2) \in \mathbb{G} \times \mathbb{G}$, such that $e(\mathbf{s}_1, \tilde{\mathbf{u}}_m^{-1}) \cdot e(\mathbf{s}_2, \mathbf{g}_1) = e(\mathbf{h}_1, \mathbf{g}_2)$, where $\tilde{\mathbf{u}}_m := \mathbf{u}_0 \prod_{b_i=1} \mathbf{u}_i$ and $H(m) = b_1 \cdots b_k \in \{0,1\}^k$. Waters signature is secure assuming the CDH for the group $\mathbb{G}$. With overwhelming probability, the signing algorithm will produce a signature $(\mathbf{s}_1, \mathbf{s}_2)$ where neither $\mathbf{s}_1$ nor $\mathbf{s}_2$ are 1, so we can effectively assume this is always the case.

To prove knowledge of a Waters signature $(\mathbf{s}_1, \mathbf{s}_2) \in \mathbb{G} \times \mathbb{G}$ on a message $m \in \{0,1\}^*$, we may use the following protocol. The prover chooses $w_1, w_2 \in \mathbb{Z}_q^*$ at random, and computes $\bar{\mathbf{s}}_1 \leftarrow \mathbf{s}_1^{1/w_1}$ and $\bar{\mathbf{s}}_2 \leftarrow \mathbf{s}_2^{1/w_2}$. The prover then sends $\bar{\mathbf{s}}_1$ and $\bar{\mathbf{s}}_2$ to the verifier, and uses a standard Σ-protocol to prove knowledge of exponents $w_1, w_2 \in \mathbb{Z}_q$ such that $\gamma_1^{w_1} \gamma_2^{w_2} = \gamma$ where $\gamma_1 := e(\bar{\mathbf{s}}_1, \tilde{\mathbf{u}}_m^{-1})$, $\gamma_2 := e(\bar{\mathbf{s}}_2, \mathbf{g}_1)$, and $\gamma := e(\mathbf{h}_1, \mathbf{g}_2)$.

The identity-based commitment scheme derived from the above Σ-protocol works as follows. Let $ID \in \{0,1\}^*$ be the identity, and let $m \in \mathbb{Z}_q$ be the message to be committed. The commitment is computed as follows: $\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2 \xleftarrow{\$} \mathbb{G}^*$, $d_1, d_2 \xleftarrow{\$} \mathbb{Z}_q$, $\gamma_1 \leftarrow e(\bar{\mathbf{s}}_1, \tilde{\mathbf{u}}_{ID}^{-1})$, $\gamma_2 \leftarrow e(\bar{\mathbf{s}}_2, \mathbf{g}_1)$, $\gamma \leftarrow e(\mathbf{h}_1, \mathbf{g}_2)$, $\bar{\gamma} \leftarrow \gamma_1^{d_1} \gamma_2^{d_2} \gamma^m$. The commitment is $(\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2, \bar{\gamma})$.

A commitment $(\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2, \bar{\gamma}) \in \mathbb{G}^* \times \mathbb{G}^* \times \mathbb{H}$ is opened by revealing d_1, d_2, m that satisfies the equation $\gamma_1^{d_1} \gamma_2^{d_2} \gamma^m = \bar{\gamma}$, where $\gamma_1, \gamma_2, \gamma$ are computed as in the commitment algorithm, using the given values $\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2$.

The trapdoor for such a commitment is a Waters signature on the identity ID. Using such a signature, one can just run the Σ-protocol, and open the commitment to any value. The commitment will look the same as an ordinary commitment, unless either component of the signature is the identity element, which happens with negligible probability.

As the opening of a commitment is essentially just a representation of a group element relative to three bases, there is a standard Σ-protocol for proving knowledge of an opening of a given commitment. (It is easily verified that the protocol additionally satisfies the reverse state construction property of Augmented Σ-protocols.) Moreover, using techniques from Camenisch and Shoup [17], we can actually build an Ω-protocol for such a proof of knowledge, which avoids the cut-and-choose paradigm.

Garay et al [50] give an Ω-protocol for a very similar task, which could easily be adapted for our purposes, except that the protocol in [50] does not satisfy the dense reference parameters property, which is crucial for our construction of a GUC commitment. To appreciate the technical difficulty, the MacKenzie et al. protocol is based on Paillier encryption, using an RSA modulus N. The secret key for this encryption scheme is the factorization of N, and this is used as "global" trapdoor to a CRS in their proof of security in the UC/CRS model. However, in the GUC framework, we cannot have such a global trapdoor, which is why we make use of Camenisch and Shoup's approach.[6]

The Camenisch and Shoup approach is based on a variant of Paillier encryption, introduced in Cramer and Shoup [33], which we call here *projective Paillier encryption*. While the goal in [17] and [33] was to build a chosen ciphertext secure encryption scheme, and we only require semantic security, it turns out their schemes do not require that the factorization of the RSA modulus N be a part of the secret key. Indeed, the modulus N can be generated by a trusted party, who then erases the factorization and goes disappears, and N can be used as a shared system parameter. We can easily "strip down" the scheme in [17], so that it only provides semantic security. The resulting Ω-protocol will satisfy all the properties we need to build a GUC commitment, under standard assumptions (the Quadratic Residuosity, Decision Composite Residuosity, and Strong RSA).

We now give the details of these constructions.

5.4.1 An Efficient IBTC Supporting an Ω-protocol

We present an efficient identity-based commitment scheme for which an efficient Ω-protocol for proof of possession of an opening may be readily constructed.

Waters' Signature Scheme

Our starting point is Waters signature scheme, which we review here. Let $\mathbb{G}$ and $\mathbb{H}$ be a groups of prime order q, let $e : \mathbb{G} \to \mathbb{H}$ be an efficiently computable, non-degenerate bilinear map, and

[6] It should be noted that the "mixed commitments" of Damgard and Nielsen [35] also have a very similar global extraction trapdoor, which is why we also cannot use them to build GUC commitments.

let $\mathbb{G}^* := \mathbb{G} := \{1\}$.

system parameters: a description of $\mathbb{G}$, $\mathbb{H}$, and e, along with

- random group elements $\mathbf{g}_1, \mathbf{g}_2, \mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_k \in \mathbb{G}$,
- a description of a collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^k$.

key generation: a random $x \in \mathbb{Z}_q$ is chosen, $\mathbf{h}_1 \in \mathbb{G}$ is computed as $\mathbf{h}_1 \leftarrow \mathbf{g}_1^x$, and $\mathbf{h}_2 \in \mathbb{G}$ is computed as $\mathbf{h}_2 \leftarrow \mathbf{g}_2^x$; the public key is $\mathbf{h}_1$, the secret key is $\mathbf{h}_2$.

signing: to sign a message $m \in \{0,1\}^*$, the hash $H(m) = b_1 \cdots b_k$ is computed (where each $b_i \in \{0,1\}$), a random $r \in \mathbb{Z}_q$ is chosen, and the signature $(\mathbf{s}_1, \mathbf{s}_2) \in \mathbb{G} \times \mathbb{G}$ is computed as follows:

$$\mathbf{s}_1 \leftarrow \mathbf{g}_1^r, \quad \mathbf{s}_2 \leftarrow \mathbf{h}_2 \tilde{\mathbf{u}}_m^r,$$

where

$$\tilde{\mathbf{u}}_m := \mathbf{u}_0 \prod_{b_i=1} \mathbf{u}_i.$$

verification: given a message $m \in \{0,1\}^*$ and a signature $(\mathbf{s}_1, \mathbf{s}_2) \in \mathbb{G} \times \mathbb{G}$, the verification algorithm checks that

$$e(\mathbf{s}_1, \tilde{\mathbf{u}}_m^{-1}) \cdot e(\mathbf{s}_2, \mathbf{g}_1) = e(\mathbf{h}_1, \mathbf{g}_2),$$

where $\tilde{\mathbf{u}}_m$ is as above.

The Waters signature is secure under the *computational Diffie-Hellman (CDH)* assumption in $\mathbb{G}$, together with the assumption that H is collision resistant.

5.4.2 Proof of Knowledge of a Waters Signature

To prove knowledge of a Waters signature $(\mathbf{s}_1, \mathbf{s}_2) \in \mathbb{G} \times \mathbb{G}$ on a message $m \in \{0,1\}^*$, we may use the following protocol:

The prover chooses $w_1, w_2 \in \mathbb{Z}_q^*$ at random, and computes

$$\bar{\mathbf{s}}_1 \leftarrow \mathbf{s}_1^{1/w_1} \quad \text{and} \quad \bar{\mathbf{s}}_2 \leftarrow \mathbf{s}_2^{1/w_2}.$$

The prover then sends $\bar{\mathbf{s}}_1$ and $\bar{\mathbf{s}}_2$ to the verifier, and uses a standard Σ-protocol to prove knowledge of exponents $w_1, w_2 \in \mathbb{Z}_q$ such that

$$\gamma_1^{w_1}\gamma_2^{w_2} = \gamma,$$

where

$$\gamma_1 := e(\bar{\mathbf{s}}_1, \tilde{\mathbf{u}}_m^{-1}), \quad \gamma_2 := e(\bar{\mathbf{s}}_2, \mathbf{g}_1), \quad \text{and} \quad \gamma := e(\mathbf{h}_1, \mathbf{g}_2).$$

The details are as follows:

1. The prover chooses $w_1, w_2 \in \mathbb{Z}_q^*$ at random, and computes

   $$\bar{\mathbf{s}}_1 \leftarrow \mathbf{s}_1^{1/w_1} \quad \text{and} \quad \bar{\mathbf{s}}_2 \leftarrow \mathbf{s}_2^{1/w_2}.$$

   Let

   $$\gamma_1 := e(\bar{\mathbf{s}}_1, \tilde{\mathbf{u}}_m^{-1}), \quad \gamma_2 := e(\bar{\mathbf{s}}_2, \mathbf{g}_1), \quad \text{and} \quad \gamma := e(\mathbf{h}_1, \mathbf{g}_2). \tag{5.1}$$

   The prover then chooses $\bar{w}_1, \bar{w}_2 \in \mathbb{Z}_q$ at random, and computes $\bar{\gamma} \leftarrow \gamma_1^{\bar{w}_1}\gamma_2^{\bar{w}_2}$.

   The prover sends the values

   $$\bar{\mathbf{s}}_1 \in \mathbb{G}, \quad \bar{\mathbf{s}}_2 \in \mathbb{G}, \quad \bar{\gamma} \in \mathbb{H}$$

   to the verifier.
2. The verifier chooses a challenge $c \in \mathbb{Z}_q$ at random, and sends c to the prover.
3. The prover computes

   $$\hat{w}_1 \leftarrow \bar{w}_1 - cw_1 \quad \text{and} \quad \hat{w}_2 \leftarrow \bar{w}_2 - cw_2$$

   and sends the values

   $$\hat{w}_1 \in \mathbb{Z}_q, \quad \hat{w}_2 \in \mathbb{Z}_q$$

   to the verifier.
4. The verifier checks that

   $$\gamma_1^{\hat{w}_1}\gamma_2^{\hat{w}_2}\gamma^c = \bar{\gamma},$$

   where $\gamma_1, \gamma_2, \gamma$ are as defined in (5.1).

It is easily verified that this Σ-protocol is HVZK (in fact, that it satisfies the stronger reverse state construction property of Augmented Σ-protocols), at least with respect to signatures of the

form $(\mathbf{s}_1, \mathbf{s}_2)$, where $\mathbf{s}_1 \neq 1$ and $\mathbf{s}_2 \neq 1$. Indeed, for such a signature, $\bar{\mathbf{s}}_1$ and $\bar{\mathbf{s}}_2$ are independent and uniformly distributed over $\mathbb{G}^*$, and the rest of the protocol may be simulated using standard techniques. Since signatures output by the signing algorithm are of this form with overwhelming probability, this is sufficient for our purposes.

Also, this Σ-protocol satisfies the special soundness property. Indeed, given two accepting conversations with the same first flow, $(\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2, \bar{\gamma})$, one obtains $w_1, w_2 \in \mathbb{Z}_q$ such that

$$e(\bar{\mathbf{s}}_1, \tilde{\mathbf{u}}_m^{-1})^{w_1} \cdot e(\bar{\mathbf{s}}_2, \mathbf{g}_1)^{w_2} = e(\mathbf{h}_1, \mathbf{g}_2),$$

and since

$$e(\bar{\mathbf{s}}_1, \tilde{\mathbf{u}}_m^{-1})^{w_1} = e(\bar{\mathbf{s}}_1^{w_1}, \tilde{\mathbf{u}}_m^{-1}) \quad \text{and} \quad e(\bar{\mathbf{s}}_2, \mathbf{g}_1)^{w_2} = e(\bar{\mathbf{s}}_2^{w_2}, \mathbf{g}_1),$$

it follows that $(\bar{\mathbf{s}}_1^{w_1}, \bar{\mathbf{s}}_2^{w_2})$ is a valid Waters signature on m.

An IBTC Scheme

The identity-based commitment scheme derived from the above Σ-protocol works as follows. Let $ID \in \{0,1\}^*$ be the identity to be associated with the commitment, and let $m \in \mathbb{Z}_q$ be the message to be committed. The commitment is computed as follows:

$\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2 \xleftarrow{\$} \mathbb{G}^*$, $d_1, d_2 \xleftarrow{\$} \mathbb{Z}_q$
$\gamma_1 \leftarrow e(\bar{\mathbf{s}}_1, \tilde{\mathbf{u}}_{ID}^{-1})$, $\gamma_2 \leftarrow e(\bar{\mathbf{s}}_2, \mathbf{g}_1)$, $\gamma \leftarrow e(\mathbf{h}_1, \mathbf{g}_2)$
$\bar{\gamma} \leftarrow \gamma_1^{d_1} \gamma_2^{d_2} \gamma^m$
output $(\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2, \bar{\gamma})$

A commitment $(\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2, \bar{\gamma}) \in \mathbb{G}^* \times \mathbb{G}^* \times \mathbb{H}$ is opened by revealing d_1, d_2, m that satisfies the equation

$$\gamma_1^{d_1} \gamma_2^{d_2} \gamma^m = \bar{\gamma},$$

where $\gamma_1, \gamma_2, \gamma$ are computed as in the commitment algorithm, using the given values $\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2$.

The trapdoor for such a commitment is a Waters signature on the identity ID. Using such a signature, one can just run the Σ-protocol, and open the commitment to any value. The commitment will look the same as an ordinary commitment, unless either component of the signature is the identity element, which happens with negligible probability.

As the opening of a commitment is essentially just a representation of a public group element with respect to public bases, we can easily build a Σ-protocol for proving knowledge of an opening of a given commitment. Indeed, we will show how to build an efficient Ω-protocol, where the message m is trapdoor extractable.

5.4.3 An Efficient Ω-protocol for Proving Knowledge of a Representation

We begin by first reviewing the necessary number theoretic background and tools to be used in the Ω-protocol.

Number theory background

Let N be a positive integer.

- $[N]$ denotes the set $\{0, \ldots, N-1\}$;
- for $a \in \mathbb{Z}$, $a \bmod N$ denotes the unique integer $x \in [N]$ such that $a \equiv x \pmod{N}$;
- more generally, if $a, b \in \mathbb{Z}$ with $b \neq 0$ and $\gcd(b, N) = 1$, $(a/b) \bmod N$ denotes the unique integer $x \in [N]$ such that $a \equiv xb \pmod{N}$;
- $\mathbb{Z}_N$ denotes the ring of integers modulo N, and $\mathbb{Z}_N^*$ the multiplicative group of units;
- for $a \in \mathbb{Z}$, $[a]_N \in \mathbb{Z}_N$ denotes the residue class modulo N containing a.

The schemes we shall present below use as a system parameter an RSA modulus of the form $N = PQ$, where P and Q are large, distinct, "strong primes," *i.e.*, primes of the form $P = 2P' + 1$ and $Q = 2Q' + 1$, where P' and Q' are odd primes. Define $N' := P'Q'$.

Note that in all applications, no entity is required to know the factorization of N — not even a simulator in a security proof. We assume N is generated by a trusted party who immediately disappears, taking the factorization of N with it.

We shall make use of the two abelian groups $\mathbb{Z}_N^*$ and $\mathbb{Z}_{N^2}^*$. We recall some basic facts:

- $\mathbb{Z}_N^*$ is isomorphic to $\mathbb{Z}_{N'} \times \mathbb{Z}_2 \times \mathbb{Z}_2$;
- if $j_N := \{[a]_N : (a \mid N) = 1\}$, where $(\cdot \mid \cdot)$ is the Jacobi symbol, then this definition of j_N is unambiguous, and j_N is a subgroup of index 2 in $\mathbb{Z}_N^*$; observe that $[-1]_N \in j_N$;

- the subgroup of squares $(\mathbb{Z}_N^*)^2$ has index 2 in j_N; note that $[-1]_N \notin (\mathbb{Z}_N^*)^2$;
- $\mathbb{Z}_{N^2}^*$ is isomorphic to $\mathbb{Z}_N \times \mathbb{Z}_{N'} \times \mathbb{Z}_2 \times \mathbb{Z}_2$;
- the special element $\mathfrak{w} := [1+N]_{N^2} \in \mathbb{Z}_{N^2}^*$ has order N, and moreover, for each $m \in \mathbb{Z}$, we have $\mathfrak{w}^m = [1+Nm]_{N^2}$;
- if $J_N := \{[a]_{N^2} : (a \mid N) = 1\}$, then this definition of J_N is unambiguous, and J_N is a subgroup of index 2 in $\mathbb{Z}_{N^2}^*$; observe that $[-1]_{N^2} \in J_N$;
- the subgroup of squares $(\mathbb{Z}_{N^2}^*)^2$ has index 2 in J_N; moreover, for all $a \in \mathbb{Z}$, we have $[a]_{N^2} \in (\mathbb{Z}_{N^2}^*)^2$ if and only if $[a]_N \in (\mathbb{Z}_N^*)^2$; in particular, $[-1]_{N^2} \notin (\mathbb{Z}_{N^2}^*)^2$;
- the subgroup of Nth powers $(\mathbb{Z}_{N^2}^*)^N$ has index N in $\mathbb{Z}_{N^2}^*$.

Now we state the intractability assumptions we will need:

- The *Strong RSA assumption* says that given a random $\mathsf{h} \in \mathbb{Z}_N^*$, it is hard to find $\mathsf{g} \in \mathbb{Z}_N^*$ and an integer $e > 1$ such that $\mathsf{g}^e = \mathsf{h}$.
- The *Quadratic Residuosity (QR) assumption* says that it is hard to distinguish a random element of j_N from a random element of $(\mathbb{Z}_N^*)^2$.
- The *Decision Composite Residuosity (DCR) assumption* says that it is hard to distinguish a random element of $\mathbb{Z}_{N^2}^*$ from a random element of $(\mathbb{Z}_{N^2}^*)^N$.

Another convenient fact is the uniform distribution on $[N/4]$ is statistically indistinguishable from the uniform distribution on $[N']$. Similarly, the uniform distribution on $[N^2/4]$ is statistically indistinguishable from the uniform distribution on $[NN']$.

Some consequences:

Lemma 5.1. *Under the QR assumption, it is hard to distinguish a random element of J_N from a random element of $(\mathbb{Z}_{N^2}^*)^2$. Under the DCR assumption, it is hard to distinguish a random element of $(\mathbb{Z}_{N^2}^*)^2$ from a random element of $(\mathbb{Z}_{N^2}^*)^{2N}$. Under the QR and DCR assumptions, it is hard to distinguish a random element of J_N from a random element of $(\mathbb{Z}_{N^2}^*)^{2N}$.*

The proof of the preceding lemma is direct, and is left as simple exercise.

The following lemma is a simple generalization of a lemma appearing in Camenisch and Shoup [17]:

Lemma 5.2. *Under the strong RSA assumption, given random elements* $\mathsf{h}_1, \ldots, \mathsf{h}_k \in (\mathbb{Z}_N^*)^2$*, it is hard to find* $\mathsf{g} \in \mathbb{Z}_N^*$*, along with integers* $c, d_1, \ldots, d_k$*, such that*

$$\mathsf{g}^c = \mathsf{h}_1^{d_1} \cdots \mathsf{h}_k^{d_k} \quad \textit{and} \quad c \nmid d_i \textit{ for some } i = 1, \ldots, k.$$

Projective Paillier Encryption

In [33], Cramer and Shoup proposed a variation of Paillier encryption [72]. Although their motivation was completely different than ours (constructing a CCA2-secure encryption scheme), it turns out that some the ideas can be utilized here. The same ideas were also used to similar effect by Camenisch and Shoup in [17], although again, their motivation was somewhat different than ours.

In a nutshell, we present a variation of Paillier encryption that is semantically secure under the DCR assumption, and preserves essentially the same homomorphic properties of Paillier encryption; however, unlike the original Paillier scheme, the scheme we present here has a dense set of public-keys, in a sense corresponding to that of Section 2.3.2. Following the terminology in Cramer and Shoup [33], let us call this scheme the *Projective Paillier encryption scheme*.

system parameters: in addition to the RSA modulus N (of the form described in §5.4.3), the system parameters also include a random element

$$\mathfrak{g} \in (\mathbb{Z}_{N^2}^*)^{2N};$$

note that $\mathfrak{g}$ has order dividing N', and this order is equal to N' with overwhelming probability;

recall that $\mathfrak{w} := [1 + N]_{N^2} \in \mathbb{Z}_{N^2}^*$ is the special element of order N;

key generation: compute $t \xleftarrow{\$} [N/4]$ and $\mathfrak{h} \leftarrow \mathfrak{g}^t$; the public key is $\mathfrak{h}$ and the secret key is t;

encryption: to encrypt a message $m \in [N]$ using a public key $\mathfrak{h}$, the encryption algorithm runs as follows:

$$r \xleftarrow{\$} [N/4], \quad \mathfrak{u} \leftarrow \mathfrak{g}^r, \quad \mathfrak{v} \leftarrow \mathfrak{h}^r \mathfrak{w}^m;$$

the ciphertext is $(\mathfrak{u}, \mathfrak{v})$;

decryption: given a ciphertext $(\mathfrak{u}, \mathfrak{v})$ and a secret key t, the decryption algorithm computes

$$\mathfrak{w}' \leftarrow \mathfrak{v}/\mathfrak{u}^t;$$

if $\mathfrak{w}'$ is of the form $[1 + Nm]_{N^2}$ for some $m \in [N]$, then the algorithm outputs m, and otherwise, it outputs "reject."

Lemma 5.3. *Under the DCR assumption, the Projective Paillier encryption scheme is semantically secure.*

Proof. This follows from results in Cramer and Shoup [33]; however, we sketch the idea directly, as follows. Suppose we encrypt a message m as $(\mathfrak{u}, \mathfrak{v}) := (\mathfrak{g}^r, \mathfrak{h}^r \mathfrak{w}^m)$, where r is chosen at random from $[N/4]$. Certainly, we may instead choose r at random $[N^2/4]$ without affecting security. Under the DCR assumption (see Lemma 5.1), we may instead choose $\mathfrak{h}$ of the form $\mathfrak{g}^t \mathfrak{w}^s$, where s is chosen at random from $[N]$, subject to $\gcd(s, N) = 1$, without affecting security. Now suppose we instead choose r at random from $[NN']$, which also does not affect security. Writing $r = r_0 + N' r_1$, we see that r_1 is uniformly distributed over $[N]$ and is independent of $\mathfrak{u} = \mathfrak{g}^r = \mathfrak{g}^{r_0}$. But now the ciphertext perfectly hides m, since $\mathfrak{v} = \mathfrak{g}^{r_0 t} \mathfrak{w}^{(r_0 + N' r_1)s + m}$. □

The Ω-protocol

We are now ready to describe our Ω-protocol Π for proving knowledge of a representation. Our protocol works for any abelian group $\mathbb{H}$ of prime order q. The protocol will prove knowledge of a representation relative to k bases, allowing trapdoor extraction of $\ell \leq k$ of the exponents. In our application to commitments based on Waters signatures, $k = 3$ and $\ell = 1$.

In addition to a description of $\mathbb{H}$, the system parameters for Π consist of the RSA modulus N (as described in §5.4.3), along with the system parameter $\mathfrak{g} \in (\mathbb{Z}^*_{N^2})^{2N}$ used for Projective Paillier encryption. Recall that $\mathfrak{w} := [1 + N]_{N^2} \in \mathbb{Z}^*_{N^2}$ is the special group element of order N. In addition, the system parameters include random group elements

$$\mathfrak{g}_0, \mathfrak{g}_1, \ldots, \mathfrak{g}_\ell \in (\mathbb{Z}^*_N)^2.$$

We need two more parameters, B_{c} and B_{p}. Here, B_{c} is a bound on the size of the challenge space, and B_{p} is a "padding bound." The property required is that $1/B_{\mathrm{c}}$ and $1/B_{\mathrm{p}}$ are negligible.

In addition, we require that

$$B_{\mathrm{c}} B_{\mathrm{p}} q \leq N/2 \quad \text{and} \quad B_{\mathrm{c}} \leq \min\{q, P', Q'\}. \tag{5.2}$$

The reference parameter generation algorithm for Π is the key generation algorithm for the Projective Paillier encryption scheme. A reference parameter is a public key $\mathfrak{h} \in (\mathbb{Z}_{N^2}^*)^{2N}$ for the encryption scheme, and the corresponding trapdoor is the secret key $t \in [N/4]$, where $\mathfrak{g}^t = \mathfrak{h}$.

Now let $\gamma_1, \ldots, \gamma_k, \gamma \in \mathbb{H}$ and $w_1, \ldots, w_k \in [q]$, where

$$\gamma_1^{w_1} \cdots \gamma_k^{w_k} = \gamma. \tag{5.3}$$

The common inputs to the prover and verifier are the group elements $\gamma_1, \ldots, \gamma_k, \gamma$. The prover also gets the tuple $(w_1, \ldots, w_k)$ as a witness. Our protocol will prove knowledge of values $w_1, \ldots, w_k \in [q]$ satisfying (5.3), with the values $w_1, \ldots, w_\ell$ being trapdoor extractable. More precisely, our protocol will satisfy the partial trapdoor soundness property relative to the function $f(w_1, \ldots, w_\ell) := (w_1, \ldots, w_k)$.

The protocol Π runs as follows:

1. The prover computes

   $r_1, \ldots, r_\ell, s \xleftarrow{\$} [N/4]$

   for $i \leftarrow 1$ to ℓ: $\mathfrak{u}_i \leftarrow \mathfrak{g}^{r_i}$, $\mathfrak{v}_i \leftarrow \mathfrak{h}^{r_i} \mathfrak{w}_i^{w_i}$

   $\mathsf{h} \leftarrow \mathsf{g}_0^s \mathsf{g}_1^{w_1} \cdots \mathsf{g}_k^{w_\ell}$

   $\bar{r}_1, \ldots, \bar{r}_\ell, \bar{s} \xleftarrow{\$} [B_{\mathrm{p}} B_{\mathrm{c}} N/4] \setminus [B_{\mathrm{c}} N/4]$

   $\bar{w}_1, \ldots, \bar{w}_k \xleftarrow{\$} [B_{\mathrm{p}} B_{\mathrm{c}} q] \setminus [B_{\mathrm{c}} q]$

   $\bar{\gamma} \leftarrow \gamma_1^{\bar{w}_1} \cdots \gamma_k^{\bar{w}_k}$

   for $i \leftarrow 1$ to ℓ: $\bar{\mathfrak{u}}_i \leftarrow \mathfrak{g}^{\bar{r}_i}$, $\bar{\mathfrak{v}}_i \leftarrow \mathfrak{h}^{\bar{r}_i} \mathfrak{w}_i^{\bar{w}_i}$

   $\bar{\mathsf{h}} \leftarrow \mathsf{g}_0^{\bar{s}} \mathsf{g}_1^{\bar{w}_1} \cdots \mathsf{g}_k^{\bar{w}_\ell}$

   and sends

   $$\{(\mathfrak{u}_i, \mathfrak{v}_i, \bar{\mathfrak{u}}_i, \bar{\mathfrak{v}}_i)\}_{i=1}^{\ell}, \quad \bar{\gamma}, \quad \mathsf{h}, \bar{\mathsf{h}}$$

   to the verifier.

2. The verifier chooses a random challenge $c \in [B_{\mathrm{c}}]$.

3. The prover computes

for $i \leftarrow 1$ to k: $\hat{w}_i \leftarrow \bar{w}_i - cw_i$

for $i \leftarrow 1$ to ℓ: $\hat{r}_i \leftarrow \bar{r}_i - cr_i$

$\hat{s} \leftarrow \bar{s} - cs$

and sends

$$\{\hat{w}_i\}_{i=1}^{k}, \quad \{\hat{r}_i\}_{i=1}^{\ell}, \quad \hat{s}$$

to the verifier.

4. The verifier checks that

$$\hat{w}_i \in [N/2] \quad \text{for } i = 1, \ldots, \ell,$$

and verifies the following relations:

$$\bar{\gamma} = \gamma^c \cdot \prod_{i=1}^{k} \gamma_i^{\hat{w}_i}, \qquad \bar{\mathsf{h}} = \mathsf{h}^c \cdot \mathsf{g}_0^{\hat{s}} \prod_{i=1}^{\ell} \mathsf{g}_i^{\hat{w}_i},$$

$$\bar{\mathfrak{u}}_i = \mathfrak{u}_i^c \cdot \mathfrak{g}^{\hat{r}_i} \quad (i = 1, \ldots, \ell), \qquad \bar{\mathfrak{v}}_i = \mathfrak{v}_i^c \cdot \mathfrak{h}^{\hat{r}_i} \mathfrak{w}^{\hat{w}_i} \quad (i = 1, \ldots, \ell).$$

Analysis

In the attack game for partial trapdoor soundness, we assume an adversary has produced two accepting conversations

$$\{(\mathfrak{u}_i, \mathfrak{v}_i, \bar{\mathfrak{u}}_i, \bar{\mathfrak{v}}_i)\}_{i=1}^{\ell}, \bar{\gamma}, \mathsf{h}, \bar{\mathsf{h}}, \quad c, \quad \{\hat{w}_i\}_{i=1}^{k}, \{\hat{r}_i\}_{i=1}^{\ell}, \hat{s},$$
$$\{(\mathfrak{u}_i, \mathfrak{v}_i, \bar{\mathfrak{u}}_i, \bar{\mathfrak{v}}_i)\}_{i=1}^{\ell}, \bar{\gamma}, \mathsf{h}, \bar{\mathsf{h}}, \quad c', \quad \{\hat{w}'_i\}_{i=1}^{k}, \{\hat{r}'_i\}_{i=1}^{\ell}, \hat{s}',$$

where $c \neq c'$. Both conversations are fed into the rewinding extractor, while the first conversation, together with the trapdoor t, is fed into the trapdoor extractor. Let us define

$$\Delta c := c' - c, \qquad \Delta w_i := \hat{w}_i - \hat{w}'_i \ (i = 1, \ldots, k),$$

$$\Delta r_i := \hat{r}_i - \hat{r}'_i \ (i = 1, \ldots, \ell), \qquad \Delta s := \hat{s} - \hat{s}'.$$

From the verification relations, we have

$$|\Delta w_i| < N/2 \quad (i = 1, \ldots, \ell) \tag{5.4}$$

$$\gamma^{\Delta c} = \prod_{i=1}^{k} \gamma_i^{\Delta w_i}, \tag{5.5}$$

$$\mathfrak{h}^{\Delta c} = \mathfrak{g}_0^{\Delta s} \prod_{i=1}^{\ell} \mathfrak{g}_i^{\Delta w_i}, \tag{5.6}$$

$$\mathfrak{u}_i^{\Delta c} = \mathfrak{g}^{\Delta r_i} \quad (i = 1, \ldots, \ell), \tag{5.7}$$

$$\mathfrak{v}_i^{\Delta c} = \mathfrak{h}^{\Delta r_i} \mathfrak{w}^{\Delta w_i} \quad (i = 1, \ldots, \ell). \tag{5.8}$$

We also know that $|\Delta c| < B_{\mathrm{c}}$.

The rewinding extractor. Give two accepting conversations as above, since $0 < |\Delta c| < q$, the rewinding extractor may compute

$$w_i \leftarrow (\Delta w_i / \Delta c) \bmod q \quad (i = 1, \ldots, k).$$

From (5.5), it is clear that $(w_1, \ldots, w_k)$ is indeed a valid witness, *i.e.*, $\gamma = \prod_{i=1}^{k} \gamma_i^{w_i}$.

The trapdoor extractor. Given an accepting conversation as above, together with the trapdoor t, the trapdoor extractor runs as follows:

for $i \leftarrow 1$ to ℓ do
  $\mathfrak{w}'_i \leftarrow (\mathfrak{v}_i / \mathfrak{u}_i^t)^2$
  if $\mathfrak{w}'_i = [1 + N z_i]_{N^2}$ for some $z_i \in [N]$ then
    $z_i \leftarrow (z_i / 2) \bmod N$
    if $z_i \geq N/2$ then $z_i \leftarrow z_i - N$ // *compute a "balanced" remainder*
    $w_i \leftarrow z_i \bmod q$
  else
    $w_i \leftarrow 0$ // *this is an error*

Lemma 5.4. *With the given rewinding and trapdoor extractors, under the Strong RSA assumption, protocol* Π *satisfies the trapdoor* f*-extractable property, where* $f(w_1, \ldots, w_k) := (w_1, \ldots, w_\ell)$.

Proof. This follows the same line of reasoning as in Camenisch and Shoup [17]. Given two valid conversation as above, as we already argued, the rewinding extractor always produces a valid

witness $(w_1, \ldots, w_k)$, where

$$w_i := (\Delta w_i / \Delta c) \bmod q \quad (i = 1, \ldots, k).$$

We want to show that the trapdoor extractor outputs $(w_1, \ldots, w_\ell)$ with overwhelming probability. From the identity (5.6), with overwhelming probability, we have $\Delta w_i / \Delta c \in \mathbb{Z}$ for each $i = 1, \ldots, \ell$. This is where we use the Strong RSA assumption (see Lemma 5.2). Moreover, from (5.4), we have $|\Delta w_i / \Delta c| < N/2$ for each $i = 1, \ldots, \ell$. From (5.7) and (5.8), and the relation $\mathfrak{h} = \mathfrak{g}^t$, one obtains

$$\Big(\frac{\mathfrak{v}_i / \mathfrak{u}_i^t}{\mathfrak{w}^{\Delta w_i / \Delta c}}\Big)^{\Delta c} = 1 \quad (i = 1, \ldots, \ell).$$

Now, the group $\mathbb{Z}_{N^2}^*$ has exponent $2NN'$, and since $|\Delta c| < B_{\mathrm{c}} \leq \min\{P', Q'\}$, it follows that $\gcd(\Delta c, 2NN') \in \{1, 2\}$, which implies that

$$\Big(\frac{\mathfrak{v}_i / \mathfrak{u}_i^t}{\mathfrak{w}^{\Delta w_i / \Delta c}}\Big)^{2} = 1 \quad (i = 1, \ldots, \ell).$$

This, together with the fact that $|\Delta w_i / \Delta c| < N/2$, implies that the output of the trapdoor extractor agrees with the output of the rewinding extractor. □

The zero-knowledge simulator. Given a challenge c, the simulator runs as follows:

$r_1, \ldots, r_\ell, s \xleftarrow{\$} [N/4]$
for $i \leftarrow 1$ to ℓ: $\mathfrak{u}_i \leftarrow \mathfrak{g}^{r_i}$, $\mathfrak{v}_i \leftarrow \mathfrak{h}^{r_i}$
$\mathsf{h} \leftarrow \mathsf{g}_0^s$
$\hat{r}_1, \ldots, \hat{r}_\ell, \hat{s} \xleftarrow{\$} [B_{\mathrm{p}} B_{\mathrm{c}} N/4]$
$\hat{w}_1, \ldots, \hat{w}_k \xleftarrow{\$} [B_{\mathrm{p}} B_{\mathrm{c}} q]$
$\bar{\gamma} \leftarrow \gamma^c \cdot \prod_{i=1}^k \gamma_i^{\hat{w}_i}$
$\bar{\mathsf{h}} \leftarrow \mathsf{h}^c \cdot \mathsf{g}_0^{\hat{s}} \prod_{i=1}^\ell \mathsf{g}_i^{\hat{w}_i}$
for $i \leftarrow 1$ to ℓ do
    $\bar{\mathfrak{u}}_i \leftarrow \mathfrak{u}_i^c \cdot \mathfrak{g}^{\hat{r}_i}$, $\bar{\mathfrak{v}}_i \leftarrow \mathfrak{v}_i^c \cdot \mathfrak{h}^{\hat{r}_i} \mathfrak{w}^{\hat{w}_i}$

The first flow of the simulated conversation is

$$\{(\mathfrak{u}_i, \mathfrak{v}_i, \bar{\mathfrak{u}}_i, \bar{\mathfrak{v}}_i)\}_{i=1}^\ell, \quad \bar{\gamma}, \quad \mathsf{h}, \bar{\mathsf{h}},$$

while the third flow is

$$\{\hat{w}_i\}_{i=1}^{k}, \quad \{\hat{r}_i\}_{i=1}^{\ell}, \quad \hat{s}.$$

Lemma 5.5. *With the given simulator, under the DCR assumption, protocol* Π *satisfies the special HVZK property.*

Proof. This follows from the semantic security of Projective Paillier, and standard statistical distance arguments. □

Dense reference parameters. The set of reference parameters is suitably dense, in the sense of Section 2.3.2. Namely, under the QR and DCR assumptions, a randomly generated public key $\mathfrak{h}$ is computationally indistinguishable from a random element of the subgroup J_N of $\mathbb{Z}_{N^2}^*$; this follows from Lemma 5.1. Moreover, the set J_N is efficiently recognizable (just evaluate a Jacobi symbol) and the uniform distribution on J_N is efficiently samplable; indeed, one may generate a random element of J_N as follows:

$b \xleftarrow{\$} \{0,1\}$, $\mathfrak{r} \xleftarrow{\$} \mathbb{Z}_{N^2}^*$
output $(-1)^b \mathfrak{r}^2$

5.5 Reducing Round Complexity with Random Oracles

While our constructions for GUC zero-knowledge and commitments are efficient in both computational and communication complexity, and the constant round complexity of 6 messages is reasonable, it would be nice improve the round complexity, and possibly weaken the data erasure assumption. In this section we address the question if such improvements are possible in the random oracle (RO) model (see Section 2.2.4). We first remark that even the RO model, without any additional setup, does not suffice for realizing GUC commitments or zero-knowledge (as per the impossibility result of Section 4.2). Surprisingly, we can still obtain some additional efficiency benefits by combining the ACRS and RO models. Ideally, we would like to obtain 2-message protocols in order to match the lower bound of Section 5.1. Indeed, we now show that it is possible to achieve optimal 2-message ZK and commitment protocols in the GUC setting by combining both the ACRS and RO setups. (Recall that the lower bound indeed extends to include this case, so our protocols are round-optimal even for the combined ACRS and RO model.)

We achieve our goal by simply applying the Fiat-Shamir heuristic [49] to our efficient zero-knowledge and commitment protocols, replacing the first three and last three messages of each protocol with a single message.

More specifically, given a random oracle H, we transform our zero-knowledge protocol as follows. The first message is sent by the verifier, who generates the values κ_1 from Step 1 of the protocol and (d_1, ρ_1) from Step 3 of the protocol as usual, and then sends (κ_1, d_1, ρ_1) to the prover. Both the prover and the verifier will set $\rho_2 \leftarrow H(\kappa_1)$ and compute ρ accordingly. The second message is then sent by the prover, who computes κ' as in Step 5, sets the challenge $c \leftarrow H(\kappa')$, and computes the (d', a, z) as in Step 7. The prover then sends (κ', d', a, z) to the verifier, and verification is done in the obvious manner. The changes to the commitment protocol are entirely analogous. Also, note that the only erasure required by our protocols now occurs entirely during a single local computation, without delay – namely, during the computation of the second message, an entire run of protocol Π is computed and the randomness used to generate that run of Π is then immediately erased.

The proof of security for the modified protocols is virtually unaltered by the use of the Fiat-Shamir heuristic, except that we must make slight alterations in our reduction techniques. In particular, observe that the GUC simulator $\mathcal{S}$ uses identical simulation strategies, and *does not* need to have access to a transcript of oracle queries, nor does it require the ability to "program" oracle responses. However, we will still need to use these properties of the random oracle model – but only in the *proof of security* (to prove that the environment cannot tell apart the real and the ideal worlds).

Theorem 5.10. *The protocols obtained by applying the Fiat-Shamir heuristic to protocols* DOZK *and* DOC *in the combined $\bar{\mathcal{G}}_{acrs}$ and $\bar{\mathcal{G}}_{ro}$ shared hybrid model yield* 2*-message GUC-secure realizations of $\mathcal{F}_{zk}$ and $\mathcal{F}_{com}$ (respectively), in the adaptive corruption model with erasure of ephemeral data.*

Proof. Since the proof of security is only minimally impacted by the use of the Fiat-Shamir heuristic, we will only sketch the changes here.

For both protocols, we observe that the simulation strategies remain entirely unchanged (except that now, we must view the messages in Step 2 and Step 6 of the protocols as originating

from the random oracle instead of the verifier/recipient, and we need not concern ourselves with adaptive corruptions during the protocol except between Step 4 and Step 5). However, make the following alterations to the proof of simulation indistinguishability.

First, for both protocols, we will alter the reduction which proves that I_3 is indistinguishable from I_2. Namely, instead of the rewinding technique of Lemma 4.1, we will employ the technique used for proving the security of the Fiat-Shamir heuristic as applied to signature schemes [49]. Namely, it is possible to rewind the experiment and "reprogram" the oracle query, after having seen a single valid opening of the first flow (input to the oracle). After such a rewinding, the adversary can be expected to *reuse the same "first flow"* (with reference to the original, untransformed protocol) and still give a valid opening of the commitment with non-negligible probability. Indeed, this is how signature forgeries are obtained when proving the security of Fiat-Shamir based signature schemes by reduction forgery (see the "forking lemma" in [77]). In our case, this means it is still possible to reduce the task of distinguishing I_3 from I_2 to the HVZK property of Π – since we may "rig" the output of the first flow (so that it yields the challenge ρ sent to the reduction) by seeing the adversary open the commitment in the "first flow" once, then rewinding it and programming the corresponding random oracle query appropriately.

The remainder of the proof proceeds identically for zero-knowledge protocol DOZK. However, we still need to make one further modification to the proof in the case of the commitment protocol DOC. Namely, the reduction to partial trapdoor soundness of the Ω-protocol (unlike ordinary trapdoor soundness) must output two accepting transcripts of the Ω-protocol that share the same first flow. Here again, we may apply the same reduction technique used to prove the security of signatures based on the Fiat-Shamir heuristic. With non-negligible probability, this allows us to rewind the adversary to the point where it queried the oracle for the commitment to the "first flow" of Π, and reprogram it so that we learn a second valid response for a different challenge (but still with the same "first flow"). Thus, having obtained two valid transcripts with the same first flow via rewinding the adversary, the reduction can proceed as before.

□

We stress that since the GUC modeling of a random oracle (accurately) allows the oracle to be accessed directly by all entities – including the environment – the aforementioned observation

that $\mathcal{S}$ itself does not require a transcript of all oracle queries, nor the ability to program oracle responses, is *crucial* for deniability. It was already observed in [73] that deniable zero-knowledge simulators must not program oracle queries. However, we observe that even using a "non-programmable random oracle" for the simulator is still not sufficient to ensure truly deniable zero-knowledge. In particular, if the modeling allows the simulator to observe interactions with the random oracle (even without altering any responses to oracle queries), this can lead to attacks on deniability. In fact, there is a very practical attack stemming from precisely this issue that will break the deniability of the protocols proposed in [73] (see Section 3.2.1). Our GUC security modeling precludes the possibility of any such attacks.[7]

Of course, unlike the model of [73], we superimpose the ACRS model on the RO model, providing all parties with implicit secret keys. This bears a strong resemblance to the model of [60], which employs the following intuitive approach to provide deniability for the prover P: instead proving the statement, P will prove "either the statement is true, or I know the verifier's secret key". Indeed, our approach is quite similar in spirit. However, we achieve a much stronger notion of deniability than that of [60]. Our zero-knowledge protocols are the first constant round protocols to simultaneously achieve straight-line extractability (required for concurrent composability) and deniability against an adversary who can perform adaptive corruptions.

[7] Similarly, the modeling of [59] also rules out such attacks. However, their protocols make use of special hardware based "signature cards" and require more than 2 rounds. They also do not consider the issue of adaptive corruptions at all.

Conclusion

In this thesis, we introduced the Generalized Universal Composability (GUC) framework for modeling the security requirements of network protocols. Along with the GUC framework, we also introduced the Augmented Common Reference String (ACRS) setup model, which facilitates the implementation of secure network protocols. As one would naturally anticipate when any novel security models are introduced, our models suggest many potential avenues to explore for future advancements and refinements. In our concluding remarks, we discuss some of the many challenges that still lie ahead.

Future directions for the GUC framework.

At its core, the GUC framework aims at enabling the realization of ideal cryptographic tasks without any loss of the innate security properties of the task (for instance, loss of deniability). While we show a general feasibility result for realizing a very large class of tasks, it is important to understand that there are still some limitations inherent within the class of tasks that we consider in this work.

For instance, we only concern ourselves with realizing tasks that allow the adversary to arbitrarily schedule the delivery of outputs (since, after all, most communications networks do not provide timeliness or delivery guarantees). Yet, there are some circumstances where such behavior is undesirable in the extreme. The prototypical example arises in the context of contract signing: two parties who each wish to exchange a digital signature on a contract if and only if the other party also signs. In such a scenario, it would be *unfair* if one party were to receive the protocol output (a signed contract) while the other party fails to receive the protocol output – no matter how "secure" the protocol for computing the signed contract might be. Therefore, an important direction for the GUC security framework is the addition of *fairness* requirements. Following in the footsteps of works such as [15, 1, 4], one possible approach for supporting such fairness requirements is to employ the services of an "optimistically offline" trusted third party. Under normal circumstances, when both parties manage to successfully complete a protocol, the trusted third party is completely uninvolved. However, in the event that a communications failure (malicious or otherwise) prevents one of the parties from learning the protocol output,

the trusted third party can step in to help resolve any unfair outcomes. Since unfair outcomes are likely to be rare (barring intentional "denial of service" attacks), the nature of the burden placed upon the trusted third party is only slightly greater than that of our ACRS setup entity (see Section 2.2.2). Thus, if one is already using an ACRS setup entity, it may make sense to combine the two roles. Due to subtle complications, a direct application of techniques from previous works achieving fairness via optimistically offline trusted parties will not suffice in our setting. Therefore, secure realization of tasks requiring fairness in the GUC framework remains a challenging open problem.

In a similar vein, some tasks require security guarantees that can only be provided with special corruption models. A prime example of such a task is electronic voting, which requires a special security property often referred to as *incoercibility*. Incoercibility is closely related to deniability. In the context of electronic voting, incoercibility requires that all parties running the voting protocol must be incapable of producing any evidence of their voting strategy, even when they willingly deviate from the protocol specification in order to do so. More generally, incoercibility requires that even a party who is willing to deviate from the protocol *cannot* provide convincing evidence that a particular action was performed within a real protocol session (as opposed to being simulated during a run of the ideal ideal protocol). This requirement stems from concerns that an otherwise honest party might be "coerced" to deviate from the protocol (either by threat of force, blackmail, or even bribery). On the other hand, the corruption models used in this work assume that honest parties will never deviate from the protocol. One approach to handling such security requirements is to introduce a new kind of corruption, referred to as a *coercion*, which specifies the ways in which an honest (but coerced) party might deviate from the protocol specification. In general, security against a coercion model allowing for arbitrary protocol deviations seems impossible to achieve without the use of some trusted hardware or an interactive trusted third party who actively participates in the protocol. Future directions for research related to incoercibility include defining appropriate coercion models, properly classifying the types of trusted parties and hardware that can be used to achieve incoercibility in said models, and designing practical protocols for important tasks that require incoercibility (such as voting).

At the other end of the security spectrum, it is also interesting to consider the most nat-

ural ways to model protocols that are inherently undeniable (perhaps even intentionally non-repudiable). The prototypical example of an undeniable protocol is that of ordinary digital signatures. While Canetti [21] showed that ordinary signature schemes are modeled by $\mathcal{F}_{auth}$ in the original UC framework, we have argued that such modeling is undesirable. In particular, the $\mathcal{F}_{auth}$ task itself is inherently deniable, and does not provide protocol designers making use of $\mathcal{F}_{auth}$ with any indication that they might be leaving behind a long-lived (and non-repudiable) digital signature of all authenticated messages sent by their protocol. Instead, we prefer to use the more realistic modeling enabled by the GUC framework, which *correctly* rules out digital signatures as a viable implementation of the *deniable* authentication functionality $\mathcal{F}_{auth}$. However, this leaves us with the question of how to naturally define the corresponding task of *undeniable* authentication that can be realized using digital signatures. The definitional issues for such undeniable tasks entail many subtleties (for example, consider the *mostly* deniable task $\mathcal{F}_{keia}$ introduced in Section 3.3). Thus, the question of how best to model the security provided by digital signatures (and other non-repudiable cryptographic protocols) in the GUC framework must still be addressed.

Finally, we note that the GUC framework itself is still merely an abstraction of realistic network behavior, and is consequently limited. Although the GUC model aims to provide a highly realistic security notion, there remain many realistic phenomena that are not yet incorporated into the framework. For instance, *side-channel attacks*, wherein an attacker is able to obtain secret information via external means (rather than the communications network), are not modeled. Similarly, the bandwidth limitations of the adversary and the underlying communications network are not modeled. Indeed, these issues are related, since bandwidth limitations can be used in order to minimize the impact of side-channel attacks. Such an approach is suggested by the Bounded Retrieval Model (BRM) techniques introduced by [36]. Integrating the BRM into the GUC framework can potentially yield protocols that remain GUC secure even in the presence of (bandwidth-limited) side-channel attacks, and even after the party running the protocols has suffered a security breach. Similar benefits might potentially be obtained by introducing other realistic phenomena into the GUC framework. Explorations along such lines are a promising approach for the development of more practical security notions, and possibly even more efficient protocols.

Future directions for setup models.

There is also much room for additional research within the confines of the present formulation of GUC security. For instance, the minimal setup assumptions that we give for realizing authenticated channels require both a trusted "bulletin board" (for publishing public keys), and a trusted ACRS setup entity. Alternatively, a full-fledged trusted certification authority – which verifies that all parties who have public keys also possess corresponding secret keys – can be used. It is conceivable that the same goal can be achieved using even more reasonable setup assumptions than these.

On the flip side, perhaps the use of even *stronger* setup assumptions is warranted in order to achieve more efficient implementations. For example, there is the "signature card" setup model of [59], and the more general tamper-proof hardware model of [64] (which can be made more realistic via our notion of shared functionalities). Much as our use of the random oracle model in Section 5.5 enabled us to improve the efficiency of the protocols we propose there, the use of stronger setup assumptions may enable other efficiency improvements. Provided that the cost of implementing such setup models is outweighed by the resulting efficiency improvements, this avenue of investigation may lead to many practical GUC-secure protocols.

Another reason to use stronger setup assumptions is our impossibility result for (deniable) authentication in the PKI model, given in Section 3.2.2. While a Symmetric Key Infrastructure (SKI) is much more costly to implement directly than a PKI, our result in Section 3.2.4 shows that SKI setup supports an extremely efficient (and completely deniable) authentication protocol. While the approach of using a one-time "mostly deniable" key exchange (formally defined using $\mathcal{F}_{keia}$ in Section 3.3) to implement the SKI via a PKI setup can considerably reduce the cost, it also introduces a brief window of opportunity for the adversary to potentially break the deniability property of the SKI setup phase. Ideally, a compromise might be found using a setup assumption that is more practical than a direct implementation of SKI, and still does not require any window of vulnerability where potentially undeniable communication occurs.

Of course, another important goal is to minimize the trust that parties are required to place in the setup entity. For example, all the protocols we propose in this work have *forward security* with respect to eventual corruption of the trusted setup entity (*i.e.*, the security of *past* protocol executions is not harmed if the trusted setup entity is compromised). Further minimization

of the trust requirements for the setup entity is an important avenue for future investigation. Alternatively, we can minimize the trust placed in any *single* setup entity by distributing trust assumptions among several different entities. A practical approach to implementing such distribution of trust for the ACRS setup is described in the remarks at the end of Section 2.2.2.

We also note that there are useful features of our setup assumptions that can be further exploited for practical benefit. In particular, since the ACRS setup entity already possesses "trapdoors" for all parties in the system, it naturally lends itself to the task of *key escrow*. In a key escrow system, parties agree to share their secret keys with a trusted third party, known as an escrow agent, with the understanding that the escrow agent will only release their secret keys to authorized entities (*e.g.*, to a law enforcement officer with a wiretap warrant). All players participating in the system are required to share their actual secret keys with the escrow agent, and typically this rule is enforced cryptographically. In our ACRS model protocols, the individualized trapdoors provided to parties by the ACRS setup entity can *only* be used to violate *their own* security (as is necessary for the purposes of conducting attack simulations). Therefore, providing a trapdoor for party P to a law enforcement agent (who has a warrant) only enables the agent to violate the security of P, and no one else. Additionally, by encoding time periods into the party identities (which are each associated with an individualized trapdoor) it becomes possible for the judge to issue warrants that automatically expire after a fixed amount of time. The usefulness of the ACRS model in the context of key escrow suggests the development of formal models for key escrow systems in the GUC framework.

Future directions for secure protocols.

While Chapter 5 provides some examples of efficient GUC-secure protocols for commitments and zero-knowledge proofs, there are many other useful cryptographic tasks for which we lack practical GUC-secure implementations. Indeed, here we have only scratched the surface of what is possible and open questions abound. Even for the case of zero-knowledge and commitment schemes, our efficient protocols require erasures, leaving open the problem of finding efficient protocols for these tasks that do not require any erasure. Furthermore, our round-optimal constructions for these tasks required the use of the random oracle model, and achieving round-optimality without random oracles also remains an open problem.

Similarly, there is much room for progress in the area of deniable authentication. Although we strongly conjecture that forward secure deniable authentication (and key exchange) protocols in the PKI model are not possible even with erasures, the existence of erasure-based protocols is not ruled out by the result of Section 3.2.2. Indeed, one possible method for achieving forward secure deniable authentication might be to use a "key evolving" scheme, where the secret keys of the parties are continually being updated and erased.

Finally, we note that perhaps the most important and challenging practical issue that has yet be overcome regards the efficient implementation of private channels. In particular, the use of non-committing encryption for privacy purposes seems unavoidable in the setting of adaptive corruptions. Unfortunately, all currently known non-committing encryption schemes are highly inefficient. Finding techniques to *efficiently* implement GUC-secure channels that are both authenticated *and* private represents the last significant hurdle to many practical applications of GUC-security.

Bibliography

[1] N. Asokan, M. Schunter, and M. Waidner. Optimistic Protocols for Fair Exchange. In *Proc. of ACM CCS*, pp. 6–17, 1997.

[2] N. Asokan, V. Shoup, and M. Waidner. Optimistic Fair Exchange of Digital Signatures. Extended abstract in *Proc. of Eurocrypt*, 1998. In *IEEE J. Selected Areas in Communications*, 18(4), pp. 593–610, 2000.

[3] G. Ateniese and B. de Medeiros. Identity-based Chameleon Hash and Applications. In *Proc. of Financial Cryptography*, 2004.

[4] F. Bao, R. H. Deng, and W. Mao. Efficient and Practical Fair Exchange Protocols with Off-line TTP. In *Proc. of IEEE Symp. Security and Privacy*, pp. 77–85, 1998.

[5] B. Barak, R. Canetti, Y. Lindell, R. Pass, and T. Rabin. Secure Computation Without Authentication. In *Proc. of Crypto*, pp. 361–377, 2005.

[6] B. Barak, R. Canetti, J. Nielsen, and R. Pass. Universally Composable Protocols with Relaxed Set-up Assumptions. In *Proc. of FOCS*, 2004.

[7] B. Barak and Y. Lindell. Strict Polynomial-time Simulation and Extraction. In *SIAM J. Comput.*, 33(4), pp. 783–818, 2004.

[8] B. Barak and A. Sahai. How To Play Almost Any Mental Game Over the Net - Concurrent Composition via Super-Polynomial Simulation. In *Proc. of FOCS*, 2005.

[9] D. Beaver. Secure Multi-Party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. In *J. Cryptology*, vol 4., pp. 75–122, 1991.

[10] M. Bellare and A. Palacio. GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In *Proc. of Crypto*, 2002.

[11] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proc. of ACM CCS*, pp. 62–73, 1993.

[12] M. Blum. How to Prove a Theorem So No One Else Can Claim It. In *Proc. of the International Congress of Mathematicians*, 1986.

[13] D. Boneh, and M. Franklin. Identity Based Encryption from the Weil Pairing. In *Proc. of Crypto*, 2001.

[14] N. Borisov, I. Goldberg, and E. Brewer. Off-The Record Communication, or, Why Not to Use PGP. In *Proc. of WPES*, pp. 77–84, 2004.

[15] H. Burk and A. Pfitzmann. Value Exchange Systems Enabling Security and Unobservability. In *Computers and Security*, vol. 9, pp. 715–721, 1990.

[16] J. Camenisch and A. Lysyanskaya. A Signature Scheme with Efficient Protocols. In *Conference on Security in Communication Networks (SCN)*, 2002.

[17] J. Camenisch and V. Shoup. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In *Proc. of Crypto*, 2003.

[18] R. Canetti. Security and Composition of Multi-Party Cryptographic Protocols. In *J. of Cryptology*, vol. 13, no. 1, winter 2000.

[19] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proc. of FOCS*, pp. 136–145, 2001.

[20] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Cryptology ePrint Archive, Report 2000/067, **revised edition from Dec. 2005**. Available at `http://eprint.iacr.org/2000/067/`.

[21] R. Canetti. Universally Composable Signature, Certification, and Authentication. In *Proc. of CSFW*, p. 219, 2004.

[22] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively Secure Computation. In *Proc. of STOC*, pp. 639–648, 1996.

[23] R. Canetti and M. Fischlin. Universally Composable Commitments. In *Proc. of Crypto*, pp. 19–40, 2001.

[24] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable Zero-Knowledge. In *Proc. of STOC*, pp. 235–244, 2000.

[25] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. In *Proc. of STOC*, pp. 209–218, 1998.

[26] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally Composable Password-Based Key Exchange. In *Proc. of Eurocrypt*, pp. 404–421, 2005.

[27] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires (Almost) Logarithmically Many Rounds. In *SIAM J. Comput.*, 32(1), pp. 1–47, 2002.

[28] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In *Proc. of Eurocrypt*, 2002.

[29] R. Canetti, E. Kushilevitz, and Y. Lindell. On the Limitations of Universally Composable Two-Party Computation Without Set-Up Assumptions. In *Proc. of Eurocrypt*, pp. 68–86, 2003.

[30] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally Composable Two-Party and Multi-Party Secure Computation. In *Proc. of STOC*, pp. 494–503, 2002.

[31] R. Canetti and T. Rabin. Universal Composition with Joint State. In *Proc. of Crypto 2003*, pp. 265–281, 2003.

[32] R. Cramer, I. Damgard, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Proc. of Crypto*, pp. 174–187, 1994.

[33] R. Cramer and V. Shoup. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public Key Encryption. In *Proc. of Eurocrypt*, 2002.

[34] I. Damgard and J. Nielsen. Improved Non-Committing Encryption Schemes based on a General Complexity Assumption. In *Proc. of Crypto*, 2000.

[35] I. Damgard and J. Nielsen. Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor. In *Proc. of Crypto*, pp. 581–596, 2002.

[36] G. Di Crescenzo, R. Lipton, and S. Walfish. Perfectly Secure Password Protocols in the Bounded Retrieval Model. In *Proc. of TCC*, 2006.

[37] M. Di Raimondo and R. Gennaro. New Approaches for Deniable Authentication. In *Proc. of ACM CCS*, pp. 112–121, 2005.

[38] M. Di Raimondo, R. Gennaro, and H. Krawczyk. Secure Off-the-Record Messaging. In *Proc. of WPES*, 2005.

[39] M. Di Raimondo, R. Gennaro, and H. Krawczyk. Deniable Authentication and Key Exchange. In *Proc. of ACM CCS*, pp. 400–409, 2006.

[40] W. Diffie and M. Hellman. New Directions in Cryptography. In *IEEE Trans. on Information Theory*, vol. 22, no. 6, pp 644–654, 1976.

[41] Y. Dodis and S. Micali. Parallel Reducibility for Information-Theoretically Secure Computation. In *Proc. of Crypto*, pp. 74–92, 2000.

[42] D. Dolev, C. Dwork, and M. Naor. Non-malleable Cryptography. In *SIAM J. Comput.*, 30(2), pp. 391–436, 2000.

[43] C. Dwork and M. Naor. Zaps and their Applications. In *Proc. of FOCS*, 2000.

[44] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-knowledge. In *Proc. of STOC*, 1998.

[45] C. Dwork and A. Sahai. Concurrent Zero-Knowledge: Reducing the Need for Timing Constraints. In *Proc. Crypto*, pp. 442–457, 1998.

[46] U. Feige. Alternative Models for Zero Knowledge Interactive Proofs. Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel, 1990.

[47] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Based on a Single Random String. In *Proc. of FOCS*, 1990.

[48] U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In *Proc. of Crypto*, pp. 526–545, 1989.

[49] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Proc. of Crypto*, pp. 181–187, 1987.

[50] J. Garay, P. MacKenzie, and K. Yang. Strengthening Zero-Knowledge Protocols Using Signatures. In *Proc. of Eurocrypt*, pp. 177–194, 2003.

[51] O. Goldreich, S. Micali, and A. Wigderson. How to Solve Any Protocol Problem. In *Proc.of STOC*, 1987.

[52] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing But their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. In *J. of the ACM*, 38(1), pp. 691–729, 1991.

[53] S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *Proc. of Crypto*, 1990.

[54] S. Goldwasser and S. Micali. Probabilistic Encryption. In *J. of Computer and Systems Sci.*, 28(2), pp. 270–299, 1984.

[55] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. In *SIAM J. Comput.*, 18(1), pp. 186–208, 1989.

[56] J. Groth and R. Ostrovsky. Cryptography in the Multi-String Model. In *Proc. of Crypto*, pp. 323–341, 2007.

[57] J. Hastad, R. Impagliazzo, L. Levin, and M. Luby. A Pseudorandom Generator from Any One-way Function. In *SIAM J. Comput.*, 28(4), pp. 1364–1396, 1999.

[58] J. Herzog, M. Liskov, and S. Micali. Plaintext Awareness via Key Registration. In *Proc. of Crypto*, 2003.

[59] D. Hofheinz, J. Muller-Quade, and D. Unruh. Universally Composable Zero-Knowledge Arguments and Commitments from Signature Cards. In *Proc. of MoraviaCrypt*, 2005.

[60] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated Verifier Proofs and Their Applications. In *Proc. of Eurocrypt*, 1996.

[61] S. Jiang. Deniable Authentication on the Internet. Available at `http://eprint.iacr.org/2007/082/`.

[62] Y. Kalai, Y. Lindell, and M. Prabhakaran. Concurrent General Composition of Secure Protocols in the Timing Model. In *Proc. of STOC*, 2005.

[63] J. Katz. Efficient Cryptographic Protocols Preventing 'Man-in-the-Middle' Attacks. Ph.D. Thesis, Columbia University, 2002.

[64] J. Katz. Universally Composable Multi-Party Computation Using Tamper-Proof Hardware. In *Proc. of Eurocrypt*, 2007.

[65] J. Katz and R. Ostrovsky. Round-Optimal Secure Two-Party Computation. In *Proc. of Crypto*, 2004.

[66] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. In *Proc. of FOCS*, pp. 394–403, 2003.

[67] Y. Lindell. Bounded-Concurrent Secure Two-Party Computation Without Setup Assumptions. In *Proc. of STOC*, pp. 683–692, 2003.

[68] P. MacKenzie and K. Yang. On Simulation-Sound Trapdoor Commitments. In *Proc. of Eurocrypt*, pp. 382–400, 2004.

[69] S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *Proc. of Crypto*, 1991.

[70] M. Naor. Bit Commitment Using Pseudo-Randomness. In *Proc. of Crypto*, pp. 128–136, 1989.

[71] M. Naor. Deniable Ring Authentication. In *Proc. of Crypto*, pp. 481–498, 2002.

[72] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proc. Eurocrypt*, 1999.

[73] R. Pass. On Deniabililty in the Common Reference String and Random Oracle Model. In *Proc. of Crypto*, pp. 216–337, 2003.

[74] R. Pass. Bounded-Concurrent Secure Multi-Party Computation with a Dishonest Majority. In *Proc. of STOC*, pp. 232–241, 2004.

[75] R. Pass and A. Rosen. Bounded-Concurrent Secure Two-Party Computation in a Constant Number of Rounds. In *Proc. of FOCS*, pp. 404–413, 2003.

[76] B. Pfitzmann and M. Waidner. Composition and Integrity Preservation of Secure Reactive Systems. In *Proc. of ACM CCS*, pp. 245–254, 2000.

[77] D. Pointcheval and J. Stern. Security Proofs for Signature Schemes. In *Proc. of Eurocrypt*, pp. 387–398, 1996.

[78] M. Prabhakaran and A. Sahai. New Notions of Security: Achieving Universal Composability Without Trusted Setup. In *Proc. of STOC*, 2004.

[79] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *Proc. of Eurocrypt*, 1999.

[80] A. Sahai. Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen Ciphertext Security. In *Proc. of FOCS*, pp. 543–553, 1999.

[81] C. Shannon. Communication Theory of Secrecy Systems. In *Bell Systems Technical Journal*, 28(4), pp. 656–715, 1949.

[82] W. Susilo and Y. Mu. Non-Interactive Deniable Ring Authentication. In *Proc. of ICISC*, pp. 386–401, 2003.

[83] W. Susilo and Y. Mu. Deniable Ring Authentication Revisited. In *Proc. of ACNS*, pp. 149–163, 2004.

[84] B. Waters. Efficient Identity-Based Encryption Without Random Oracles. In *Proc. of Eurocrypt*, pp. 114–127, 2005.

[85] A. Yao. How to Generate and Exchange Secrets. In *Proc. of FOCS*, pp. 162–167, 1986.

[86] A. Yao, F. Yao, Y. Zhao, and B. Zhu. Deniable Internet Key-Exchange. Available at http://eprint.iacr.org/2007/191/.

[87] F. Zhang, R. Safavi-Naini, and W. Susilo. ID-Based Chameleon Hashes from Bilinear Pairings. Available at http://eprint.iacr.org/2003/208/.