Universally Composable Security: A New Paradigm for Cryptographic Protocols

Ran Canetti[*]

February 4, 2008

Abstract

We present a general framework for representing cryptographic protocols and analyzing their security. The framework allows specifying the security requirements of practically any cryptographic task in a unified and systematic way. Furthermore, in this framework the security of protocols is preserved under a general protocol composition operation, called universal composition.

The proposed framework with its security-preserving composition operation allow for modular design and analysis of complex cryptographic protocols from relatively simple building blocks. Moreover, within this framework, protocols are guaranteed to maintain their security in any context, even in the presence of an unbounded number of arbitrary protocol instances that run concurrently in an adversarially controlled manner. This is a useful guarantee, that allows arguing about the security of cryptographic protocols in complex and unpredictable environments such as modern communication networks.

Keywords: cryptographic protocols, security analysis, protocol composition, universal composition.

[*]IBM T.J. Watson Research Center, canetti@watson.ibm.com. Supported by NSF CyberTrust Grant #0430450.

Contents

1 Introduction

Rigorously demonstrating that a protocol "does its job securely" is an essential component of cryptographic protocol design. This requires coming up with an appropriate mathematical model for representing protocols, and then formulating, within that model, a *definition of security* that captures the requirements of the task at hand. Once such a definition is in place, we can show that a protocol "does its job securely" by demonstrating that its mathematical representation satisfies the definition of security within the devised mathematical model.

However, coming up with a good mathematical model for representing protocols, and even more so formulating appropriate definitions of security within the devised model, turns out to be a tricky business. The model should be rich enough to represent all realistic adversarial behaviors, and the definition should guarantee that the intuitive notion of security is captured with respect to any adversarial behavior under consideration.

One main aspect of the security of cryptographic protocols is their sensitivity to the execution environment. Indeed, the behavior of cryptographic protocols is intimately tied to the specific execution environment, and in particular to the other protocols that are running in the same system or network. Consequently it is important, when assessing the security of a cryptographic protocol, to consider the behavior of the protocol in conjunction with other protocols, or more generally when the protocol is used as a component within a larger system. This aspect is the focal point of this work.

Initially, definitions of security for cryptographic tasks considered only a single execution of the analyzed protocol, without giving much attention to more complex execution environments. This is indeed a good choice for first-cut definitions of security. In particular, it allows for relatively concise and intuitive problem statement, and for simpler analysis of protocols. However, in many cases it turned out that the initial definitions are insufficient in more complex contexts, where protocols are deployed within more general protocol environments. Some examples include encryption, where the basic notion of semantic security [GM84] was later augmented with several flavors of security against chosen ciphertext attacks [NY90, DDN00, RS91, BDPR98] and adaptive security [BH92, CFGN96], in order to address general protocol settings; Commitment, where the original notions were later augmented with some flavors of non-malleability [DDN00, DIO98, FF00] and equivocation [BCC88, B96] in order to address the requirement of some applications; Zero-Knowledge protocols, where the original notions [GMRa89, GO94] were shown not to be closed under parallel and concurrent composition, and new notions and constructions were needed [GK89, F91, DNS98, RK99, BGGL04]; Key Exchange, where the original notions do not suffice for providing secure sessions [BR93, BCK98, Sh99, CK01]; Oblivious Transfer [R81, EGL85, GM00], where the first definitions do not guarantee security under concurrent composition.

One way to capture the security concerns that arise in a specific protocol environment or in a given application is to directly represent the given environment or application within an extended definition of security. Such an approach is taken, for instance in the cases of key-exchange [BR93, CK01], non-malleable commitments [DDN00], and concurrent zero-knowledge [DNS98], where the definitions explicitly model several adversarially coordinated instances of the protocol in question. This approach, however, results in definitions with ever-growing complexity, and is inherently limited in scope since it addresses only specific environments and concerns.

An alternative approach, taken in this work, is to use definitions that consider the protocol in isolation, but guarantee *secure composition.* That is, here definitions of security inspect only a single instance of the protocol *"in vitro"*. Security *"in vivo"*, namely in more realistic settings where a protocol instance may run concurrently with other protocols, is guaranteed by making sure that the security is preserved under a general *composition operation* on protocols. This approach considerably simplifies the process of formulating a definition of security and analyzing protocols. Furthermore, it guarantees security in arbitrary protocol environments, even ones which have not been explicitly considered.

In order to make such an approach meaningful, we first need to have a general framework for representing cryptographic protocols and their security properties. Indeed, otherwise it is not clear what "preserving security when running alongside other protocols" means, especially when these other protocols and the security requirements from them are arbitrary. Several general definitions of secure protocols were developed over the years, e.g. [GL90, MR91, B91, BCG93, PW94, C00, HM00, PSW00, DM00, PW00]. These definitions are obvious candidates for such a general framework. However, many of these works consider only restricted settings and classes of tasks; more importantly, the composition operations considered in those works fall short of guaranteeing general secure composition of cryptographic protocols, especially in settings where security holds only for computationally bounded adversaries and many protocol instances may be running concurrently in an adversarially coordinated way. We further elaborate on these works and their relation to the present one in Appendix A.

This work proposes a framework for representing and analyzing the security of cryptographic protocols. Within this framework, we formulate a general methodology for expressing the security requirements of cryptographic tasks. Furthermore, we define a very general method for composing protocols, and show that notions of security expressed within this framework preserve security under this composition operation. We call this composition operation universal composition and say that definitions of security in this framework (and the protocols that satisfy them) are universally composable (UC). Consequently, we dub this framework the UC security framework.[1] As we'll see, the fact that security in this framework is preserved under universal composition implies that a secure protocol for some task remains secure even it is running in an arbitrary and unknown multi-party, multi-execution environment. In particular, some standard security concerns, such as non-malleability and security under concurrent composition, are satisfied even with respect to an unbounded number of instances of either the same protocol or other protocols.

The rest of the Introduction is organized as follows. Section 1.1 presents the basic definitional approach and the ideas underlying the formalism. Section 1.2 presents the universal composition operation and theorem. Section 1.3 discusses the issues associated with substantiating the general approach into a rigorous and usable framework. Related work, including both prior work and work that was done following the publication of the first version of this work, is reviewed in Appendix A.

1.1 The definitional approach

We briefly sketch the proposed framework and highlight some of its properties. A more comprehensive motivational presentation appears in [C06]. The overall definitional approach is the same as in most other general definitional frameworks mentioned above, and goes back to the seminal work of Goldreich, Micali and Wigderson [GMW87]: In order to determine whether a given protocol is secure for some cryptographic task, first envision an *ideal process* for carrying out the task in a secure way. In the ideal process all parties hand their inputs to a *trusted party* who locally computes the outputs, and hands each party its prescribed outputs. This ideal process can be regarded as a "formal specification" of the security requirements of the task. A protocol is said to *securely realize* the task if running the protocol "emulates" the ideal process for the task, in the sense that any

[1] We use similar names for two very different objects: A notion of security and a composition operation. This choice of names is motivated in Appendix A.

damage that can be caused by an adversary interacting with the protocol can also be caused by an adversary in the ideal process for the task.

Several formalizations of this general definitional approach exist, including the definitional works mentioned above, providing a range of secure composability guarantees in a variety of computational models. To better understand the present framework, we first briefly sketch the definitional framework of [C00], which provides a basic instantiation of the "ideal process paradigm" for the traditional task of secure function evaluation, namely evaluating a known function of the secret inputs of the parties in a synchronous, ideally authenticated network.

The model of protocol execution considered in [C00] consists of a set of interacting computing elements, representing the parties running the protocol. Formally, these elements are modeled as Interactive Turing Machines (ITMs). An additional ITM represents the adversary, who controls a subset of the parties. In addition, the adversary has some control over the scheduling of message delivery, subject to the synchrony guarantee. The parties and adversary interact on a given set of inputs and each party eventually generates local output. The concatenation of the local outputs of the adversary and all parties is called the global output. In the ideal process for evaluating some function f, all parties ideally hand their inputs to an incorruptible *trusted party,* who computes the function values and hands them to the parties as specified. Here the adversary is limited to interacting with the trusted party in the name of the corrupted parties. Protocol π securely evaluates a function f if for any adversary $\mathcal{A}$ (that interacts with the protocol) there exists an ideal-process adversary $\mathcal{S}$ such that, for any set of inputs to the parties, the global output of running π with $\mathcal{A}$ is indistinguishable from the global output of the ideal process for f with adversary $\mathcal{S}$.

This definition suffices for capturing the security of protocols in a "stand-alone" setting where only a single protocol instance runs in isolation. Indeed, if π securely evaluates f then the parties running π are guaranteed to generate outputs that are indistinguishable from the values of f on the same inputs. Furthermore, any information gathered by an adversary that interacts with π is generatable by an adversary that only gets the inputs and outputs of the corrupted parties. In addition, this definition is shown to guarantee security under non-concurrent composition, namely as long as no two protocol instances run concurrently. However, when protocol instances run concurrently, this definition no longer guarantees security: There are natural protocols that meet the [C00] definition but are insecure when as few as *two* instances run concurrently. We refer the reader to [C00, C06] for more discussions on the implications of and motivation for this definitional approach, including the failure to preserve security under concurrent composition.

The UC framework preserves the overall structure of that approach. The difference lies in new formulations of the model of computation and the notion of "emulation". As a preliminary step towards presenting these new formulation, we first present an alternative and equivalent formulation of the [C00] definition. In that formulation a new algorithmic entity, called the environment machine, is added to the model of computation. (The environment machine can be regarded as representing *whatever is external to the current protocol execution.* This includes other protocol executions and their adversaries, human users, etc.) The environment interacts with the protocol execution twice: First, it hands arbitrary inputs of its choosing to the parties and to the adversary. Next, it collects the outputs from the parties and the adversary. Finally, the environment outputs a single bit, which is interpreted as saying whether the environment thinks that it has interacted with the protocol or with the ideal process for f. Now, say that protocol π securely evaluates a function f if for any adversary $\mathcal{A}$ there exists an "ideal adversary" $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with π and $\mathcal{A}$ or with $\mathcal{S}$ and the ideal process for f. (In fact, a similar notion of environment is already used in [C00] to capture non-concurrent composability for adaptive adversaries.)

The main difference between the UC framework and the basic framework of [C00] is in the way the environment interacts with the adversary. Specifically, in the UC framework the environment and the adversary are allowed to interact freely throughout the course of the computation. In particular, they can exchange information after each message or output generated by a party running the protocol. If protocol π securely realizes function f with respect to this type of "interactive environment" then we say that π UC-realizes f.

This seemingly small difference in the formulation of the computational models is in fact very significant. From a conceptual point of view, it represents the fact that "information flow" between the protocol instance under consideration and the rest of the network may happen at any time during the run of the protocol, rather than only at input or output events. Furthermore, at each point the information flow may be directed both "from the outside in" and "from the inside out". Modeling such "circular" information flow is essential for capturing the threats of a multi-instance concurrent execution environment. (See some concrete examples in [C06].) From a technical point of view, the environment now serves as an "interactive distinguisher" between the protocol execution and the ideal process. This imposes a considerably more severe restriction on the ideal adversary $\mathcal{S}$, which must be constructed in the proof of security: In order to make sure that the environment $\mathcal{Z}$ cannot tell between a real protocol execution and the ideal process, $\mathcal{S}$ now has to interact with $\mathcal{Z}$ throughout the execution, just as $\mathcal{A}$ did. Furthermore, $\mathcal{S}$ cannot "rewind" $\mathcal{Z}$. Indeed, it is this pattern of free interaction between $\mathcal{Z}$ and $\mathcal{A}$ that allows proving that security is preserved under universal composition.

An additional difference between the UC framework and the basic framework of [C00] is that the UC framework allows capturing not only secure function evaluation but also *reactive* tasks where new input and output values are generated throughout the computation, and may depend on previously generated values. This is obtained by replacing the "trusted party" in the ideal process for secure function evaluation with a general algorithmic entity called an ideal functionality. The ideal functionality, which is modeled as another ITM, repeatedly receives inputs from the parties and provides them with appropriate output values, while maintaining local state in between. This modeling guarantees that the outputs of the parties in the ideal process have the expected properties with respect to the inputs, even when new inputs are chosen adaptively based on previous outputs. We note that this extension of the model is "orthogonal" to the previous one, in the sense that either extension is valid on its own. Some other differences from [C00] (e.g., capturing different communication models) are discussed in later sections.

The resulting definition of security turns out to be quite robust, in the sense that several natural definitional variants end up being equivalent. For instance, the notion of security as stated above is equivalent to the seemingly weaker variants where $\mathcal{S}$ may depend on the environment, or where the real-life adversary $\mathcal{A}$ is restricted to simply serve as a channel for relaying information between the environment and the protocol. It is also equivalent to the seemingly stronger variant where the ideal adversary $\mathcal{S}$ is restricted to black-box access to the adversary $\mathcal{A}$. (We remark that in other frameworks these variants result in different formal requirements; see e.g. [HU05].)

1.2 Universal Composition

Consider the following method for composing two protocols into a single composite protocol. (It may be useful to think of this composition operation as a generalization of the "subroutine substitution" operation for sequential algorithms to the case of distributed protocols.) Let π be some arbitrary protocol where the parties make ideal calls to some ideal functionality $\mathcal{F}$; in fact, they may make calls to multiple instances of $\mathcal{F}$. That is, in addition to the standard set of instructions, π may include instructions to provide instances of $\mathcal{F}$ with some input values, and to obtain output values from these instances of $\mathcal{F}$. Here the different instances of $\mathcal{F}$ are running at the same time without any global coordination. We call such protocols $\mathcal{F}$-hybrid protocols. (For instance, π may be a zero-knowledge protocol and $\mathcal{F}$ may provide the functionality of "digital envelopes", representing ideal commitment.) The burden of distinguishing among the instances of $\mathcal{F}$ is left with protocol π; we provide a generic mechanism for doing so, using *session identifiers.*

Now, let ρ be a protocol that UC-realizes $\mathcal{F}$, according to the above definition. Construct the composed protocol π^ρ by starting with protocol π, and replacing each invocation of a new instance of $\mathcal{F}$ with an invocation of a new instance of ρ. Similarly, inputs given to an instance of $\mathcal{F}$ are now given to the corresponding instance of ρ, and any output of an instance of ρ is treated as an output obtained from the corresponding instance of $\mathcal{F}$. It is stressed that, since protocol π may use an unbounded number of instances of $\mathcal{F}$ at the same time, we have that in protocol π^ρ there may be an unbounded number of instances of ρ which are running concurrently on related and dynamically chosen inputs.

The universal composition theorem states that running protocol π^ρ, with no access to $\mathcal{F}$, has essentially the same effect as running the original $\mathcal{F}$-hybrid protocol π. More precisely, it guarantees that for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{A}_\mathcal{F}$ such that no environment machine can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running π^ρ, or with $\mathcal{A}_\mathcal{F}$ and parties running π. In particular, if π UC-realizes some ideal functionality $\mathcal{G}$ then so does π^ρ.

On the universality of universal composition. Many different ways of "composing together" protocols into larger systems are considered in the literature. Examples include sequential, parallel, and concurrent composition, of varying number of protocol instances, where the composed instances are run either by the same set of parties or by different sets of parties, use either the same program or different programs, and have either the same input or different inputs. A more detailed discussion appears in [C06].

All these composition methods can be captured as special cases of universal composition. That is, any such method for composing together protocol instances can be captured via an appropriate "calling protocol" ρ that uses the appropriate number of protocol instances as subroutines, provides them with appropriately chosen inputs, and arranges for the appropriate synchronization in message delivery among the various subroutine instances. Consequently, it is guaranteed that a protocol that UC-realizes an ideal functionality $\mathcal{F}$ continues to UC-realize $\mathcal{F}$ even when composed with other protocols using any of the composition operations considered in the literature. In addition, universal composition allows expressing new ways of composing protocols, such as composing protocols where some of the local states are shared by multiple instances.

Universal composition also allows formulating new ways to put together protocols (or, equivalently, new ways to decompose complex systems); a salient example here is the case of composition of protocol instances that have some joint state and randomness.

Interpreting the composition theorem. Traditionally, secure composition theorems are treated as tools for modular design and analysis of complex protocols. (For instance, this is the main motivation in [MR91, C00, DM00, PW00, PW01].) That is, given a complex task, first partition the task to several, simpler sub-tasks. Then, design protocols for securely realizing the sub-tasks, and in addition design a protocol for realizing the given task assuming that evaluation of the sub-tasks is possible. Finally, use the composition theorem to argue that the protocol composed from the already-designed sub-protocols securely realizes the given task. Note that in this interpretation the protocol designer knows in advance which protocol instances are running together and can control how protocols are scheduled.

The above application is indeed very useful. In addition, this work proposes another interpretation of the composition theorem, which is arguably stronger: We use it as a tool for addressing the concern described at the beginning of the Introduction, namely for gaining confidence in the sufficiency of a definition of security in a given protocol environment. Indeed, protocols that UC-realize some functionality are guaranteed to continue doing so within any protocol environment — even environments that are not known a-priori, and even environments where the participants in a protocol execution are unaware of other protocol instances that may be running concurrently in the system in an adversarially coordinated manner. This is a very useful (in fact, almost essential) security guarantee for protocols that run in complex and unpredictable environments, such as modern communication networks.

1.3 Using the framework

Perhaps the most important criterion for a general analytical framework such as the present one is its usefulness in analyzing the security of protocols of interest in settings of interest. There are a number of aspects here. First off, a useful framework should allow us to represent the protocols we are interested in. It should also allow expressing the security requirements of tasks of interest. Similarly, it should allow us to adequately express the execution environment we are interested in; This includes capturing the security threats we are concerned about, as well as the security guarantees we are given. Finally, the framework should be as intuitive and easy to use as possible. This means that representing requirements and guarantees should be natural and transparent. It also means that the framework should be as simple as possible, and that there should be easy and flexible ways to delineate and isolate individual parts of a given, potentially complex system.

These considerations are the leading guidelines in the many definitional choices made in substantiating the definitional ideas described earlier. Here we briefly highlight few of these choices. More elaborate discussions of definitional choices appear throughout this work.

One set of choices is geared towards enhancing the ability of the model to express a variety of realistic situations, protocol execution methods, and threats. Towards this goal, the model allows capturing open, multi-party distributed systems where no a-priori bound on the number of participants is known in advance, and where parties can join the system with programs and identities that are chosen dynamically during the course of the computation. Modeling such systems requires formulating mechanisms for participants to identify each other and to address messages to the desired recipient. It also requires extending the traditional definitions of resource-bounded computation. In addition, we provide mechanisms for identifying individual *protocols* and *protocol instances* within such systems, so that the number and identities of the participants in each protocol instance, as well as the number of protocol instances running concurrently, can change dynamically depending on the execution with no a-priori bound. We remark that this modeling approach stands in contrast to existing models of distributed computing. Indeed, existing models typically impose more static restrictions on the system; this results in reduced ability to express scenarios and concerns that are prevalent in modern networks.

Another set of choices is geared toward allowing the protocol analyst to express security requirements in a precise and flexible way. This includes providing simple ways to express basic concerns, such as correctness, secrecy and fairness. It also includes providing ways to "fine-tune" the requirements at wish. In the present model, where a set of security requirements translates to a program of an ideal functionality, this means providing tools for writing ideal functionalities that express different types of requirements. A main tool here is to have the ideal functionality exchange information directly with the adversary throughout the computation. This allows expressing both the allowed adversarial influence and the allowed information leakage. In addition, the model allows the ideal functionality to execute code provided by the adversary; this allows expressing adversarial influence that is legitimate as long as it is carried out in isolation from the external environment.

Yet another set of choices is geared towards keeping the basic framework simple and concise, while allowing to capture a variety of communication, corruption, and trust models. For this purpose we use the same technical tool, ideal functionalities, to represent both a security specification for protocols, and the abstractions provided by a given communication, corruption or trust model. In the first use, we consider protocols that UC-realize the given ideal functionality, $\mathcal{F}$. In the second use, we consider $\mathcal{F}$-hybrid protocols, namely protocols that use the ideal functionality as a "trusted subroutine". In fact, it is often convenient to use an ideal functionality in both ways. Here the universal composition theorem implies that a protocol that UC-realizes $\mathcal{F}$ can be composed with a protocol designed in the abstract model captured by $\mathcal{F}$, to obtain a protocol that no longer needs the abstractions provided by $\mathcal{F}$.

Relying on this dual interpretation of ideal functionalities, we allow the framework to provide only very basic and rudimentary methods of communication between parties. This basic model is not intended to adequately capture any realistic setting. Realistic settings are then captured via formulating appropriate ideal functionalities within the basic model. This approach greatly simplifies the basic framework and facilitates arguing about it. It also makes assertions about the framework (such as the universal composition theorem) more general, since they directly apply to any communication, corruption or trust model devised within the framework. In addition, this approach provides greater flexibility in expressing different abstractions and variants thereof. The present modular approach to capturing abstract models should be contrasted with other models in the literature for security analysis, which are typically tied to a specific setting and have to be re-done with the appropriate modifications for each new setting.

To exemplify these aspects of the UC framework, we present a handful of ideal functionalities that are aimed at capturing some salient communication and corruption models. Some other ideal functionalities appear in the literature (see Appendix A).

1.4 Organization

Due to its length, the review of related work is postponed to the Appendix. Section 2 contains an informal exposition of the framework, definition of security, and composition theorem. The basic model for representing multiparty protocols is presented in Section 3. The general definition of security is presented in Section 4. The composition theorem and its proof are presented in Section 5. Finally, Section 6 demonstrates how some salient models of computation may be captured within the UC framework.

2 The framework in a nutshell

This section presents a simplified and somewhat informal version of the definition of security and the composition theorem. The purpose of this presentation is to highlight the salient definitional ideas that underlie the UC framework. For this purpose, we intentionally "gloss over" many details that are essential for a rigorous general treatment; these are postponed to later sections. Still, this section intends to be self-contained.

Section 2.1 sketches a model for representing multiple interacting computer programs. Section 2.2 presents the definition of security. Section 2.3 presents the composition theorem and its proof.

2.1 The underlying computational model

We present a rudimentary model for representing computing elements that interact over an asynchronous and untrusted network. A considerably more detailed and general model is presented in Section 3, along with motivating discussions.

The basic computing unit. The basic computing unit represents a running instance of a computer program (algorithm). For now, we omit a precise description of such a unit. Possible formalizations include an interactive Turing machine as in [GMRa89, G01], a process (as in [M89, M99, H85, LMMS99]) an I/O automaton (as in [Ly96]), etc. For the rest of this section we'll call such a unit a machine.

The model of computation consists of several machines that run "concurrently" (i.e., alongside each other) and provide each other with information. It will be convenient to distinguish three different ways in which a machine M can provide information to a machine M'. M can either provide input to M', send a message to M', or provide subroutine output to M'. Figure 2.1 presents a graphical depiction of a machine.

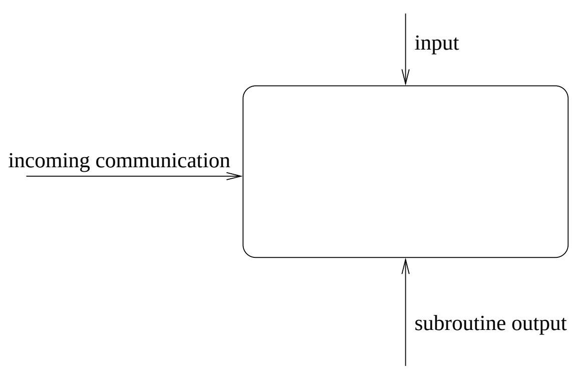

Figure 1: A basic computing unit (machine). Information from the outside world comes as either *input*, or *incoming communication*, or *subroutine output*. For graphical clarity, in future drawings we draw inputs as lines coming from above, incoming communication as lines coming from either side, and subroutine outputs as lines coming from below.

There are many ways to model a system whose components run concurrently. While the definitional approach of this work applies regardless of the specific modeling, for concreteness we consider a specific execution model. The model is simple: An execution of a system of machines $M_1, M_2, ...$ on input x starts by running M_1, called the *initial machine*, with input x. From this point on, the machines take turns in executing according to the following order: Whenever a machine M provides information to machine M', the execution of M is suspended and the execution of M' begins (or resumes).

Protocols and protocol instances. Intuitively, a protocol is an algorithm written for a distributed system. That is, a protocol consists of a collection of computer programs that exchange information with each other, where each program is run by a different participant and with different local input. For simplicity, in this section we restrict attention to protocols where the number of participants is fixed; an m-party protocol has m participants.

An instance of a protocol within a system of machines represents a sequence of machines, such that all these machines "relate to each other as part of the same protocol instance". A precise formulation of this seemingly simple concept is tricky; for the purpose of this section we simply say that an instance of an m-party protocol $\pi = \pi_1, ..., \pi_m$ within a system $\mathcal{M} = M_1, M_2, ...$ of machines is a subsequence of $\mathcal{M}$, such that the ith machine in the subsequence runs the ith program, π_i. (Of course, not any such subsequence is a "valid" protocol instance.)

Some of the machines (or, parties) in a protocol instance may be designated as subroutines of other parties in the instance. Intuitively, if party M' is a subroutine of party M then M will provide input to M' and obtain subroutine output from M'. A party of protocol π that is not a subroutine of another party of π is called a main party of π.

Polynomial time ITMs and protocols. We restrict attention to systems where all the machines have only "feasible" computation time, where feasible is interpreted as polynomial in some parameter. Furthermore, we assume that the overall computation of a system is "polynomial", in the sense that it can be simulated on a standard polynomially bounded Turing machine. We defer more precise treatment to subsequent sections.

2.2 Defining security of protocols

Following [GMW87], security of protocols with respect to a given task is defined by comparing an execution of the protocol to an ideal process where the outputs are computed by a trusted party that sees all the inputs. We substantiate this approach as follows. First, we substantiate the process of executing a protocol in the presence of an adversary and in a given computational environment. Next, the "ideal process" for carrying out the task is substantiated. Finally, we define what it means for an execution of the protocol to "mimic" the ideal process.

The model of protocol execution. We describe the model of executing an m-party protocol π in the presence of an adversary and in a given execution environment (or, "context").

The model consists of a system of machines $(\mathcal{Z}, \mathcal{A}, \pi_1, ..., \pi_m)$ where $\mathcal{Z}$ is called the environment, $\mathcal{A}$ is called the adversary, and $\pi_1, ..., \pi_m$ are an instance of protocol π. Intuitively, the environment represents all the other protocols running in the system, including the protocols that provide inputs to, and obtain outputs from, the protocol instance under consideration. The adversary represents adversarial activities that are directly aimed at the protocol execution under consideration, including attacks on protocol messages and corruption of protocol participants.

We impose the following restrictions on the way in which the machines may interact. The environment $\mathcal{Z}$ is allowed to provide only *inputs* to other machines. Furthermore, it can provide inputs only to $\mathcal{A}$ and to the *main* parties of π. A party of π may send messages to $\mathcal{A}$, give inputs to its subroutines, or give subroutine output to the machines whose subroutine it is. If this party is a main party of π then it may provide subroutine output to $\mathcal{Z}$. The adversary $\mathcal{A}$ may send messages to the parties of π or give subroutine output to $\mathcal{Z}$. A graphical depiction of the model of protocol execution appears in Figure 2.

Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(z)$ denote the random variable (over the local random choices of all the involved machines) describing the output of environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and parties running protocol π on input z as described above. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the en-

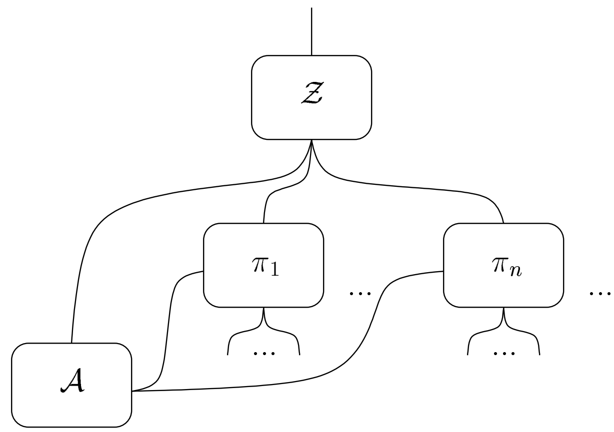

Figure 2: The model of protocol execution. The environment $\mathcal{Z}$ writes the inputs and reads the subroutine outputs of the main parties running the protocol, while the adversary $\mathcal{A}$ controls the communication. In addition, $\mathcal{Z}$ and $\mathcal{A}$ interact freely. The parties of π may have subroutines, to which $\mathcal{Z}$ has no direct access.

semble $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(z)\}_{z\in\{0,1\}^*}$. For the purpose of the present framework, it suffices to consider the case where the environment is allowed to output only a single bit. In other words, the ensemble $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ is an ensemble of distributions over $\{0,1\}$.

Discussion. Several remarks are in order at this point. First note that, throughout the process of protocol execution, the environment $\mathcal{Z}$ has access only to the inputs and outputs of the main parties of π. It has direct access neither to the communication among the parties, nor to the inputs and outputs of the subroutines of π. The adversary $\mathcal{A}$ has access only to the communication among the parties and has no access to their inputs and outputs. This is in keeping with the intuition that $\mathcal{Z}$ represents the protocols that provides inputs to and obtains outputs from the present instance of π, while $\mathcal{A}$ represents an adversary that attacks the protocol via the communication links, without having access to the local (and potentially secret) inputs and outputs.

In addition, $\mathcal{Z}$ and $\mathcal{A}$ may exchange information freely during the course of the computation. It may appear at first that no generality is lost by assuming that $\mathcal{A}$ and $\mathcal{Z}$ disclose their entire internal states to each other. A closer look shows that, while no generality is lost by assuming that $\mathcal{A}$ reveals its entire state to $\mathcal{Z}$, the interesting cases occur when $\mathcal{Z}$ holds some "secret" information back from $\mathcal{A}$, and tests whether the information received from $\mathcal{A}$ is correlated with the "secret" information. In fact, as we'll see, keeping $\mathcal{A}$ and $\mathcal{Z}$ separate is crucial for the notion of security to make sense.

Another point to notice is that this model gives the adversary complete control over the communication. That is, the model represents a completely asynchronous, unauthenticated, and unreliable network. This is indeed a very rudimentary model for communication; we call it the bare model. More "abstract" models are defined later, building on this bare model.

Yet another point is that the model does not contain a dedicated instruction for party corruption. Party corruption is modeled as a special type of incoming message to the party; the party's response to this message is determined by a special part of the party's program. (To guarantee that the environment knows who is corrupted, we restrict attention to adversaries that notify the environment upon each corruption of a party.)

Finally, note that the only external input to the process of protocol execution is the input of $\mathcal{Z}$. This input can be seen as representing an initial state of the system; in particular, it includes the inputs of all parties. From a complexity-theoretic point of view, providing the environment with arbitrary input is equivalent to stating that the environment is a non-uniform ITM.

Ideal functionalities and ideal protocols. Security of protocols is defined via comparing the protocol execution to an *ideal process* for carrying out the task at hand. For convenience of presentation, we formulate the ideal process for a task as a special protocol within the above model of protocol execution. (This avoids formulating an ideal process from scratch.) A key ingredient in this special protocol, called the ideal protocol, is an ideal functionality that captures the desired functionality, or the specification, of the task by way of a set of instructions for a "trusted party".

That is, let $\mathcal{F}$ be an ideal functionality (i.e., an algorithm for the trusted party), and assume that $\mathcal{F}$ expects to interact with m participants. Then an instance of the ideal protocol $\textsc{ideal}_{\mathcal{F}}$ consists of m main parties, called dummy parties, plus a party $\mathcal{F}$ that's a subroutine of all the main parties. Upon receiving an input v, each dummy party forwards v as input to the subroutine running $\mathcal{F}$. Any subroutine output coming from $\mathcal{F}$ is forwarded by the dummy party as subroutine output for the environment. Note that the inputs and outputs are handed over directly and reliably. A graphical depiction of the ideal protocol appears in Figure 3.

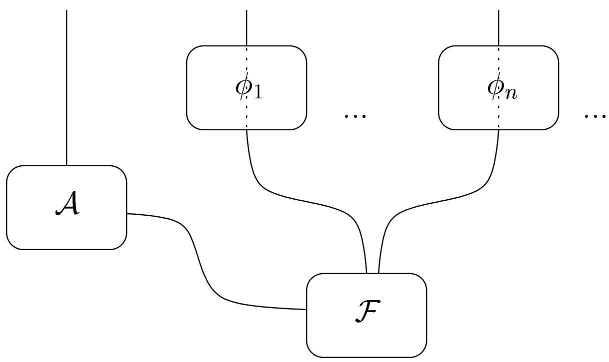

Figure 3: The ideal protocol $\textsc{ideal}_{\mathcal{F}}$ for an ideal functionality $\mathcal{F}$. The main parties of $\textsc{ideal}_{\mathcal{F}}$ are "dummy parties": they relay inputs to $\mathcal{F}$, and relay outputs of $\mathcal{F}$ to $\mathcal{Z}$. The adversary $\mathcal{A}$ communicates only with $\mathcal{F}$.

Note that $\mathcal{F}$ can model reactive computation, in the sense that it can maintain local state and its outputs may depend on all the inputs received and all random choices so far. In addition, $\mathcal{F}$ may receive messages directly from the adversary $\mathcal{A}$, and may contain instructions to send messages to $\mathcal{A}$. This "back-door channel" of direct communication between $\mathcal{F}$ and $\mathcal{A}$ provides a way to *relax* the security guarantees provided $\mathcal{F}$. Specifically, by letting $\mathcal{F}$ take into account information received from $\mathcal{A}$, it is possible to capture the "allowed influence" of the adversary on the outputs of the parties, in terms of both contents and timing. By letting $\mathcal{F}$ provide information directly to $\mathcal{A}$ it is possible to capture the "allowed leakage" of information on the inputs and outputs of the parties.

Protocol emulation. It remains to define what it means for a protocol to "mimic" or "emulate" the ideal process for some task. As a step towards this goal, we first formulate a more general notion of emulation, which applies to any two protocols. Informally, protocol π emulates protocol ϕ if, from the point of view of any environment, protocol π is "just as good" as ϕ, in the sense that no environment can tell whether it is interacting with π and some (known) adversary, or with ϕ and some other adversary. More precisely:

Definition (protocol emulation, informal statement): *Protocol* π UC-emulates *protocol* ϕ *if for any adversary* $\mathcal{A}$ *there exists an adversary* $\mathcal{S}$ *such that, for any environment* $\mathcal{Z}$ *the ensembles* $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ *and* $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ *are indistinguishable. That is, on any input, the probability that* $\mathcal{Z}$ *outputs 1 after interacting with* $\mathcal{A}$ *and parties running* π *differs by at most a negligible amount from the probability that* $\mathcal{Z}$ *outputs 1 after interacting with* $\mathcal{S}$ *and* ϕ*.*

We often call the adversary $\mathcal{S}$ a *simulator.* This is due to the fact that in typical proofs of security the constructed $\mathcal{S}$ operates by simulating an execution of $\mathcal{A}$. Also, we call the emulated protocol ϕ as a reminder that in the definition of realizing a functionality (see below), ϕ takes the role of the ideal protocol for some ideal functionality $\mathcal{F}$.

The notion of protocol emulation treats the environment as an "interactive distinguisher" between the process of running protocol π with adversary $\mathcal{A}$ and the process of running protocol ϕ with adversary $\mathcal{S}$; it is required that the two interactions remain indistinguishable even given the ability to interact with them as they evolve. In this sense, this notion strengthens the traditional notion of indistinguishability of distributions. On the other hand, this notion is a relaxation of the notion of *observational equivalence* of processes (see, e.g., [M89]); indeed, observational equivalence essentially fixes the entire system outside the protocol instances, whereas protocol emulation allows the analyst to choose an appropriate simulator that will make the two systems look observationally equivalent.

Securely realizing an ideal functionality. Once the general notion of protocol emulation is defined, the notion of realizing an ideal functionality is immediate:

Definition (realizing functionalities, informal statement): *Protocol* π UC-realizes *an ideal functionality* $\mathcal{F}$ *if* π *emulates* $\text{IDEAL}_{\mathcal{F}}$*, the ideal protocol for* $\mathcal{F}$*.*

To appreciate the meaning of this notion, consider a protocol π that UC-realizes and ideal functionality $\mathcal{F}$. Observe that the parties running π are guaranteed to generate outputs that are indistinguishable from the outputs provided by $\mathcal{F}$ on the same inputs; this guarantees *correctness.* Furthermore, any information gathered by an adversary that interacts with π is obtainable by an adversary that only interacts with $\mathcal{F}$; this guarantees *secrecy.* See [C06] for more discussion on the motivation for and meaning of this definitional style.

2.3 The composition theorem

As in the case of protocol emulation, we present the composition operation and theorem in the more general context of composing two arbitrary protocols. The case of ideal protocols and protocols that UC-realize them follows as a corollary.

We first define what it means for one protocol to use another protocol as a subroutine. Essentially, a protocol ρ uses protocol ϕ as a subroutine if some or all of the programs of ρ use programs of ϕ as subroutines, and these programs of ϕ communicate with each other as a single protocol instance. Said otherwise, protocol instance $\vec{\phi} = \phi_1, ..., \phi_m$ of ϕ is a subroutine of protocol instance $\vec{\rho} = \rho_1, ..., \rho_{m'}$ of ρ if each machine in $\vec{\phi}$ appears as a machine in $\vec{\rho}$ and is a subroutine of another machine in $\vec{\rho}$. We note that an instance of ρ may use multiple subroutine instances of ϕ; these instances of ϕ may run concurrently, in the sense that activations of parties in the two instances may interleave an an arbitrary order.

The universal composition operation. The universal composition operation is a natural generalization of the "subroutine substitution" operation for sequential algorithms to the case of distributed protocols. That is, let ρ be a protocol that uses protocol ϕ as a subroutine, and let π be a protocol that UC-emulates ϕ. The composed protocol, denoted $\rho^{\pi/\phi}$, is the protocol that is identical to ρ, except for the following change. For *each instance of* ϕ that's a subroutine of this instance of ρ, and for all i, the ith machine of this instance of ϕ now runs the program of the ith machine of π. In particular, all the inputs provided to an instance of ϕ that's a subroutine of this instance of ρ are now given to the corresponding instance of π, and all the outputs of this instance of π are treated as coming from the corresponding instance of ϕ. It is stressed that an instance of ρ may use multiple instances of ϕ; In this case, an instance $\pi^{\rho/\phi}$ will use multiple instances of ρ. A graphical depiction of the composition operation appears in Figure 4.

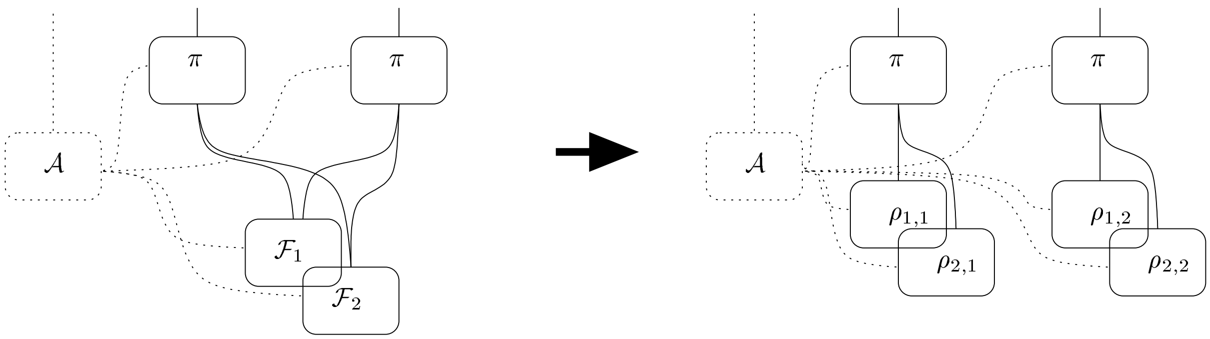

Figure 4: The universal composition operation, for the case where the replaced protocol is an ideal protocol for $\mathcal{F}$. Each instance of $\mathcal{F}$ (left figure) is replaced by an instance of π (right figure). Specifically, for $i = 1, 2$, $\rho_{i,1}$ and $\rho_{i,2}$ constitute an instance of protocol ρ that replaces the instance $\mathcal{F}_i$ of $\mathcal{F}$. The solid lines represent inputs and outputs. The dotted lines represent communication. The "dummy parties" for $\mathcal{F}$ are omitted from the left figure for graphical clarity.

The composition theorem. In its general form, the composition theorem says that if protocol π UC-emulates protocol ϕ then, for any protocol ρ, the composed protocol $\rho^{\pi/\phi}$ emulates ρ. This can be interpreted as asserting that replacing calls to ϕ with calls to π does not affect the behavior of ρ in any distinguishable way:

Theorem (universal composition, informal statement): *Let ρ, ϕ, π be protocols such that ρ uses ϕ as subroutine and π UC-emulates ϕ. Then protocol $\rho^{\pi/\phi}$ UC-emulates ρ.*

A first, immediate corollary of the general theorem states that if protocol π UC-realizes an ideal functionality $\mathcal{F}$, and ρ uses as subroutine protocol $\text{IDEAL}_{\mathcal{F}}$, the ideal protocol for $\mathcal{F}$, then the composed protocol $\rho^{\pi/\phi^{\mathcal{F}}}$ UC-emulates ρ. Another corollary is that if ρ UC-realizes an ideal functionality $\mathcal{G}$, then so does $\rho^{\pi/\phi}$

On the proof of the universal composition theorem. We briefly sketch the main ideas behind the proof. Let $\mathcal{A}$ be an adversary that interacts with parties running $\rho^{\pi/\phi}$. We need to construct an adversary $\mathcal{A}_\rho$, such that no environment $\mathcal{E}$ will be able to tell whether it is interacting with $\rho^{\pi/\phi}$ and $\mathcal{A}$ or with ρ and $\mathcal{A}_\rho$. The idea is to construct $\mathcal{A}_\rho$ in two steps: First we define a special adversary, denoted $\mathcal{D}$, that operates against protocol π as a stand-alone protocol. The fact that π emulates ϕ guarantees that there exist an adversary ("simulator") $\mathcal{A}_\phi$, such that no environment can tell whether it is interacting with π and $\mathcal{D}$ or with ϕ and $\mathcal{A}_\phi$. Next, we construct $\mathcal{A}_\rho$ out of $\mathcal{A}$ and $\mathcal{A}_\phi$.

We sketch the above steps. Adversary $\mathcal{D}$ will be a "dummy adversary" that merely serves as a "channel" between $\mathcal{A}$ and a single instance of π. That is, $\mathcal{D}$ expects to receive in its input (coming from the environment $\mathcal{E}$) requests to deliver messages to prescribed parties of an instance of π. $\mathcal{D}$ then carries out these requests. In addition, any incoming message (from some party of the instance of π) is forwarded by $\mathcal{D}$ to its environment. Recall that we are now given an adversary ("simulator") $\mathcal{A}_\phi$, such that no environment can tell whether it is interacting with π and $\mathcal{D}$ or with ϕ and $\mathcal{A}_\phi$.

The construction of $\mathcal{A}_\rho$ proceeds as follows. $\mathcal{A}_\rho$ runs a simulated instance of $\mathcal{A}$ and follows the instructions of $\mathcal{A}$, with the following exception. $\mathcal{A}$ expects to interact with multiple instances of the subroutine protocol π, whereas $\mathcal{A}_\rho$ interacts with instances of ϕ rather than instances of π. For this purpose, the interaction of $\mathcal{A}$ with the various instances of π is simulated using instances of $\mathcal{S}_\phi$. Here $\mathcal{A}_\rho$ plays the role of the environment for the instances of $\mathcal{S}_\phi$. The ability of $\mathcal{A}_\rho$ to obtain timely information from the instances of $\mathcal{S}_\phi$, by playing the environment for them, is at the crux of the proof. We also use the fact that $\mathcal{S}_\phi$ must be defined independently of the environment; this is guaranteed by the order of quantifiers.

The validity of the simulation is demonstrated via reduction to the validity of $\mathcal{S}_\phi$. Dealing with many instances of $\mathcal{S}_\phi$ running concurrently is done using a hybrid argument: We define many hybrid executions, where the first execution is an execution of ρ with $\mathcal{A}$, and in each successive hybrid execution one more instance of ϕ is replaced with an instance of π. Now, given an environment $\mathcal{Z}$ that distinguishes between two successive hybrid executions, we construct an environment $\mathcal{Z}_\pi$ that distinguishes between an execution of π with adversary $\mathcal{D}$ and an execution of ϕ with adversary $\mathcal{S}_\phi$. Essentially, $\mathcal{Z}_\pi$ orchestrates for $\mathcal{Z}$ an entire interaction with ρ, where some of the instances of ϕ are replaced with instances of π, and where one of these instances is relayed to the external system that $\mathcal{Z}_\pi$ interacts with. This is done so that if $\mathcal{Z}_\pi$ interacts with an instance of π then $\mathcal{Z}$, run by $\mathcal{Z}_\pi$, "sees" an interaction with one hybrid execution, and if $\mathcal{Z}_\pi$ interacts with an instance of ϕ then $\mathcal{Z}$ "sees" an interaction with the next hybrid execution. Here we use the fact that an execution of an entire system can be efficiently simulated on a single machine.

The composition theorem can be extended to handle polynomially many applications, namely polynomial "depth of nesting" in calls to subroutines. However, when dealing with computational security (i.e., polynomially bounded environment and adversaries), the composition theorem does not hold in general in cases where the emulated protocol ϕ is not polynomially bounded.

3 The model of computation

In order to turn the definitional sketch of Section 2 into a concrete definition, one first needs to formulate a rigorous model for representing distributed systems and interacting computer programs within them. This section presents such a model. We also put forth a definition of resource-bounded computation that behaves well within the proposed model. While the proposed model is rooted in existing ones, many of the choices are new. We try to motivate and justify departure from existing formalisms. In order to allow for adequate presentation and motivation of the main definitional ideas, we present the basic model of distributed computation separately from the definitions of security (Section 4). This also simplifies the presentation in subsequent sections.

It is tempting to dismiss the specific details of such a model as being "of no real significance" for the validity and meaningfulness of the notions of security built on top the model. This intuition is perhaps rooted in the Church-Turing thesis, which stipulates that "all reasonable models of computation lead to essentially equivalent notions and essentially the same results". However, at least in the case of distributed, resource-bounded systems with potentially adversarial components, this intuition does not seem to hold. There are numerous meaningful definitional choices to be made on several levels, including the basic modeling of resource-bounded computation and the communication between components, the modeling of scheduling and concurrency, and the representation of the dynamically changing aspects of distributed systems. Indeed, as we'll see, some seemingly small and "inconsequential" choices in the modeling turn out to have significant effects on the meaning of the notions of computability and security built on top of the model.

The present model strives to achieve both *simplicity* and *generality.* That is, the model attempts to be general and flexible enough so as to allow expressing the salient aspects of modern distributed systems. At the same time, it attempts to be as simple and clear as possible, with a minimal number of formal and notational constructs. While the model is intended to make sense on its own as a model for distributed computation, it is of course geared towards enabling the representation of security properties of algorithms in a way that is as natural and simple as possible, and with as little as possible reference to the mechanics of the underlying model.

Section 3.1 presents the basic model. Section 3.2 presents the definition of resource-bounded computation. Some alternative formulations are mentioned throughout, as well as in Appendix A.1.

3.1 The basic model

The main objects we wish to analyze are *algorithms,* or *computer programs,* written for a distributed system. (We often use the term *protocols* when referring to such algorithms.) There are several differences between distributed algorithms and algorithms written for standard one-shot sequential execution. First, a distributed algorithm (protocol) may consist of several programs, where each program runs on a separate computing device, with its own local input, and where the programs interact by exchanging information with each other. Also, computations are often *reactive,* namely they allow for interleaved inputs and outputs where each output potentially depends on all prior inputs. Second, different programs may be run by different entities with different interests, and one has to take into account the case where some of these entities deviate from the prescribed program. Third, distributed systems often consist of several different protocols (or several instances of the same protocol), all running "at the same time". These protocols may also exchange information with each other during their execution, say by providing inputs and outputs to each other. Fourth, modern systems allow programs to be generated dynamically within one device and "executed" remotely within another device. The devised model should account for these aspects of distributed algorithms.

We proceed in two main steps. First (Section 3.1.1), we define a *syntax,* or a "programming language" for protocols. This language, which extends the notion of *interactive Turing machine* [GMRa89], allows expressing instructions and constructs needed for operating in a distributed system. Next (Section 3.1.2), we define the *semantics* of a protocol, namely an execution model for distributed systems which consist of one or more protocols as sketched above. To facilitate readability, we postpone most of the motivating discussions to section 3.1.3. We point to the relevant parts of the discussion as we go along.

3.1.1 Interactive Turing Machines (ITMs)

An interactive Turing machine (ITM) extends the standard Turing machine formalism to capture a distributed algorithm (protocol). A definition of interactive Turing machines, geared towards capturing *pairs* of interacting machines is given in [GMRa89] (see also [G01, Vol I, Ch. 4.2.1]). That definition adds to the standard definition of a Turing machine a mechanism that allows a *pair* of machines to exchange information via writing on special "shared tapes". Here we extend this formalism to accommodate protocols written for systems with *multiple* computing elements, and where multiple concurrent executions of various protocols co-exist. For this purpose, we define a somewhat richer syntax (or, "programming language") for ITMs. The full meaning of the added syntax will become clear only in conjunction with the model of execution (Section 3.1.2). Still, we sketch here the main additions.

First, we provide a mechanism for identifying a specific *addressee* for a given piece of transmitted information. Similarly, we allow a recipient of a piece of information to identify the source. Second, we provide a mechanism for distinguishing between different "protocol instances" within a multi-component system.

Third, we provide several types of "shared tapes", in order to facilitate distinguishing between different types of communicated information. The distinction proceeds on two planes: On one plane, we wish to facilitate distinguishing between communication "internal to a protocol instance" (namely, among the participants of the protocol instance) and communication "with other protocol instances". Furthermore, we wish to facilitate distinguishing between communication with "calling protocols" and communication with "subroutine protocols". On another plane, we wish to distinguish between information communicated over a "trusted medium", say locally within the same computing environment, and communication over an "untrusted medium", say across a network.

As a disclaimer we note that the formalism, or "programming language" below is not *essential*, in the sense that several other alternative languages would probably suffice as a basis for a similar formalism. (In fact, all the constructs below can be easily implemented on top of a standard Turing machine.) Still, as we'll see, fixing *some* language is essential for considering a number of salient points. We postpone to Section 3.1.3 additional discussion on the reasons for choosing the ITM model as the basic model of computation, as well as several other choices.

Definition 1 *An* interactive Turing machine (ITM) *M is a Turing machine (as in, say, [Si05]) with the following augmentations:*

Special tapes *(i.e., data structures)*:

- *An* identity tape. *The contents of this tape is interpreted as two strings. The first string contains a description, using some standard encoding, of the program of M (namely, its transition function). We call this description the* code *of M. The second string is called the* identity *of M. The identity of M together with its code is called the* extended identity *of M. This tape is "read only". That is, M cannot write to this tape.*
- *An* outgoing message tape. *Informally, this tape holds the current outgoing message generated by M, together with sufficient addressing information for delivery of the message.*
- *An* input tape. *Informally, this tape represents inputs coming from other programs or physical devices. Information on this tape will be treated as coming over a "trusted medium".*
- *An* incoming communication tape. *Informally, this tape represents information coming*

from other programs within the same "protocol instance". Information on this tape will be treated as coming over an "untrusted medium".

- *A* subroutine output tape. *Informally, this tape represents outputs coming from programs or modules that were created as "subroutines" of the present program. Information on this tape will be treated as coming over a "trusted medium".*
- *A one-bit* activation tape. *Informally, this tape represents whether the ITM is currently "in execution".*

The use the generic term externally writable tapes *to denote the input, incoming communication and subroutine output tapes.*

New instructions:

- *An* external write *instruction. Very roughly, the effect of this instruction is that the message currently written on the outgoing message tape is possibly written to the specified tape of the machine with the identity specified in the outgoing message tape. Precise specification is postponed to Section 3.1.2.*
- *A* read next message *instruction. This instruction specifies a tape out of* {input, incoming communication, subroutine output}*. The effect is that the reading head jumps to the beginning of the next message. (To implement this instruction, we assume that each message ends with a special* end-of-message (eom) *character.)*

3.1.2 Executing Systems of ITMs

We specify the mechanics of executing a system that consists of multiple ITMs. Very roughly, the execution process resembles the sketch given in Section 2. However, some significant differences from that sketch do exist. First, here we make an explicit distinction between an ITM, which is a "static object", namely an algorithm or a program, and an *ITM instance (ITI),* which is a "run-time object", namely an instance of a program running on some specific data. In particular, the same program (ITM) may have multiple instances (ITIs) in an execution of a system. (Conceptually, an ITI is closely related to a *process* in a process calculus. We refrain from using this term to avoid confusion with other formalisms.)

Second, here we give a more rigorous account of the mechanism for exchanging information between ITIs, or in other words the semantics of the "external write" instruction. This includes more detailed specification of the information given to the source and destination ITIs in an external write operation.

Third, the present model allows the number of ITIs to grow dynamically in an unbounded way as a function of the initial parameters, by explicitly modeling the "generation" of new ITIs. Furthermore, new ITIs may have dynamically determined programs. Here the fact that programs of ITMs can be represented as strings plays a central role.

Fourth, we augment the execution model with a *control function,* which regulates the transfer of information between ITIs. That is, the control function determines which "external write" instructions are "allowed"; "un-allowed" instructions are ignored. This added construct provides both clarity and flexibility to the execution model: All the model restrictions are expressed explicitly and in "one place." Furthermore, it is easy to define quite different execution models simply by changing the control function.

Systems of ITMs. Formally, a system of ITMs is a pair $S = (I, C)$ where I is an ITM, called the initial ITM, and $C : \{0,1\}^* \rightarrow \{allow, disallow\}$ is a control function.

Executions of systems of ITMs. A configuration of an ITM M consists of the description of the control state, the contents of all tapes and the head positions. (Recall that the program, or the transition function of M is written on the identity tape, so there is no need to explicitly specify the code in a configuration.) A configuration is active if the activation tape is set to 1, else it is inactive.

An instance $\mu = (M, id)$ of an ITM M consists of the program (transition function) of M, plus an identity string $id \in \{0,1\}^*$. We say that a configuration is a configuration of instance μ if the contents of the identity tape in the configuration agrees with μ, namely if the program encoded in the identity tape is M and the rest of the identity tape holds the string id. We use the acronym ITI to denote an ITM instance.

An activation of an ITI $\mu = (M, id)$ is a sequence of configurations that correspond to a computation of M starting from some *active configuration* of μ, until an inactive configuration is reached. (Informally, at this point the activation is complete and μ is waiting for the next activation.) If a special halt state is reached then we say that μ has halted; in this case, it does nothing in all future activations.

An execution of a system $S = (I, C)$ with identity string id_0 and input x consists of a sequence of activations of ITIs. The first activation starts from the configuration where the identity tape contains the code of I followed by the identity id_0, the input tape contains the value x, and a sufficiently long random string is written on the random tape. In accordance, the ITI (I, id_0) is called the initial ITI in this execution.

An execution ends when the initial ITI halts (that is, when a halting configuration of the initial ITI is reached). An execution is accepting if the initial ITI halted in an accepting state. An execution prefix is a prefix of an execution.

To complete the definition of an execution, it remains to specify: (a) The effect of an external-write instruction, and (b) How to determine the first configuration in the next activation, once an activation is complete. These points are described next.

Writing to a tape of another ITI and invoking new ITIs. The mechanism that allows communication between ITIs is the external write instruction. The same instruction is used also for invoking new ITIs. More specifically, the effect of an external write instruction is as follows. Let $\mu = (M, id)$ denote the ITI which performs the external write transition, The current contents of the outgoing message tape is interpreted (using some standard encoding) as consisting of an extended identity $\mu' = (M', id')$ of a "target ITI", a tape name out of {input, incoming communication, subroutine output}, and a string m called the message. Then:

1. If the control function C, applied to the current execution prefix, does not allow μ to write to the specified tape of μ' (i.e., it returns a *disallow* value) then the instruction is ignored.
2. If C allows the operation, and an ITI $\mu'' = (M'', id'')$ with identity $id'' = id'$ currently exists in the system (namely, one of the past configurations in the current execution prefix has identity id'), then:
   (a) If the target tape is the incoming communication tape, then the message m is written on the incoming communication tape of μ'', starting at the next blank space. (That is,

a new configuration of μ'' is generated. This configuration is the last configuration of μ'' in this execution, with the new information written on the incoming communication tape.) It is stressed that the code M'' of μ'' need not equal the code M' specified in the external write request.

This convention has the effect that an ITI does not necessarily know the code of the ITI it sends messages to using the communication tape. The recipient ITI learns neither the identity nor the code of the writing ITI. (Of course, this information may be included in the message itself, but the model provides no guarantees regarding the authenticity of this information.) The intuitive goal is to capture the effect of standard communication over an untrusted medium, such as a communication network.

(b) If the target tape is the input tape or subroutine output tape, and $M'' = M'$, then the specified message is copied to the specified tape of μ'', *along with the code and identity of* μ. If $M' \neq M''$ then the message is not copied and μ transitions to a special error state.

This convention has the effect that an ITI can verify the code of the ITI to whom it provides input or subroutine output. Furthermore, the recipient of an input or subroutine output knows both the identity and the code of the writing ITI. The intuitive goal here is to capture the effect of communication between processes within a "trusted" computing environment.

3. If C allows the operation, and no ITI with identity id' exists in the system, then a new ITI μ' with code M' and identity id' is invoked. That is, a new configuration is generated, with code M', the value id' written on the identity tape, and a sufficiently long random string is written on the random input tape. Once the new ITI is invoked, the external-write instruction is carried out as in Step 2. In this case, we say that μ invoked μ'.

On the uniqueness of identities. Section 3.1.3 discusses several aspects of the external-write instruction, and in particular motivates the differences from the inter-component communication mechanisms provided in other frameworks. At this point we only observe that the above invocation rules for ITIs, together with the fact that the execution starts with a single ITI, guarantee that each ITI in a system has unique identity. That it, no execution of a system of ITIs has two ITIs with the same identity, regardless of their codes. This property makes sure that the present addressing mechanism is unambiguous.

Determining the next activated ITI. In order to simplify the process of determining the next ITI to be activated, we allow an ITI to execute at most a single external-write instruction per activation. (Formally, this restriction is imposed by the control function. That is, we consider only control functions that enforce this restriction.) Then, once an activation is complete, the next ITI to be activated is the ITI μ'' whose tapes were written to. That is, the first configuration in the next activation is the last configuration of μ'' in the current execution prefix, with the exception that the activation tape of μ'' is set to 1. If no external-write operation was performed then the initial ITI is the next one to be activated. See Section 3.1.3 for a discussion on the order of activations of ITMs.

Outputs of executions. We use the following notation. Let $\text{OUT}_{I,C}(x, id_0)$ denote the random variable describing the output of the execution of the system (I, C) of ITMs when I's input is x

and identity is id_0. Here the probability is taken over the random choices of all the ITMs in the system.

Extended systems. The above definition of a system of ITMs postulates that an ITI always sets the code of the ITIs it invokes. The definition of security, formulated in Section 4, requires a more general way for generating ITIs: In some cases, it will be needed that the code of the generated ITI μ' is set by some "external" mechanism to be different than the code "intended" by the generating ITI, μ. The mechanism we use to provide this extra flexibility is the control function. That is, we extend the definition of a control function so that it can also *modify* the external-write requests made by parties. Recall that in a system $S = (I, C)$ the control function C takes as input a sequence of external-write requests and outputs either *allowed* or *disallowed.* In an extended system the output of C consists of an entire external-write instruction, which may be different than the input request. The executed instruction is the output of C. We stress that, although the above definition of an extended system gives the control function complete power in modifying the external-write instructions, the extended systems considered in this work use control functions that only modify the *codes* of the target ITIs.

Subroutines, etc. When an ITI μ writes a message m to the incoming communication tape of ITI μ', we say that μ sends m to μ'. When μ writes a value x onto the input tape of μ', we say that μ passes input x to μ'. When μ' writes x to the subroutine-output tape of μ, we say that μ' passes output x (or simply outputs x) to μ. We say that μ' is a subroutine of μ if μ has passed input to μ' or μ' has passed output to μ in this execution. (Note that μ' may be a subroutine of μ even when μ' was invoked by an ITI other than μ.) μ' is a subsidiary of μ if μ' is a subroutine of μ or of another subsidiary of μ.

Protocols. A protocol is defined as a (single) ITM as in Definition 1. As already discussed, the goal is to capture the notion of an *algorithm* written for a distributed system where physically separated participants engage in a joint computation; namely, the ITM describes the program to be run by each participant in the computation. If the protocol specifies different programs for different participants, or "roles", then the ITM should describe all these programs. (Alternatively, protocols can be defined as sets, or sequences of machines, where different machines represent the code to be run by different participants. However, such a formulation would add unnecessary complication to the basic model, e.g. to the definition of protocol instances, considered next.)

Protocol instances. The notion of a *running instance* of a protocol has strong intuitive appeal. However, rigorously defining it in way that's both natural and reasonably general turns out to be tricky. Indeed, what would be a natural way to delineate, or isolate, a single instance of a protocol within an execution of a dynamic system where multiple parties run multiple pieces of code?

Traditionally, an instance of a protocol in a running system is defined as a fixed set of machines that run a predefined program, often with identities that are fixed in advance. Such a definitional approach, however, does not account for protocol instances where the number of participants, or perhaps even only their identities, are determined dynamically as the execution unfolds. It also doesnt account for instances of protocols where the code has been determined dynamically, rather than being fixed at the onset of the execution. Thus, a more flexible definition is desirable.

The definition proposed here attempts to formalize the following intuition: "A set of ITIs in an execution of a system belong to the same instance of some protocol π if they all run π, and in addition they were invoked with the intention of interacting with each other for a joint purpose." In fact, since different participants in an instance are typically invoked within different physical entities in a distributed system, the last condition should probably be rephrased to say: "...and in addition the invoker of each ITI in the instance intends that ITI to participate in a joint interaction with the other ITIs in that instance."

We provide a formal way for an invoker of an ITI to specify which protocol instance this ITI is to participate in. The construct we use for this purpose is the identity string. That is, we interpret (via some standard unambiguous encoding) the identity of an ITI as *two* strings, called the session identifier (SID) and the party identifier (PID). We then say that a set of ITIs in a given execution prefix of some system is an instance of protocol π if all these ITIs have the code π and all have the same SID. The PIDs are used to differentiate between ITIs within a protocol instance; they can also be used to associate ITIs with "clusters", such as physical computers in a network. More discussion on the SID/PID mechanism appears in Section 3.1.3.

Consider some execution prefix of some system of ITMs. Each ITI in a protocol instance in this execution is called a party of that instance. A sub-party of an instance is a subroutine either of a party or of another sub-party of the instance. The extended instance of some protocol instance includes all the parties and sub-parties of this instance. If two protocol instances I and I' have the property that each party in instance I is a subroutine of a party in instance I' then we say that I is a subroutine instance of I'.

3.1.3 Discussion

This section contains more lengthy discussion that highlights and motivates certain aspects of the model.

Motivating the use of ITMs. Interactive Turing machines are only one of several standard mathematical models aimed at capturing computers in a network (or, more abstractly, interacting computing agents). Other models include the CSP model of Hoare [H85], the π-calculus of Milner [M89, M99] (that is based on the λ-calculus as its basic model of computation), the *spi*-calculus of Abadi and Gordon [AG97] (that is based on π-calculus), the framework of Lincoln et. al. [LMMS98] (that uses the functional representation of probabilistic polynomial time from [MMS98]), the I/O automata of Merritt and Lynch [Ly96], or the probabilistic I/O automata of Lynch, Segala and Vaandrager [SL95, LSV03].

Several reasons motivate the decision to use ITMs as a basis for this framework. First and foremost, ITMs seem to best represent the subtle interplay between communication, often with adversarial scheduling, and the complexity of local computations, which in addition may be randomized. This interplay is often at the heart of security analysis of protocols. In particular, ITMs naturally allow the asymptotic treatment of security as a function of the length of the input to the computation. See more discussion on these issues in Section 3.2.

Furthermore, ITMs seem to faithfully represent the way in which existing computers operate in a network. Examples include the duality between data and code, which facilitates the modeling of dynamic code generation, transmission and activation ("download"), and the use of a small number of physical communication channels to interact with a large (in fact, potentially unbounded) number of other parties. In particular, ITMs allow considering some "low level" complexity issues that are sometimes glossed over in other frameworks, such as the work spent on the addressing, sending, and receiving of messages as a function of the message length or the address space.

Also, ITMs seem to best mesh with standard models used in complexity theory (such as standard Turing machines, oracle machines, and circuit families).

On the down side, the ITM model, or "programming language" provides a relatively low level abstraction of computer programs and protocols. In contrast, practically all existing protocols are described in a much more high-level (and thus inherently informal) language. One way to bridge this gap is to develop a library of subroutines that will allow for more convenient representation of protocols as ITMs. An alternative way is to demonstrate "security preserving correspondences" between programs written in more abstract models of computation and limited forms of the ITMs model, such as the correspondences in [AR00, MW04, CH04, C+05].

Finally, that said, we stress that the present ITM model is in no way the only valid instantiation of the definitional approach presented in later sections. Any other "reasonable" model that allows representing resource-bounded computation together with adversarially controlled communication would do. Still, care should be taken not to use overly abstract or restricted models that do not allow expressing realistic concerns.

On the external-write mechanism. Traditionally, models of distributed computation (such as the ones mentioned above) allow the different components to exchange information via "dedicated named channels". That is, a component can, under various restrictions, write information to, and read information from a "channel name." Channel names are typically treated as fixed "system parameters", in the sense that they are not mutable by the programs running in the system. Furthermore, sending information on a channel is treated as an atomic operation regardless of the number of components in the system or the length of the message. Also, in some models the channel names to be used by each component have to be declared in advance.

This modeling of the communication is clean and elegant. It also facilitates reasoning about protocols framed within that model. I particular, it facilitates analytical operations that separate a system into smaller components by "cutting the channels", and re-connecting the components in different ways. However, this modeling is somewhat abstract and over-restrictive for our purposes. For one, The fact that the channel names are fixed in advance does not allow for effective addressing in settings where the number and makeup of components changes as the system evolves. Also, it does not allow for simple representation of protocols where the addressing mechanism is created within the protocol itself. It also doesnt account for the complexity of message addressing and delivery; in a dynamically growing systems this complexity may be an important factor. Finally, it does not account for dynamic generation of new programs.

The external-write mechanism, together with the control function, are aimed at providing a sufficiently low-level and flexible modeling that allows taking into account the above issues. First, they allow the analyzed protocols to generate the identities of participants in an algorithmic way, as part of the execution. They also provide a more dynamic and flexible extension of the static notions of "communication channels" or "ports" used in other models. Also, Requiring an ITI to explicitly write the recipient address and the message on a special tape allows considering the associated computational overhead. The only abstraction provided by this mechanism is the guarantee of global uniqueness of identities. Indeed, this guarantee is essential for unambiguous addressing. See more discussion on this point later on in this section. Below we highlight two additional aspects of the external write mechanism.

Letting the addressing depend on the code. The external write mechanism provides the writing party with the option to require that the message be written only if the target ITI runs the *code* (program) specified by the writing party. Furthermore, in some cases the recipient ITI learns the code of the writing party. Such an option is not explicitly given in other models. Indeed, at first glance it may seem odd: Why should the sender and recipient gain such information or control?

We first argue that, even when modeling systems that interact with untrusted entities, it is necessary for effective modeling to give a program the ability to sometimes completely specify the program it interacts with. Indeed, when writing a program it is often convenient to partition it into separate modules whose behavior is of interest only when interacting with specific other modules. (Typically, these modules will run within a single "trusted environment".)

Now, in models where the way in which components communicate is specified statically in advance, programs can be given such a guarantee simply by considering only systems where the appropriate connections are made. However, in the present modeling, where identities are generated dynamically and no static notions of "connections" or "channels" exist, an alternative mechanism is needed for verifying the code run by the communicating peer. The ability to explicitly specify the code of the recipient ITI and disclose the code of the writing ITI is such a mechanism.

It is stressed that in order to make the above mechanism meaningful, programs need to be written in ways that are "recognizable" by the peers. This can be done using standard encoding mechanisms. Indeed, a peer may accept one representation of a program, and reject another representation, even though the two might be equivalent. Alternatively, programs may be written in a way that allows the peer to verify some desired properties. (For instance, a recipient may decide to ignore the received data unless the sending ITI's code provides some basic guarantees which appropriately restrict its behavior.)

Invoking new ITIs. Allowing for dynamic invocation of new ITIs is important for modeling realistic situations where parties may join a computation as it unfolds, and where the number of protocol instances that run in the systems is not known in advance. (Indeed, such situations impose security requirements on protocols, that are not naturally expressible in a model where the number of components is fixed in advance. One example is the study of concurrent Zero-Knowledge, where the number of sessions depends on the adversary and cannot be bounded by any fixed polynomial. See e.g. [R06].) Similarly, we would like to be able to model commonplace situations where programs are generated automatically, "downloaded", and incorporated in a computation "on the fly". The external write mechanism provides a flexible way for incorporating in the computation new programs, as well as new instances of existing programs.

Two additional remarks are in order here. First, it may seem at first glance that the ability to dynamically generate arbitrary new code is not needed, since it suffices to consider only ITIs that run the code of a "universal Turing machine", and then provide these ITIs with appropriate code to be run as part of the input. However, if such a generic convention were adopted then ITIs would not be able to effectively verify that their communicating peers are running some acceptable programs; the necessity of such a mechanism was argued in the previous remark. It is thus important to allow newly generated ITIs to have code with some verifiable properties.

Second, recall that the invocation of a new ITI is implicit, i.e. it occurs only when an existing ITI attempts to write to a tape of a non-existing ITI. Furthermore, the writing ITI does not learn whether a new ITI was invoked. We adopt this convention since it seems natural; furthermore, it simplifies the model, the definition of security, and subsequently the presentation and analysis of protocols. Still, it is not essential: Once could, without significant effect on the expressibility of the model, add an explicit "instance invocation" operation and require that an ITI is invoked before it is first activated. In this case, however, a different mechanism for guaranteeing uniqueness of identities would be needed.

On the distinction between input, communication, and subroutine output tapes. The definition of ITMs provides three syntactically different methods to transfer information from one ITI to another: either via the input tape, via the incoming communication tape, or via the subroutine output tape. This distinction is used for a number of different purposes. A first use is to facilitate the distinction between various types of incoming information in a multi-party, multi-protocol, multi-instance system, and in particular to make explicit the notion of subroutines (which is central for the universal composition operation and theorem). Here the input tape is used to model information coming from a "calling program," namely a program that uses the present program as a subroutine. The communication tape is used to model information coming from "peers", or other programs within the same protocol instance. The subroutine output tape is used to model information coming from "subroutines" of the present program. This interpretation of the different tapes is represented in Figure 2.1 on page 8.

A second use is to facilitate the distinction between "trusted" and "untrusted" communication, as expressed in the different semantics of the external write instruction to different tapes.

A third use is made by the definition of resource-bounded computation (Section 3.2). There, only information written on the input tape is used in the calculation of the allowed runtime.

We note that these three uses of the distinction among different communication methods are, in principle, orthogonal to each other. The specific "bundling" of attributes with tapes is rather ad hoc, and is done for convenience only. For instance, the fact that the same tape is used to represent communication with peers and also to represent an untrusted medium is not essential. It only makes sense since in typical systems the various parties in a protocol instance are physically separated and connected via a network. Similarly, the fact that the tape used to represent information coming from a subroutine program also models trusted communication is not essential, and merely represents typical systems of interest. A more general modeling would allow for full separation of these interpretations, or "attributes" of tapes. That is, each external write instruction would specify, in addition to the tape to be written to, whether the code of the receiver is to be verified as a precondition to the writing operation, whether the identity and code of the sender is to be included, and whether the incoming data should be used towards determining the allowed runtime of the recipient. In the interest of simplicity, we choose not to provide such level of generality.

Jumping to the next received message. Recall that Definition 1 allows an ITM to move, in a single instruction, the reading head on each of the three incoming data tapes to the beginning of the next incoming message. At first, this instruction seems superfluous: Indeed, why not let the ITM simply move the head in the usual way, namely cell by cell?

The reason is that such an instruction becomes necessary in order to maintain a reasonable notion of resource-bounded computation in a heterogeneous and untrusted network, where the computational powers of participants vary considerably, and in addition some participants may be adversarial. In such a system, powerful participants may try to "overwhelm" less powerful participants by simply sending them very long messages. In reality, such an "attack" can be easily thwarted by having parties simply "drop" long messages, namely abort attempt to interpreted incoming messages that become too long. However, without a "jump to the next message" instruction, the ITM model does not allow such an abortion, since the reading head must be moved to the next incoming message in a cell-by-cell manner. (There are of course other ways in which powerful parties may try to "overwhelm" less powerful ones. But, with respect to these, the ITM model seems to adequately represent reality.)

We remark that the above discussion exemplifies the subtleties involved with modeling systems of ITMs. In particular, the notions of security in subsequent sections would have different technical meaning without the ability to jump to the beginning of the next incoming message. A similar phenomenon has been independently observed in [P06] in the context of Zero-Knowledge protocols.

Making the identities available to the code. In contrast with other formalisms (such as [DKMR05, K06]), The present formalism allows ITMs to read and use their identities, which are guaranteed by the model to be globally unique. Indeed, providing parties with identities that are guaranteed to be unique is a strong and very useful guarantee. In fact, in some cases having unique identities is *essential* for a meaningful solution. (This fact is exemplified in [LLR02] for the basic tasks of broadcast and Byzantine agreement.)

We provide unique identities in order to facilitate representing protocols that used identities. Still, it is of course possible to study within the present framework protocols that do not use the identities given in the model, by explicitly considering only protocols that ignore the identities.

We also note that there are a number of practical methods for guaranteeing global uniqueness of identities. One such way is to have each identity contain a component that's chosen at random from a large enough domain. Alternatively, one can use a hierarchical encoding where each new identity is pair (invoker ID, new ID). These methods can be regarded as ways to implement the abstraction of unique identities.

On the SID mechanism. As argued in the preamble to the definition of protocol instances (Section 3.1.2), the SID mechanism provides a relatively simple and flexible way to delineate protocol instances in a dynamically changing system. It also allows capturing, within the formal model, the process of "creating an instance of a protocol" in a dynamic and distributed way.

One basic aspect of this process is that some sort of agreement or coordination between the entities that invoke the participants of a protocol instance is needed. The SID mechanism postulates that this agreement take the form of agreeing on a joint identifier, namely the SID. We briefly sketch three common methods for reaching such agreement, and then point out some possible relaxations.

One method is to have the SID of the instance determined in advance (potentially as a function of the SID of the calling protocol and other run-time parameters). This method is natural when there is already some prior coordination between the entities that invoke the participants of a protocol instance, say when the protocol instance in question is a subroutine in a larger protocol.

A second method is to design the protocol so that all the ITIs in a protocol instance (except for the first one) are invoked via incoming messages from other ITIs in that instance, rather than via inputs from other protocol instances. That is, the first activation of each party occurs due to a message from another party of that protocol instance, rather than due to an input or a message from another protocol instance. Furthermore, the invoking ITI will set the SID of the invoked ITI to be the same as its own SID. This way, a multi-party instance of a protocol is created *without any prior coordination and without running an agreement protocol.* (The ideal authentication and secure message transmission functionalities from Section 1.3 are written in this manner, thus eliminating the need for prior agreement on the SID.) We note that real-life implementation of protocols that are written in this manner would require each participant to locally make sure that no two protocol sessions it participates in have the same SID.

A third alternative is to run some agreement protocol among the parties in order to determine a joint SID. This method is viable in situations where there is no prior coordination among the entities that invoke the various ITIs in a protocol instance, and the parties wish to jointly determine the SID. (See [BLR04] for a protocol and more discussion.) In this case, one convenient way to determine the SID of a protocol instance is to let it be the concatenation of the PIDs of some or all of the parties in this instance.

Finally we remark that it is possible to formulate alternative conventions that allow the SIDs of the parties in a protocol instance to be related in some other way, rather than being equal. Such a more general convention may allow more loose coordination between the ITIs in a protocol instance. (For instance, one may allow the participants to have different SIDs, and only require that there exists some global function that, given a state of the system and a pair of SIDs, determines whether these SIDs belong to the same instance.) Also, SIDs may be allowed to change during the course of the execution. However, such mechanisms would further complicate the model, and the extra generality obtained does not seem essential for our treatment.

On the order of activations. Recall the order of activations of ITMs in an execution of a system: Once an ITI μ completes its activation, the (single) ITI on whose tapes μ wrote is activated next; if μ didn't write to any other ITI then the initial ITI is activated. This simple order is relatively easy to argue about, thus simplifying the analysis of protocols. (In particular, it can be simulated efficiently on a standard Turing machine.)

However, one might wonder whether this simple order adequately represents concurrent systems. Indeed, the model postulates *sequential* activations of ITMs; this stands in contrast with the physical nature of distributed systems, where computations take place in multiple physically separate places at the same time. Furthermore, the specified order does not in itself guarantee any fairness or liveness properties.

We claim however that this order does provide a sound basis for the study of concurrent systems in general, and their security properties in particular. First we claim that, as long as individual activations represent computations that are relatively short comparing to the communication time, this sequential execution model actually provides "pseudo-concurrency" that is a reasonable approximation of "true concurrency". Furthermore we claim that, in spite of its simplicity, the present order is actually sufficient for representing liveness, synchrony and fairness properties. This is exemplified in Section 1.3. Also, as we'll see, this order suffices for representing adversarial scheduling which, on the one hand, is computationally bounded, and on the other hand can adaptively depend on the adversarial view of the execution so far.

Still, it should be kept in mind that sequential scheduling is only an abstraction of physical reality, and that other models exist where concurrency aspects of distributed systems are represented in a more direct way, often by considering non-deterministic scheduling and then quantifying over all possible schedules (see e.g. [Ly96, M89, LMMS98]). Such models are indeed more expressive in representing the scheduling and concurrency aspects of protocols. Some models that combine non-deterministic treatment of concurrency with cryptographic modeling are discussed in Appendix A.

The control function as an ITM. The control function is a convenient abstraction: First, as argued earlier, the control function can be regarded as a generalization of the more traditional notion of fixed "communication channels" or "ports". Second, the control function allows separating the definition of the basic communication model from definitions of security which are built on top of such a model. In particular, the definition of security (Section 4) defines the model for protocol execution by specifying a specific control function. Third, the control function can be used to represent models where the order of activations is different than here, such as [BPW04, DKMR05].

We note that an alternative and equivalent formulation of a system of ITMs might replace the control function by a special-purpose "router ITM" C that controls the flow of information between ITIs. Specifically, in this formulation the external input to the system is written on the input tape of C. Once activated for the first time, C copies its input to the input tape of the initial ITM I. From now on, all ITIs are allowed to write only to the incoming communication tape of C, and C is allowed to write to any externally writable tape of anther ITI. In simple (non-extended) systems, C always writes the requested value to the requested tape of the requested recipient, as long as the operation is allowed. In extended systems, C may write arbitrary values to the externally writable tapes of ITIs.

Deleting ITIs. The definition of a system of ITMs does not provide any means to "delete" an ITI from the system. That is, once an ITI is invoked, it remains present in the system for the rest of the execution, even after it has halted. In particular, its identity remains valid and "reserved" throughout. If a halted ITI is activated, it performs no operation and the initial ITI is activated next. The main reason for this convention is to avoid ambiguities in addressing of messages to ITIs.

3.2 Polynomial time ITMs and systems; Parameterized systems

We adapt the standard notions of "resource bounded computation" to the distributed setting considered in this work. This requires accommodating systems with dynamically changing number of components and communication patterns, and where multiple protocols co-exist. As usual in cryptography, where universal statements on the capabilities of *any feasible computation* are key, notions of security depend in a strong way on the precise formulation of resource bounded computation. As we'll see, current formulations do not behave well in a dynamically changing distributed setting such as the one considered in this work. We thus propose an extension that seems adequate within the present model.

Before proceeding with the definition itself, we first note that the notion of "resource bounded computation" is typically used for two quite different purposes. One is the study of *efficient algorithms.* Here we'd like to examine the number of steps required as a function of the complexity of the input, often interpreted as the input length. Another purpose is bounding the power of *feasible computation,* often for the purpose of security. Here we typically do not care whether the computation is using "efficient algorithms"; we are only concerned with what can be done within the given resource bounds. The notion developed here should be meaningful for both interpretations. In addition, the definition of security, formulated later, will in fact require presenting an *efficient transformation* from one *feasible computation* to another. The notion of resource bounded computation should allow making meaningful statements of that sort.

Let $T : \mathbf{N} \to \mathbf{N}$. Traditionally, a Turing machine is said to be T-bounded if, given any input of length n, it halts within at most $T(n)$ steps. There are several ways to generalize this notion to the case of ITMs. One option is to require that each activation of of the ITM completes within $T(n)$ steps, where n is either the length of the current incoming message, or, say, the overall length of incoming messages on all externally writable tapes ot the ITM. However, this option does not bound the overall number of activations of the ITM; this allows a system of ITMs to have unbounded executions, thus unbounded "computing power", even when all its components are resource bounded. This does not seem to capture the intuitive concept of resource bounded distributed computation.

A natural alternative is thus to let $T(n)$ bound the overall number of steps taken by the ITM since its invocation. But what should n be, in this case? One option is to let n be the overall length of incoming messages on all externally writable tapes of the ITM. However, this would still allow a situation where a system of ITMs, all of whose components are $T(n)$-bounded, consumes an unbounded number of resources. This is so since ITIs may send each other messages of repeatedly increasing lengths. In [GMRa89] this problem was solved by setting n to be the length of the input only. Indeed, in the [GMRa89] setting, where ITMs cannot write to input tapes of each other, this solution is adequate. However, in our setting no such restrictions exist; thus, when n is set to the overall length of the input received so far, infinite runs of a systems are possible even if all the ITIs are $T(n)$-bounded. Furthermore, infinite "chains" of ITIs can be created, where each ITI in the chain invokes the next one, again causing potentially infinite runs.

We prevent this "infinite runs" problem via the following simple mechanism: We define n to be the overall length of the input received so far, i.e. the number of bits written on the ITI's input tape, *minus the overall number of bits written by the ITI to the input tapes of other ITIs.* As we'll see, this provision allows guaranteeing that, for all "reasonable" functions T, the overall number of steps taken in a system of ITMs which are all T-bounded is finite. In fact, this number is bound by $T(n)$, where n is the length of the initial input to the system. Intuitively, this provision treats the characters written on the input tape as "tokens" that "buy" runtime. An ITI receives tokens on its input tape, and gives out tokens to other ITI by writing on their input tapes. This way, it is guaranteed that the number of tokens in the system remains unchanged, even if ITIs are generated dynamically and write on the input tapes of each other. In Section 3.2.1 we briefly mention some other potential formulations and their shortcomings.

Definition 2 (T-bounded, PPT) *Let $T : \mathbf{N} \rightarrow \mathbf{N}$. An ITM M is* locally T-bounded *if, at any point during an execution of M (namely, in any configuration of M), the overall number of computational steps taken by M so far is at most $T(n)$, where $n = n_I - n_O$, n_I is the overall number of bits written so far on M's input tape, and n_O is the number of bits written by M so far to input tapes of ITM instances.*

If M is locally T-bounded, and in addition either M does not make external write requests, or each external write request specifies a recipient ITM which is T-bounded, then we say that M is T-bounded.

M is PPT *if there exists a polynomial p such that M is p-bounded. A protocol is PPT if it is PPT as an ITM.*

Recall that $T : \mathbf{N} \rightarrow \mathbf{N}$ is concave if $T(n + n') \geq T(n) + T(n')$ for all n, n'. We have:

Proposition 3 *Let $T : \mathbf{N} \rightarrow \mathbf{N}$ be a concave increasing function. If the initial ITM in a system (I, C) is T-bounded, and in addition the control function C is computable in time $T'(\cdot)$, then an execution of the system can be simulated on a single (non-interactive) Turing machine M, which takes for input the initial input x, and runs in time $O(T(|x|)T'(T(|x|)))$. In particular, if both I and C are PPT then so it M. The same holds also for extended systems of ITMs, as long as all the ITMs invoked are T-bounded.*

Proof: We first claim that the overall number of configurations in an execution of a system (I, C) where I is T-bounded is at most $T(|x|)$, where x is the initial input of I. As mentioned above, this can be seen by treating the bits written on the input tapes of ITIs as "tokens" that give runtime. Initially, there are $|x|$ tokens in the system. The tokens are "passed around" between ITIs, but their number remains unchanged throughout. More formally, recall that an execution of a system of ITMs consists of a sequence of activations, where each activation is a sequence of configurations of the active ITI. Thus, an execution is essentially a sequence of configurations of ITIs. Consider an execution, and et m_i be the set of ITIs that were active up till the ith configuration in the execution. For each $\mu \in m_i$ let $n_{\mu,i}$ be number of bits written on the input tape of ITI μ at the last configuration where it was active before the ith configuration in the execution, minus the overall number of bits written by μ to other ITIs in all previous configurations. Since I is T-bounded we have that μ is also T-bounded, namely for any i, the number of steps taken by each $\mu \in m_i$ is at most $T(n_{\mu,i})$. It follows that $i = \sum_{\mu \in m_i}(\#\text{ steps taken by }\mu) \leq \sum_{\mu \in m_i} T(|n_{\mu,i}|)$. By concavity of T we have that $i \leq \sum_{\mu \in m_i} T(n_{\mu,i}) \leq T(\sum_{\mu \in m_i} n_{\mu,i})$. However, $\sum_{\mu \in m_i} n_{\mu,i} \leq |x|$. Thus $i \leq T(|x|)$.

The machine M that simulates the execution of the system (I, C) simply writes all the configurations of $I, C)$ one after the other, until it reaches a halting configuration of I. It then accepts if this configuration accepts. To bound the runtime of M, it thus remains to bound the time spent on evaluating the control function C. However, C is evaluated at most $T(|x|)$ times, on inputs of length at most $T(|x|)$ each. The bound follows. □

We note that the control functions of all the systems in this work run in linear time.

Parameterized systems. The definition of T-bounded ITMs guarantees that an execution of a system of bounded ITMs completes in bounded time. However, it does not provide any guarantee regarding the relative computing times of different ITMs in a system. To define security of protocols we will want to consider only systems where the allowed computing times of parties do not deviate too widely. To do that, we will assume that all parties know a common value, called a security parameter, that will be taken into account when determining the allowed runtime. More specifically, we say that a system of ITMs is parameterized with security parameter k if the initial input of the initial ITI starts with the string 1^k, and furthermore, upon invoking each new ITI μ, the invoking ITI writes the string 1^k on the input tape of μ.

Jumping ahead, we note that the security parameter is used not only to guarantee some minimum runtime for parties, but also to provide some basic coordination between the parties regarding the desired "level of security". Also, we will be interested in the behavior of systems when the length of the initial input is at most some function of the security parameter, say it is polynomial in the security parameter.

3.2.1 Discussion

We discuss some aspects of Definition 2 and mention some related definitional approaches.

Recognizing PPT ITMs. One general concern regarding notions of PPT Turing machines is how to decide whether a given ITM is PPT. The standard way of getting around this potentially undecidable problem is to specify a set of rules on encodings of ITMs such that: (a) it is easy to verify whether a given string obeys the rules, (b) all strings obeying these rules encode PPT ITMs, and (c) for essentially any PPT ITM there is a string that encodes it and obeys the rules. If there exists such a set of rules for a given notion of PPT, then we say that the notion is *efficiently recognizable.*

It can be readily seen that the notion of PPT in Definition 2 is efficiently recognizable. Specifically, an encoding σ of a *locally* PPT ITM will first specify an exponent c. It is then understood that the ITM encoded in σ counts its computational steps and halts after n^c steps. An encoding of a PPT ITM will guarantee in addition that that all the codes specified by the external write operations are also $n^{c'}$-bounded with an exponent $c' \leq c$. These are simple conditions that are straightforward to recognize.

Imposing an overall bound on the running time. Recall that it does not suffice in it itself to simply bound the runtime of each individual activation of an ITI by some function of the length of the contents of the externally writable tapes. This is so since, as discussed prior to Definition 2, we might still have unbounded executions of systems even when all the ITMs are bounded. Definition 2 gets around this problem by making a restriction on the *overall* number of steps taken by the ITI so far. An alternative approach might be to directly impose an overall bound on the runtime of the system. For instance, one can potentially bound the overall number of bits that are externally written in the execution. This approach seems attractive at first since it is considerably simpler; it also avoids directly "linking" the runtime in an activation of an ITM to the run-times in previous activations of this ITM. However this approach has a severe drawback: It causes an execution of a system to halt at a point which is determined by the overall number of steps taken by the system, rather than by the local behavior of the last ITI to be activated (namely the initial ITI). This provides an "artificial" way for the initial ITI to obtain global information on the execution via the timing in which the execution halts. (For instance, the initial ITI I can start in a rejecting state, and then pass control to another ITI μ. If I ever gets activated again, it moves to an accepting state. Now, whether I gets activated again depends only on whether the computation carried out by μ, together with the ITIs that μ might have invoked, exceeds the allotted number of steps, which in turn may be known to I. Thus, we have that whether I accepts depends on information that should not be "legitimately available" to I in a distributed system.) Jumping ahead, we note that this property would cause the notions of security considered in the rest of this work to be artificially restrictive. Specifically, the environment would now be able to distinguish between two executions as soon as the overall number of steps in the two executions differs even by one operation. In contrast, we would like to consider two systems equivalent from the point of view of the environment even in cases where the overall number of computational steps and communicated bits in the systems might differ by some polynomial amount.

Bounding the runtime by a function of the security parameter alone. Another alternative way to define resource bounded ITMs is to consider parameterized systems as defined above, and then restrict the number of steps taken by each ITI in the computation by a function of the security parameter *alone.* That is, let the overall number of steps taken by each ITI in the system be bounded by $T(k)$, where k is the security parameter. This formulation is actually quite popular; In particular, it is the notion of choice in earlier versions of this work as well as in [C00, PW00, BPW04, BPW04a, MMS03, C+05].

Bounding the runtime this way is simpler than the method used here. It also allows proving a proposition akin to Proposition 3. However, it has a number of drawbacks. First, it does not allow capturing algorithms and protocols which work for any input size, or alternatively work for any number of activations. For instance, any signature scheme that's PPT in the security parameter alone can only sign a number of messages that's bounded by a fixed polynomial in the security parameter. Similarly, it can only sign messages whose length is bounded by a fixed polynomial in the security parameter. In contrast, standard definitions of cryptographic primitives such as signature schemes, encryption schemes, or pseudorandom functions require schemes to handle a number of activations that's determined by an arbitrary PPT adversary, and thus cannot be bounded by any specific polynomial in the security parameter. Consequently, bounding the runtime by a fixed function of the security parameter severely restricts the set of protocols and tasks that can be expressed and analyzed within the framework.[2]

Furthermore, when this definition of bounded computation is used, security definitions are inevitably weaker, since the standard quantification over "all PPT adversaries" fails to consider those adversaries that are polynomial in the length of their inputs but not bounded by a polynomial in the security parameter. In fact, there exist protocols that are secure against adversaries that are PPT in the security parameter, but insecure against adversaries that are PPT in the length of their inputs (see e.g. the separating example in [HU05]).

Another drawback of bounding the runtime by a fixed function of the security parameter is that it does not allow taking advantage of the universality of computation and the duality between machines and their encodings. Let us elaborate, considering the case of PPT ITMs: When the runtime can vary with the length of the input, it is possible to have a single PPT ITM U that can "simulate" the operation of *all* PPT ITMs, when given sufficiently long input. (As the name suggests, U will be the universal Turing machine that receives the description of the ITM to be simulated, plus sufficiently long input that allows completing the simulation.) This universality is at the heart of the notion of "feasible computation". Also, this property turns out to be useful in gaining assurance in the validity of the definition of security, defined later in this work.

Bounding the runtime of ITMs by a function of the security parameter alone does not seem to allow for such a natural property to hold. Indeed, as discussed in Section 4.4, some of the properties of the notion of security defined here no longer hold when the runtime of ITMs is bounded this way.

Thanks. We thank Oded Goldreich, Dennis Hofheinz, Ralf Küsters, Yehuda Lindell, Jörn Müller-Quade, Rainer Steinwandt and Dominic Unruh for very useful discussions on modeling PPT ITMs and systems, and for pointing out to us shortcomings of the definition of PPT ITMs in earlier versions of this work and of some other definitional attempts. Discussions with Dennis were particularly instructive.

4 Defining security of protocols

This section presents the main definition of this work, namely the definition of protocols that securely realize a given ideal functionality, as outlined in Section 2.2. Section 4.1 presents the basic computational model for executing distributed protocols. The general notion of protocol emulation and some variants are presented in Section 4.2. Ideal functionalities and the ideal protocol for carrying out a given functionality are presented in Section 4.3, followed by the definition of securely realizing an ideal functionality. Hybrid protocols are defined next. Finally, Section 4.4 presents several alternative formalizations of the definition of protocol emulation and demonstrates their equivalence to the main one.

4.1 The model of protocol execution

The model of protocol execution is defined in terms of a system of ITMs, as defined in Section 3. The model gives the adversary complete control over the communication between parties. This modeling is not aimed at representing some specific realistic setting for distributed computation. Rather, the

[2] We remark that the difference is not only "cosmetic." For instance, pseudorandom functions with respect to a number of queries that is bounded by a fixed polynomial in the security parameter can be constructed without computational assumptions, whereas the standard notion implies one-way functions.

Execution of protocol π with environment $\mathcal{Z}$ and adversary $\mathcal{A}$

An execution of protocol π with adversary $\mathcal{A}$ and environment $\mathcal{Z}$ is a run of an extended, parameterized system of ITMs as specified in Section 3, with initial ITM $\mathcal{Z}$, and the following control function:

1. $\mathcal{Z}$ may only *pass inputs* to other parties. The first ITI that $\mathcal{Z}$ passes input to is set to be the adversary. That is, the identity of this ITI is required to have a special value, say $id = 0$. The code of that ITI is set (by the control function) to be $\mathcal{A}$. All the other ITIs that $\mathcal{Z}$ passes input to are required to have the same SID, s, determined by $\mathcal{Z}$. The code of all these ITIs is set and their code is set (again, by the control function) to be π. Call this instance the main instance of π.

2. The adversary $\mathcal{A}$ can either pass output to $\mathcal{Z}$, or deliver messages to the incoming communication tape of any party (ITI) in the system other than $\mathcal{Z}$. There are no restrictions on the contents and sender identities of delivered messages, with the following exception: Messages of the form (`Corrupt`,id,p) can be delivered only once $\mathcal{A}$ receives an input (`corrupt`,id,p) from $\mathcal{Z}$. Here id is the identity of the recipient party, and p denotes potential additional parameters. These messages are used to model party corruption; see discussion in the text.

3. ITIs other than $\mathcal{Z}$ or $\mathcal{A}$ (including the parties invoked by $\mathcal{Z}$ and $\mathcal{A}$ and their descendants) can send messages to $\mathcal{A}$, and also pass inputs and outputs to any other ITI other than $\mathcal{A}$ and $\mathcal{Z}$. In addition, the main parties of the main instance of π (namely, the parties with code π and with $SID = s$) can pass outputs to $\mathcal{Z}$.

   The responses to (`corrupt`) messages are assumed to be specified within π and may vary according to the specific corruption model and the parameters included in the message.

Figure 5: A summary of the model for protocol execution

goal is to provide a basic platform on top of which one can define various communication models that correspond to realistic settings. Several such communication models are defined in Section 6.

The model is parameterized by three ITMs: the protocol π to be executed, an environment $\mathcal{Z}$ and an adversary $\mathcal{A}$. That is, given $\pi, \mathcal{Z}, \mathcal{A}$, the model for executing π is the following extended, parameterized system of PPT ITMs $(\mathcal{Z}, C_{\text{EXEC}}^{\pi,\mathcal{A}})$ (as defined in Section 3). The initial ITM in the system is the environment $\mathcal{Z}$. The control function $C_{\text{EXEC}}^{\pi,\mathcal{A}}$ is defined in the paragraphs below. Figure 5 presents a summary of the model. A graphical depiction appears in Figure 2 on page 10.

The input of the initial ITM $\mathcal{Z}$ represents some initial state of the environment in which the protocol execution takes place. In particular, it represent all the external inputs to the system, including the local inputs of all parties. The first ITI to be invoked by $\mathcal{Z}$ is set by the control function to be $\mathcal{A}$. In addition, as the computation proceeds, $\mathcal{Z}$ can invoke and pass inputs to an unlimited number of ITIs, subject to the restriction that all these ITIs have the same SID (which is chosen by $\mathcal{Z}$). The code of these ITIs is set by the control function to be π, regardless of the code specified by $\mathcal{Z}$. Consequently, all the ITIs invoked by $\mathcal{Z}$, except for $\mathcal{A}$, are parties in a single instance of π. We call this instance the main instance of π in this execution. Other than that, $\mathcal{Z}$ may not communicate with any other ITIs. That is, $\mathcal{Z}$ can not pass inputs to any ITI other than $\mathcal{A}$ or the main parties of the main instance of π, nor can any ITI other than these pass output to $\mathcal{Z}$.[3]

[3] Jumping ahead, we note that restricting the environment to invoke only a single instance of π considerably

The control function allows the adversary $\mathcal{A}$ to send messages to any ITI in the system other than $\mathcal{Z}$. In this case, we say that $\mathcal{A}$ delivers this message. (We use a different term for the delivery operation to stress the fact that sending by the adversary models actual delivery of the message to the recipient.) There need not be any correspondence between the messages sent by the parties and the messages delivered by the adversary. $\mathcal{A}$ may not pass input to any party, nor can it pass output to any party other than $\mathcal{Z}$.

$\mathcal{A}$ may also *corrupt* parties. Corruption of a party (ITI) with identity id is modeled via a special $(\texttt{corrupt}, id, p)$ message delivered by $\mathcal{A}$ to that ITI, w here p denotes potential additional parameters. The control function allows delivery of that message only if $\mathcal{A}$ previously received a special $(\texttt{corrupt}, id, p)$ from the environment $\mathcal{Z}$. (This last stipulation makes sure that the environment knows which parties are corrupted. This is important for the notion of security to make sense.)

Any ITI other than $\mathcal{Z}$ and $\mathcal{A}$, namely the parties and sub-parties of the main instance of π, as well as the ITIs invoked by $\mathcal{A}$, are allowed to pass inputs and outputs to any other ITI other than $\mathcal{Z}$ and $\mathcal{A}$. In addition, they can send messages to $\mathcal{A}$. (These messages may indicate an identity of an intended recipient ITI; but the adversary is not obliged to respect these indications.)

The response of the party or sub-party to a (corrupt) message is not defined in the general model; rather, it is left to the protocol. This allows capturing a variety of corruption methods within a single computational model. See more discussion and elaboration in Section 6.1. For concreteness let us specify here one prevalent corruption model, namely that of Byzantine party corruption. Here, once a party or a sub-party receives a (corrupt) message, it sends to $\mathcal{A}$ its entire current local state. Also, In all future activations, a corrupted ITI merely forwards the incoming information to $\mathcal{A}$ and follows $\mathcal{A}$'s instructions.[4]

Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ denote the random variable $\text{OUT}_{\mathcal{Z},C^{\pi,\mathcal{A}}_{\text{EXEC}}}(z)$, where k is the security parameter in use.

4.2 Protocol emulation

This section formalizes the general notion of emulating one protocol via another protocol, as sketched in Section 2.2. In preparation for the formal definition, we first recall the notions of probability ensembles and indistinguishability, and define a restricted class of environments, called *balanced environments.* We then define UC-emulation in its general form, followed by some variants, including statistical, perfect, and uniform-complexity emulation. Next we formulate several more quantitative notions of emulation, and assert a somewhat surprising quantitative property of UC emulation. The section concludes with a discussion of the transitivity properties of UC emulation.

Distribution ensembles and indistinguishability. A probability distribution ensemble $X = \{X(k,z)\}_{k\in\mathbf{N},x\in\{0,1\}^*}$ is an infinite set of probability distributions, where a distribution $X(k,z)$ is associated with each $k \in \mathbf{N}$ and $z \in \{0,1\}^*$. The ensembles considered in this work describe

simplifies the analysis of protocols, since it restricts attention to a single, "stand-alone" protocol execution. An alternative formulation would allow $\mathcal{Z}$ to invoke arbitrary ITIs with multiple different SIDs. This more general (but more complex) formulation captures a meaningful extension of the model. See more details in [CDPW07].

The reason for having the control function set the code of the ITIs invoked by $\mathcal{Z}$ to $\mathcal{A}$ and π, respectively, regardless of $\mathcal{Z}$'s choice, is to enable discussing environments that do not a priori know which adversary and protocol they are interacting with. Such ability is essential for the notion of protocol emulation.

[4]Note that, even though the model lets $\mathcal{A}$ corrupt only one ITI in an activation, effectively it can corrupt multiple parties "at once", since the behavior of the corrupted ITI upon corruption makes sure that $\mathcal{A}$ is activated again immediately after each corruption.

outputs of computations where the parameter z represents input, and the parameter k represents the security parameter. As we'll see, it will suffice to restrict attention to binary distributions, i.e. distributions over $\{0,1\}$.

Definition 4 *Two binary probability distribution ensembles X and Y are* indistinguishable *(written $X \approx Y$) if for any $c, d \in \mathbf{N}$ there exists $k_0 \in \mathbf{N}$ such that for all $k > k_0$ and all $z \in \cup_{\kappa \leq k^d}\{0,1\}^\kappa$ we have:*

$$|\Pr(X(k,z)=1) - \Pr(Y(k,z)=1)| < k^{-c}.$$

The probability distribution ensembles considered in this work represent outputs of systems of ITMs, namely outputs of environments. More precisely, we consider ensembles of the form $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \stackrel{\text{def}}{=} \{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$. It is stressed that Definition 4 considers the distributions $X(k,z)$ and $Y(k,z)$ only when the length of z is polynomial in k. This essentially means that we consider only environments that set the security parameter to be some polynomial fraction of the length of their input.[5]

Balanced environments. In order to keep the notion of protocol emulation from being unnecessarily restrictive, we need to restrict attention to environments that satisfy some basic conditions regarding the lengths of inputs given to the parties. Recall that we have already restricted ourselves to parameterized systems where the length of input to each party must be at least the security parameter, and where the security parameter is a polynomial fraction of the length of input to the environment. However, this restriction does not limit the relative lengths of the inputs that the environment provides to the adversary and the parties; the difference can be any arbitrary polynomial in the security parameter, and the ratio can be arbitrary. Consequently, the model still allows the environment to create situations where the input length to the protocol, hence the protocol's complexity and communication complexity, are arbitrarily large relative to the input length and complexity of the adversary. Such situations seem unnatural; for instance, with such an environment no polytime adversary can deliver even a fraction of the protocol's communication. Indeed, it turns out that if we allow such situations then the definition of protocol emulation below (Definition 5) becomes overly restrictive.[6]

To avoid such situations, we wish to consider only environments where the amount of resources given to the adversary (namely, the length of the adversary's input) is at least some fixed polynomial fraction of the amount of resources given to the protocol. To be concrete, we consider only environments where, at any point in time during the execution, the overall length of the inputs given by $\mathcal{Z}$ to the parties of the main instance of π is at most k times the the length of input to the adversary. We call such environments balanced. It is stressed that the input of the adversary can still be arbitrarily (but polynomially) long relative to the input of the parties.

We finally turn to the definition of protocol emulation:

[5]In previous versions of this work, as well as in other works, the definition of indistinguishability considered the case where a can be of arbitrary length regardless of k. That formulation made sense since there the runtime of ITMs was measured in terms of the security parameter alone.

[6]The definition requires that any adversary be "simulatable" given a comparable amount of resources, in the sense that no environment can tell the simulated adversary from the real one. When this requirement is applied to environments that give the adversary much less resources than to the protocol, the simulation may not even be able to run the code of the protocol in question.

Definition 5 *Let π and ϕ be PPT protocols. We say that π* UC-emulates *ϕ if for any PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any balanced PPT environment $\mathcal{Z}$ we have:*

$$\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}. \tag{1}$$

On statistical and perfect emulation. Definition 5 can be extended to the standard notions of statistical and perfect emulation (as in, say, [c00]). That is, when $\mathcal{A}$ and $\mathcal{Z}$ are allowed unbounded complexity, and the simulator $\mathcal{S}$ is allowed to be polynomial in the complexity of $\mathcal{A}$, we say that π statistically UC-emulates ϕ. If in addition the two sides of (1) are required to be identical then we say that π perfectly UC-emulates ϕ. Another variant allows $\mathcal{S}$ to have unlimited computational power, regardless of the complexity of $\mathcal{A}$; however, this variant provides a weaker security guarantee, see discussion in [c00].

On security with respect to closed environments. Definition 5 considers "open environments", namely environments that take input (of some polynomial length) that was generated in an arbitrary way, perhaps not even recursively. Alternatively, one may choose to consider only "closed environments", namely environment that do not receive meaningful external input. Here the inputs given to the protocol are the result of some uniform, polynomial time process. This notion of security can be captured by considering only environments whose external input contains no information other than its length, e.g. it is 1^n for some n. Such environments would choose the inputs of the parties based on some internal stochastic process.

We note that the former notion corresponds naturally to considering environments and adversaries that are non-uniform polynomial time, whereas the latter notion corresponds to considering uniform polynomial time environments and adversaries.

More quantitative notions of emulation. The notion of protocol emulation as defined above only provides a "qualitative" measure of security. That is, it essentially only gives the guarantee that "any feasible attack against π can be turned into a feasible attack against ϕ," where "feasible" is interpreted broadly as "polynomial time". We formulate more quantitative variants of this definition. We note that, besides being informative in of itself, the material here will prove instrumental in later sections.

We quantify two parameters: the emulation slack, meaning the probability by which the environment distinguishes between the interaction with π from the interaction with ϕ, and the simulation overhead, meaning the difference between the complexity of the given adversary $\mathcal{A}$ and that of the constructed adversary $\mathcal{S}$. Recall that an ITM is T-bounded if the function bounding its running time is $T(\cdot)$ (see Definition 2), and that a functional is a function from functions to functions. Then:

Definition 6 *Let π and ϕ be protocols and let ϵ, g be functionals. We say that π UC-emulates ϕ with* emulation slack *ϵ and* simulation overhead *g (or, in short, π (e,g)*-UC-emulates *ϕ), if for any polynomial $p_\mathcal{A}(\cdot)$ and any $p_\mathcal{A}$-bounded adversary $\mathcal{A}$, there exists a $g(p_\mathcal{A})$-bounded adversary $\mathcal{S}$, such that for any polynomial $p_\mathcal{Z}$, any $p_\mathcal{Z}$-bounded environment $\mathcal{Z}$, any large enough value $k \in \mathbf{N}$ and any input $x \in \{0,1\}^{p_\mathcal{Z}(k)}$ we have:*

$$|\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}(k,x) - \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,x)| < \epsilon(p_A, p_\mathcal{Z})(k).$$

Including the security parameter k is necessary when the protocol depends on it. Naturally, when k is understood from the context it can be omitted. A more concrete variant of Definition

6 abandons the asymptotic framework and instead concentrates on a specific value of the security parameter k:

Definition 7 *Let π and ϕ be protocols, let $k \in \mathbf{N}$, and let $g, \epsilon : \mathbf{N} \to \mathbf{N}$. We say that π (k, e, g)-UC-emulates ϕ if for any $t_{\mathcal{A}} \in \mathbf{N}$ and any adversary $\mathcal{A}$ that runs in time $t_{\mathcal{A}}$ there exists an adversary $\mathcal{S}$ that runs in time $g(t_{\mathcal{A}})$ such that for any $t_{\mathcal{Z}} \in \mathbf{N}$, any environment $\mathcal{Z}$ that runs in time $t_{\mathcal{Z}}$, and any input $x \in \{0,1\}^{t_{\mathcal{Z}}}$ we have:*

$$|\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}(k,x) - \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,x)| < \epsilon(k, t_A, t_{\mathcal{Z}}).$$

It is stressed that Definition 7 still quantifies over all environments and adversaries of all complexities. One can potentially formulate a definition that parameterizes also the run-times of the environment, adversary and simulator. That is, the the quantifies only over environments and adversaries that have specific complexity. It should be noted, however, that such a definition would be considerably weaker than definition 7, since it guarantees security only for adversaries and environments that are bounded by specific run-times. Furthermore, both the protocols and the simulator can depend on these run-times. In contrast, Definition 7 bounds the specified parameters for any arbitrarily complex environment and adversary. Consequently, while the UC theorem can be easily adapted to the formulations of Definitions 6 and 7, a definition that parameterizes complexity of the environment and adversary does not seem to guarantee universal composability.

The simulation overhead is always additive. An interesting property of the notion of UC-emulation is that the simulation overhead can be always bounded by an additive polynomial factor that depends only on the protocols in question, and is independent of the adversary. That is:

Claim 8 *Let π and ϕ be protocols such that π UC-emulates ϕ with emulation slack ϵ and simulation overhead g, as in Definition 6. Then there exists a polynomial α such that π UC-emulates ϕ with emulation slack ϵ and simulation overhead $g'(p_{\mathcal{A}})(\cdot) = p_{\mathcal{A}}(\cdot) + \alpha(\cdot)$.*

Said otherwise, if π (ϵ, g)-UC-emulates ϕ then it is guaranteed that the overhead of running $\mathcal{S}$ rather than $\mathcal{A}$ can be made to be is at most an additive polynomial factor $\alpha(\cdot)$ that depends only on π and ϕ. Furthermore, this can be done with no increase in the emulation slack. We call $\alpha(\cdot)$ the intrinsic simulation overhead of π with respect to ϕ.

The proof of Claim 8, while not immediate from the definition of the simulation overhead, follows as an easy corollary from the proof of Claim 10 below. We thus postpone the proof of Claim 8 till after the proof of Claim 10.

On the transitivity of emulation. It is easy to see that if protocol π_1 UC-emulates protocol π_2, and π_2 UC-emulates π_3, then π_1 UC-emulates π_3. Moreover, if π_1 (e_1, g_1)-UC-emulates π_2, and π_2 (e_2, g_2)-UC-emulates π_3, then π_1 $(e_1+e_2, g_2 \circ g_1)$-UC-emulates π_3. (Here e_1+e_2 is the functional that output the sum of the outputs of e_1 and e_2, and $\circ$ denotes composition of functionals.) Transitivity for any number of protocols $\pi_1, ..., \pi_n$ follows in the same way. Note that if the number of protocols is not bounded by a constant then the complexity of the adversary may no longer be bounded by a polynomial. Still, when there is an overall polynomial bound on the intrinsic simulation overheads of each π_i w.r.t. π_{i+1}, Claim 8 implies that the simulation overhead remains polynomial as long as the number of protocols is polynomial. Similarly the emulation slack remains negligible as long as the number of protocols is polynomial. Finally, we stress that the question of transitivity of emulation should not be confused with the question of multiple *nesting* of protocols, which is discussed in Section 5.3.

4.3 Realizing ideal functionalities; Hybrid protocols

We now turn to applying the general machinery of protocol emulation towards one of the main goals of this work, namely defining security of protocols via realizing ideal functionalities.

Ideal functionalities. As discussed in Section 2, an ideal functionality represents the expected functionality of a certain task, or a protocol problem. This includes both "correctness", namely the expected input-output relations of uncorrupted parties, and "secrecy", or the acceptable leakage of information to the adversary. Technically, an ideal functionality $\mathcal{F}$ is an ITM as in Definition 1, with the following additional conventions. First, its input tape is expected to be written to by a number of ITIs; it also expects to write to the subroutine output tapes of multiple ITIs. This represents the fact that an ideal functionality behaves like a subroutine machine for a number of different ITIs (which are thought of as parties of some protocol instance). Next, the PID of an ideal functionality is set to $\perp$; This fact is used to distinguish ideal functionalities from other ITIs. In addition, an ideal functionality $\mathcal{F}$ expects all inputs to be written by ITIs whose SID is identical to the local SID of $\mathcal{F}$. Other inputs are ignored. The communication tape of $\mathcal{F}$ is used for messages coming from the adversary.

Typically, useful ideal functionalities will have some additional structure, such as specifying the response to party corruption requests by the adversary. However, to avoid cluttering the basic definition with unnecessary details, further restrictions and conventions regarding ideal functionalities are postponed to Section 6.

Ideal protocols. Let $\mathcal{F}$ be an ideal functionality. The ideal protocol for $\mathcal{F}$, denoted $\text{IDEAL}_{\mathcal{F}}$, is defined as follows. An instance of $\text{IDEAL}_{\mathcal{F}}$ with session ID sid consists of an ITI $\mathcal{F}_{(sid,\perp)}$ whose party ID is a special symbol $\perp$, plus ITIs with identities (sid, pid) is with $pid \neq \perp$. The ITI $\mathcal{F}_{(sid,\perp)}$ is called the ideal functionality. The other ITIs in the protocol instance are called dummy parties. Whenever a dummy party is activated with input v, it writes v onto the input tape of the ideal functionality $\mathcal{F}_{(sid,\perp)}$. (Recall that, according to the definition of a system of ITMs, $\mathcal{F}_{(sid,\perp)}$ is created at the first call to it.) Whenever a dummy party receives a value v on its subroutine output tape, it writes this value on the subroutine output tape of $\mathcal{Z}$.

Messages delivered by $\mathcal{A}$, including corruption messages, are ignored. The intention here is that, in the ideal protocol, the adversary should send corruption messages directly to the ideal functionality. The ideal functionality will then determine the effect of corrupting a party. One typical response would be to let the adversary know some or all of the inputs and outputs the party has received so far. Other, more global responses may include a reduction in the overall security guarantees, e.g. when more than a certain number of parties have been corrupted. See more discussion in Section 6.2. We sometime use the term dummy party for $\mathcal{F}$ to denote a party of an ideal protocol for $\mathcal{F}$. See Figure 3 on page 11 for a graphic depiction of the ideal protocol. Let $\text{IDEAL}_{\mathcal{F},\mathcal{A},\mathcal{Z}}(k,z)$ denote the random variable
$\text{EXEC}_{\text{IDEAL}_{\mathcal{F}},\mathcal{A},\mathcal{Z}}(k,z)$. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Realizing an ideal functionality. Protocols that realize an ideal functionality are defined as protocols that emulate the ideal protocol for this ideal functionality:

Definition 9 *Let $\mathcal{F}$ be an ideal functionality and let π be a protocol. We say that π* UC-realizes *$\mathcal{F}$ if π UC-emulates the ideal protocol for $\mathcal{F}$.*

Hybrid protocols. We define a special type of protocols, called hybrid protocols, where, in addition to communicating via the adversary in the usual way, the parties also make calls to instances of ideal functionalities. This is done in a straightforward way, by calling the corresponding instances of the ideal protocol for these functionalities. In other words, an $\mathcal{F}$-hybrid protocol π is a protocol that includes subroutine calls to $\text{IDEAL}_{\mathcal{F}}$, the ideal protocol for $\mathcal{F}$.

Recall that running $\text{IDEAL}_{\mathcal{F}}$ means invoking "dummy parties" for $\mathcal{F}$, which in turn invoke an instance of $\mathcal{F}$. This somewhat "indirect" access to $\mathcal{F}$ has the advantage that the calling parties have the freedom to specify the SIDs and PIDs of the dummy parties of $\mathcal{F}$ (namely, of the ITIs running $\text{IDEAL}_{\mathcal{F}}$). This way, different calling parties can generate dummy parties for $\mathcal{F}$ that have the same SID; these dummy parties will thus be part of the same instance of $\text{IDEAL}_{\mathcal{F}}$, and will interact with the same instance of $\mathcal{F}$. Furthermore, the generated SID and PID can be related to the SIDs and PIDs of the calling parties in any way.

Hybrid protocols are extended in the natural way to the case where the protocol makes use of multiple different ideal functionalities. A hybrid protocol with access to ideal functionalities $\mathcal{F}_1, ..., \mathcal{F}_n$ is called an $\mathcal{F}_1, ..., \mathcal{F}_n$-hybrid protocol.

4.4 Alternative formulations of UC emulation

We discuss some alternative formalizations of the definition of protocol emulation (Definition 5) and show that they are all equivalent to the main formalization.

On environments with non-binary outputs. Definition 5 quantifies only over environments that generate binary outputs. One may consider an extension to the models where the environment has arbitrary output; here the definition of security would require that the two output ensembles $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ (that would no longer be binary) be *computationally indistinguishable,* as defined by Yao [Y82] (see also [G01]). It is easy to see, however, that this extra generality results in a definition that is equivalent to Definition 5. We leave the proof as an exercise.

On deterministic environments. Since we allow the environment to receive an arbitrary external input of polynomial length, it suffices to consider only deterministic environments. That is, the definition that quantifies only over deterministic environments is equivalent to Definition 5. Again, we leave the proof as an exercise. Note however that this equivalence does *not* hold for the case of uniform-complexity security, i.e. when the environment only receives inputs of the form 1^n.

4.4.1 Emulation with respect to the dummy adversary

We show that Definition 5 can be simplified as follows. Instead of quantifying over all possible adversaries $\mathcal{A}$, it suffices to require that the ideal-protocol adversary $\mathcal{S}$ be able to simulate, for any environment $\mathcal{Z}$, the behavior of a specific and very simple adversary. This adversary, called the "dummy adversary", only delivers to parties messages generated by the environment, and delivers to the environment all messages generated by the parties. Said otherwise, we essentially show that the dummy adversary is "the hardest adversary to simulate", in the sense that simulating this adversary implies simulating all adversaries. Intuitively, the reason that the dummy adversary is the "hardest to simulate" is that it gives the environment full control over the communication. It thus leaves the simulator with very little "wiggle room."

More specifically, the dummy adversary, denoted $\mathcal{D}$, proceeds as follows. When activated with an incoming message m on its incoming communication tape, adversary $\mathcal{D}$ passes m as output to

$\mathcal{Z}$. (Recall that m already contains in it the identity of the sender.) When activated with an input (m, id, c) from $\mathcal{Z}$, where m is a message, id is an identity, and c is a code for a party, $\mathcal{D}$ delivers the message m to the party with identity id and code c. This in particular means that $\mathcal{D}$ corrupts parties when instructed by $\mathcal{Z}$, and passes all gathered information to $\mathcal{Z}$.

The above description of $\mathcal{D}$ is not necessarily PPT as in Definition 2. This is so since $\mathcal{D}$ may receive little or no input from $\mathcal{Z}$, while receiving long incoming communication from some other party. We rectify this in two steps. First, we bound the runtime of $\mathcal{D}$ by some arbitrary polynomial, say it is allowed twice as many operations as the length of its input. By itself, however, this stipulation is over-restrictive since it means that, with a dummy adversary, a protocol cannot send messages prior to receiving messages of comparable length. (This is so since the dummy adversary delivers any input given to it.) We thus let $\mathcal{D}$ distinguish between two types of input: One type of input is the one described above, to be forwarded to the appropriate party. The other type of input is of the form 1^m for some m. This input is ignored by $\mathcal{D}$; it is used only to provide $\mathcal{D}$ with extra runtime. [7]

Say that protocol π UC-emulates protocol ϕ with respect to the dummy adversary if there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{D},\mathcal{Z}}$. We show:

Claim 10 *Let π, ϕ be protocols. Then π UC-emulates ϕ according to Definition 5 if and only if it UC-emulates ϕ with respect to the dummy adversary.*

Proof: Clearly if π UC-emulates ϕ according to Definition 5 then it UC-emulates ϕ with respect to dummy adversaries. The idea of the derivation in the other direction is that, given direct access to the communication sent and received by the parties, the environment can run any adversary by itself. Thus quantifying over all environments essentially implies quantification also over all adversaries. More precisely, let π, ϕ be protocols and let $\mathcal{S}_{\mathcal{D}}$ be the adversary guaranteed by the definition of emulation with respect to dummy adversaries (that is, $\mathcal{S}_{\mathcal{D}}$ satisfies $\text{EXEC}_{\phi,\mathcal{S}_{\mathcal{D}},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{D}_{c_\pi},\mathcal{Z}}$ for all $\mathcal{Z}$.) We show that π UC-emulates ϕ according to Definition 5. For this purpose, given an adversary $\mathcal{A}$ we construct the adversary $\mathcal{S}$ as follows. $\mathcal{S}$ runs simulated instances of $\mathcal{A}$ and $\mathcal{S}_{\mathcal{D}}$. In addition:

1. $\mathcal{S}$ forwards any input from the environment to the simulated $\mathcal{A}$, and passes any output of $\mathcal{A}$ to the environment.

2. When the simulated $\mathcal{A}$ delivers a message m to an ITI with identity id and code c, $\mathcal{S}$ activates $\mathcal{S}_{\mathcal{D}}$ with input (m, id, c). Similarly, any output generated by $\mathcal{S}_{\mathcal{D}}$ is copied to the incoming communication tape of $\mathcal{A}$.

3. Whenever $\mathcal{S}_{\mathcal{D}}$ writes a message m on some tape of some ITI, $\mathcal{S}$ writes m to that tape of that ITI. Finally, when $\mathcal{S}$ obtains a message m on its incoming communication tape, it proceeds as follows: It first writes 1^m on the input tape of $\mathcal{S}_{\mathcal{D}}$; next (i.e., in the next activation) it writes m on the incoming communication tape of $\mathcal{S}_{\mathcal{D}}$.

A graphical depiction of the operation of $\mathcal{S}$ appears in Figure 6.

Analysis of $\mathcal{S}$. We first argue that $\mathcal{S}$ is PPT. The running time of $\mathcal{S}$ is dominated by the runtime of the $\mathcal{A}$ module plus the runtime of the $\mathcal{S}_{\mathcal{D}}$ module. When $\mathcal{S}$ has input of length n, the runtime

[7]Previous formulations of the dummy adversary ignored this aspect, thus resulting in an incorrect proof of Claim 10. We thank Ralf Küsters for pointing out these errors, along with similar errors in the notion of black-box simulation.

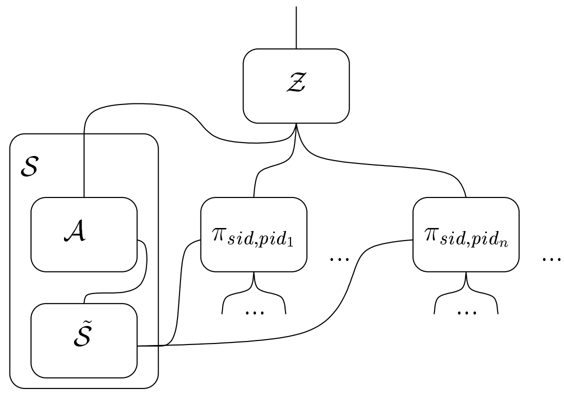

Figure 6: The operation of simulator $\mathcal{S}$ in the proof of Claim 10: Both $\mathcal{A}$ and $\mathcal{S}_{\mathcal{D}}$ are simulated internally by $\mathcal{S}$. The same structure represents also the operation of the shell adversary in the definition of black-box simulation (see Section 4.4.2).

of the $\mathcal{A}$ module is bounded by $p_{\mathcal{A}}(n)$, where $p_{\mathcal{A}}$ is the polynomial bounding the running time of $\mathcal{A}$. To bound the runtime of $\mathcal{S}_{\mathcal{D}}$, recall that $\mathcal{S}_{\mathcal{D}}$ gets inputs both from $\mathcal{A}$ and from $\mathcal{S}$ itself. The overall length of the first input is bounded by $p_{\mathcal{A}}(n)$. The length of the second input is bounded by the overall communication generated by ϕ. This quantity may in principle be unrelated to n, the length of $\mathcal{S}$'s input. However, since the environment is balanced, we have that the communication generated by π is at most $p_\pi(k * n)$, where p_π is the polynomial bounding the complexity of π. In addition, without loss of generality the length of the input coming from $\mathcal{A}$ can also be bounded by $p_\pi(k * n)$. We conclude that the polynomial bounding the runtime of $\mathcal{S}$ is at most $p_{\mathcal{S}}(\cdot) = p_{\mathcal{A}}(\cdot) + p_{\mathcal{S}_{\mathcal{D}}}(p_\pi(k * \cdot))$. Note that the right hand side summand is a polynomial that depends only on π and ϕ; this fact is used in the proof of Claim 8 below.

Next we assert the validity of $\mathcal{S}$. Assume for contradiction that there is an adversary $\mathcal{A}$ and environment $\mathcal{Z}$ such that $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \not\approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. We construct an environment $\mathcal{Z}_{\mathcal{D}}$ such that $\text{EXEC}_{\pi,\mathcal{S}_{\mathcal{D}},\mathcal{Z}_{\mathcal{D}}} \not\approx \text{EXEC}_{\pi,\mathcal{D},\mathcal{Z}_{\mathcal{D}}}$. Environment $\mathcal{Z}_{\mathcal{D}}$ runs an interaction between simulated instances of $\mathcal{Z}$ and $\mathcal{A}$. In addition:

1. All the inputs generated by $\mathcal{Z}$ to the adversary are forwarded to $\mathcal{A}$, and all of $\mathcal{A}$'s outputs are forwarded to $\mathcal{Z}$.

2. Whenever $\mathcal{Z}_{\mathcal{D}}$ receives an output value v from its adversary, $\mathcal{Z}_{\mathcal{D}}$ passes v to the simulated $\mathcal{A}$. Similarly, whenever the simulated $\mathcal{A}$ delivers a message m to some ITI, $\mathcal{Z}_{\mathcal{D}}$ instructs the external adversary to deliver message m to that ITI.

3. All inputs from $\mathcal{Z}$ to the parties of π are forwarded to the external parties, and all the outputs coming from the external parties are forwarded to $\mathcal{Z}$ as coming from the parties of π.

4. In addition, whenever $\mathcal{Z}_{\mathcal{D}}$ passes input of length m to some party, it first passes input $1^{p(m)}$ to the external adversary, where $p()$ is the maximum between the polynomials bounding the run times of ϕ and π. This makes sure that the external (dummy) adversary never "runs out of resources" for forwarding messages.

5. Finally, $\mathcal{Z}_{\mathcal{D}}$ outputs whatever the simulated $\mathcal{Z}$ outputs.

It can be readily verified that the ensembles $\text{EXEC}_{\pi,\mathcal{D},\mathcal{Z}_\mathcal{D}}$ and $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ are identical. In particular, $\mathcal{Z}_\mathcal{D}$ makes sure that $\mathcal{D}$ never halts due to insufficient runtime. Similarly, ensembles $\text{EXEC}_{\phi,\mathcal{S}_\mathcal{D},\mathcal{Z}_\mathcal{D}}$ and $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ are identical. In particular, in both cases $\mathcal{S}_\mathcal{D}$ simulates a situation where $\mathcal{D}$ never halts due to insufficient runtime. $\square$

From a technical point of view, emulation with respect to the dummy adversary is an easier definition to work with, since it involves one less quantifier, and furthermore it restricts the interface of the environment with the adversary to be very simple. Indeed, we almost always prefer to work with this notion. However, we chose not to present this formulation as the main notion of protocol emulation, since we feel is is somewhat less intuitively appealing than Definition 9. In other words, we find it harder to get convinced that this definition captures the security requirements of a given task. In particular, it looks farther away from the the basic notion of security in, say, [C00]. Also, it is less obvious that this definition has some basic closure properties such as transitivity.

Proof of Claim 8. Claim 8 states that if a protocol π UC-emulates protocol ϕ then there exists a polynomial $\alpha(\cdot)$ such that, for any adversary $\mathcal{A}$ whose running time is bounded by the polynomial $p_\mathcal{A}(\cdot)$, there is a simulator whose running time is bounded by $p_\mathcal{A}(\cdot)+\alpha(\cdot)$. The claim follows from the proof of Claim 10. Indeed, the proof of 10 shows how to construct, for any adversary $\mathcal{A}$, a valid simulator $\mathcal{S}_\mathcal{D}$, whose complexity is bounded by $p_\mathcal{A}(n)+\alpha(n)$, where $p_\mathcal{A}$ is the polynomial bounding the running time of $\mathcal{A}$ and $\alpha(\cdot)$ is a polynomial that depends only on π and ϕ. $\square$

4.4.2 Emulation with respect to black box simulation

Another alternative formulation of Definition 5 imposes the following technical restriction on the simulator $\mathcal{S}$: Instead of allowing a different simulator for any adversary $\mathcal{A}$, let the simulator have "black-box access" to $\mathcal{A}$, and require that the code of the simulator remains the same for all $\mathcal{A}$. Restricting the simulator in this manner does not seem to capture any tangible security concern. Still, in other contexts, e.g. in the classic notion of Zero-Knowledge, this requirement results in a strictly more restrictive notion of security than the definition that lets $\mathcal{S}$ depend on the description of $\mathcal{A}$, see e.g. [GK88, B01]. We show that in the UC framework security via black-box simulation is *equivalent* to the standard notion of security.

We formulate black box emulation in a way that keeps the overall model of protocol execution unchanged, and instead imposes restrictions on the operation of the simulator. Specifically, an adversary $\mathcal{S}$ is called a *shell simulator* if it operates as follows, given an ITM $\hat{\mathcal{S}}$ (called a black-box simulator) and a PPT adversary $\mathcal{A}$. $\mathcal{S}$ first internally invokes an instance of $\mathcal{A}$ and an instance of $\hat{\mathcal{S}}$. Next:

- Upon receiving an input from the environment, $\mathcal{S}$ forwards this input to $\mathcal{A}$. Any outgoing message generated by $\mathcal{A}$ is given as input to $\hat{\mathcal{S}}$. Instructions of $\hat{\mathcal{S}}$ regarding delivering messages to parties are carried out.

- Upon receiving an incoming message from some party, $\mathcal{S}$ forwards this incoming message to $\hat{\mathcal{S}}$. Outputs of $\hat{\mathcal{S}}$ are forwarded as incoming messages to $\mathcal{A}$, and outputs of $\mathcal{A}$ are outputted to $\mathcal{Z}$.

See graphical depiction of the operation of a black-box simulator in Figure 6 (substitute $\hat{\mathcal{S}}$ for $\mathcal{S}_\mathcal{D}$).

Let $\text{EXEC}_{\phi,\mathcal{S}^{\hat{\mathcal{S}},\mathcal{A}},\mathcal{Z}}$ denote the output of $\mathcal{Z}$ from an interaction with protocol ϕ and a shell adversary $\mathcal{S}$ that runs a black-box simulator $\hat{\mathcal{S}}$ and an adversary $\mathcal{A}$. Say that a protocol π UC-emulates protocol ϕ with black-box simulation if there exists a black-box simulator $\hat{\mathcal{S}}$ such that for any PPT adversary $\mathcal{A}$, the resulting shell adversary $\mathcal{S}$ is PPT, and furthermore, and any PPT environment $\mathcal{Z}$, we have $\text{EXEC}_{\phi,\mathcal{S}^{\hat{\mathcal{S}},\mathcal{A}},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. (It is stressed that $\hat{\mathcal{S}}$ need not necessarily be PPT as an ITM; only $\mathcal{S}$ needs to be PPT.) We show:

Claim 11 *Let π, ϕ be PPT multiparty protocols. Then π UC-emulates ϕ according to Definition 5 if and only if it UC-emulates ϕ with black-box simulation.*

Proof: The 'only if' direction follows from the definition. For the 'if' direction, notice that the simulator $\mathcal{S}$ in the proof of Claim 10 can be cast as a shell adversary with adversary $\mathcal{A}$ and the following black-box simulator $\hat{\mathcal{S}}$: $\hat{\mathcal{S}}$ runs $\mathcal{S}_{\mathcal{D}}$; in addition, before delivering a message of length m to the input tape of $\mathcal{S}_{\mathcal{D}}$, $\hat{\mathcal{S}}$ writes 1^m on the input tape of $\mathcal{S}_{\mathcal{D}}$. (Indeed, $\hat{\mathcal{S}}$ is not PPT as an ITM by itself. However, as argued in the proof of Claim 10, for any PPT adversary $\mathcal{A}$, the shell simulator $\mathcal{S}$ is PPT with runtime bounded by $p_{\mathcal{A}}(\cdot) + p_{\mathcal{S}_{\mathcal{D}}}(p_\phi(\cdot))$.) □

We remark that the present formulation of security via black-box simulation is considerably more restrictive than that of standard cryptographic modeling of black-box simulation. In particular, in the standard modeling S may query $\mathcal{A}$ in arbitrary ways. In contrast, here the communication between $\hat{\mathcal{S}}$ and $\mathcal{A}$ is restricted, in that $\hat{\mathcal{S}}$ cannot "reset" or "rewind" $\mathcal{A}$. Still, the present definition is equivalent to the plain (non black-box) notion of security.

We remark that the present formulation of black-box simulation is reminiscent of the notions of strong black-box simulation in [DKMR05] and in [PW00] (except for the introduction of the shell adversary). However, in these works this notion is not equivalent to the standard one, due to different formalizations of probabilistic polynomial time.

4.4.3 Letting the simulator depend on the environment

Consider a variant of Definition 5, where the simulator $\mathcal{S}$ can depend on the code of the environment $\mathcal{Z}$. That is, for any $\mathcal{A}$ and $\mathcal{Z}$ there should exist a simulator $\mathcal{S}$ that satisfies $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. Following [L03a], we call this variant security with respect to specialized simulators. We demonstrate that this variant is equivalent to the main definition (Definition 5).

Claim 12 *A protocol π UC-emulates protocol ϕ according to Definition 5 if and only if it UC-emulates ϕ with respect to specialized simulators.*

Proof: Clearly, if π UC-emulates ϕ as in Definition 5 then UC-emulates ϕ with respect to specialized simulators. To show the other direction, assume that π C emulates ϕ with respect to specialized simulators. That is, for any PPT adversary $\mathcal{A}$ and PPT environment $\mathcal{Z}$ there exists a PPT simulator $\mathcal{S}$ such that (1) holds. Consider the "universal environment" $\mathcal{Z}_u$ which expects its input to consist of $(\langle\mathcal{Z}\rangle, z, 1^t)$, where $\langle\mathcal{Z}\rangle$ is an encoding of an ITM $\mathcal{Z}$, z is an input to $\mathcal{Z}$, and t is a bound on the running time of $\mathcal{Z}$. Then, $\mathcal{Z}_u$ runs $\mathcal{Z}$ on input z for up to t steps, outputs whatever $\mathcal{Z}$ outputs, and halts. Clearly, machine $\mathcal{Z}_u$ is PPT. (in fact, it runs in linear time in its input length). We are thus guaranteed that there exists a simulator $\mathcal{S}$ for $\mathcal{Z}_u$ such that (1) holds. We claim that $\mathcal{S}$ satisfies (1) with respect to *any* PPT environment $\mathcal{Z}$. To see this, fix a PPT machine $\mathcal{Z}$ as in Definition 2, and let c be the constant exponent that bounds $\mathcal{Z}$'s running time. For each $k \in \mathbf{N}$ and $z \in \{0,1\}^*$, the distribution $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}(k,z)$ is identical to the distribution $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}_u}(k,z_u)$, where $z_u = (\langle\mathcal{Z}\rangle, z, 1^{c\cdot|z|})$. Similarly, the distribution $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ is identical to the distribution $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_u}(k,z_u)$. Consequently, for any $d \in \mathbf{N}$ we have:

$$\{\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^{\leq k^d}} \quad = \quad \{\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}_u}(k,z_u)\}_{k\in\mathbf{N},z_u=(\langle\mathcal{Z}\rangle,z\in\{0,1\}^{\leq k^d},1^{c\cdot|z|})}$$

$$\begin{aligned} &\approx \quad \{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_u}(k,z_u)\}_{k\in\mathbf{N},z_u=(\langle\mathcal{Z}\rangle,z\in\{0,1\}^{\leq k^d},1^{c\cdot|z|})} \\ &= \quad \{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^{\leq k^d}}. \end{aligned}$$

In particular, as long as $|z|$ is polynomial in k, we have that $|z_u|$ is also polynomial in k (albeit with a different polynomial). Consequently, $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. (Notice that if $|z_u|$ were not polynomial in k then the last derivation would not hold.) □

Remark: Claim 12 is an extension of the equivalence argument for the case of computationally unbounded environment and adversaries, discussed in [C00], in the context of computationally unbounded adversaries. A crucial element in the proof of this claim is the fact that the class of allowed environment permits existence of an environment $\mathcal{Z}_u$ that is universal with respect to all allowed environments. In the context of computationally bounded environments, this feature becomes possible when using a definition of PPT ITMs where the running time may depend not only on the security parameter, but also on the length of the input. Indeed, in [C00] and in previous versions of this work, which restrict ITMs to run in time that is bound by a fixed polynomial in the security parameter, standard security and security with respect to specialized simulators end up being different notions (see, e.g., [L03a, HU05]).

Finally we note that Claim 12 does *not* hold for the notion of uniform-complexity security, i.e. in the case where the environment takes inputs only of the form 1^n for some n.

5 Universal composition

This section states and proves the universal composition theorem. Section 5.1 defines the composition operation and states the composition theorem. Section 5.2 presents the proof. Section 5.3 discusses and motivates some aspects of the theorem, and sketches some extensions. Additional discussion on the implications of the UC operation and theorem appears in [C06].

5.1 The universal composition operation and theorem

While the main intended use of universal composition is for replacing an ideal functionality $\mathcal{F}$ with a protocol that securely realizes $\mathcal{F}$, we define universal composition more generally, in terms of replacing one subroutine protocol with another. This both simplifies the presentation and makes the result more powerful.

Universal composition. We present the composition operation in terms of an operator on protocols. This operator, called the universal composition operator UC(), is defined as follows. Given a protocol ϕ, a protocol π (that presumably makes subroutine calls to ϕ), and a protocol ρ (that presumably UC-emulates ϕ), the composed protocol $\pi^{\rho/\phi} = \text{UC}(\pi,\rho,\phi)$ is identical to protocol π, with the following modifications.

1. Wherever π contains an instruction to pass input x to an ITI running ϕ with identity (sid,pid), then $\pi^{\rho/\phi}$ contains instead an instruction to pass input x to an ITI running ρ with identity (sid,pid).

2. Whenever $\pi^{\rho/\phi}$ receives an output passed from $\rho_{(sid,pid')}$ (i.e., from an ITI running ρ with identity (sid,pid')), it proceeds as π proceeds when it receives an output passed from $\phi_{(sid,pid')}$.

When protocol ϕ is the ideal protocol $\textsc{ideal}_{\mathcal{F}}$ for some ideal functionality $\mathcal{F}$, we denote the composed protocol by $\pi^{\rho/\mathcal{F}}$. Also, when ϕ is understood from the context we use the shorthand π^{ρ} instead for $\pi^{\rho/\phi}$. See a graphical depiction in Figure 4 on page 13.

We remark that the composition operation can alternatively be defined as a model operation where the protocols remain unchanged, and the only change is that the control function invokes instances of ρ instead of instances ϕ. While technically equivalent, we find the present formulation, where the protocol determines the code run by its subroutines, more intuitively appealing.

Clearly, if protocols π, ϕ, and ρ are PPT then $\pi^{\rho/\phi}$ is PPT (with a bounding polynomial that is the maximum of the individual bounding polynomials).

Subroutine Respecting protocols. Before stating the theorem, We define the following natural property of protocols. Roughly speaking, a protocol ρ is subroutine respecting if the only input/output interface between each instance of ρ and other protocol instances is done by the main parties of ρ. The sub-parties of ρ, namely the descendants of the main parties of ρ, exchange input/output only with parties or sub-parties of this instance of ρ. More precisely, say that an instance of ρ in an execution of π^{ρ} is subroutine respecting if no sub-party of this instance passes inputs or outputs to or from an ITI which is not a party or sub-party of this instance. Furthermore, all sub-parties of this instance ignore all inputs and outputs received from parties other than the parties and sub-parties of ρ. (Technically speaking, ignoring an incoming value means immediately overwriting it with, say, zeroes.) Protocol ρ is subroutine respecting if any instance of ρ in any execution of π^{ρ}, for any π, is subroutine respecting.

Theorem statement. We are now ready to state the composition theorem. First we state a general theorem, to be followed by two corollaries. The general formulation makes the following statement: Let π, ϕ, ρ be protocols, such that protocol ρ UC-emulates protocol ϕ as in Definition 5 and both ϕ and ρ are subroutine respecting. Then the protocol $\pi^{\rho/\phi} = \textsc{uc}(\pi, \rho, \phi)$ UC-emulates protocol π. Here all protocols may be hybrid protocols, i.e. they may call the ideal protocol for some ideal functionalities, as long as these ideal functionalities are PPT. A more quantitative statement of the UC theorem is discussed in Section 5.3.

Theorem 13 (Universal composition: General statement) *Let π, ρ, ϕ be PPT protocols such that ρ UC-emulates ϕ and both ϕ and ρ are subroutine respecting. Then protocol $\pi^{\rho/\phi}$ UC-emulates protocol π.*

As a special case, we get:

Corollary 14 *Let π, ρ be PPT protocols such that ρ UC-realizes a PPT ideal functionality $\mathcal{F}$, and both ϕ and ρ are subroutine respecting. Then protocol $\pi^{\rho/\mathcal{F}}$ UC-emulates protocol π.*

Next we concentrate on protocols π that securely realize some ideal functionality $\mathcal{G}$. The following corollary essentially states that if protocol π securely realizes $\mathcal{G}$ using calls to an ideal functionality $\mathcal{F}$, $\mathcal{F}$ is PPT, and ρ securely realizes $\mathcal{F}$, then $\pi^{\rho/\mathcal{F}}$ securely realizes $\mathcal{G}$.

Corollary 15 (Universal composition: Realizing functionalities) *Let $\mathcal{F}, \mathcal{G}$ be ideal functionalities such that $\mathcal{F}$ is PPT. Let π be a subroutine respecting protocol that UC-realizes $\mathcal{G}$, and let ρ be a subroutine respecting protocol that securely realizes $\mathcal{F}$. Then the composed protocol $\pi^{\rho/\mathcal{F}}$ securely realizes $\mathcal{G}$.*

Proof: Let $\mathcal{A}$ be an adversary that interacts with parties running $\pi^{\rho/\mathcal{F}}$. Theorem 13 guarantees that there exists an adversary $\mathcal{A}_{\mathcal{F}}$ such that $\text{EXEC}_{\pi,\mathcal{A}_{\mathcal{F}},\mathcal{Z}} \approx \text{EXEC}_{\pi^{\rho/\mathcal{F}},\mathcal{A},\mathcal{Z}}$ for any environment $\mathcal{Z}$. Since π UC-realizes $\mathcal{G}$, there exists a simulator $\mathcal{S}$ such that $\text{IDEAL}_{\mathcal{G},\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A}_{\mathcal{F}},\mathcal{Z}}$ for any $\mathcal{Z}$. Using the transitivity of indistinguishability of ensembles we obtain that $\text{IDEAL}_{\mathcal{G},\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi^{\rho/\mathcal{F}},\mathcal{A},\mathcal{Z}}$ for any environment $\mathcal{Z}$. □

5.2 Proof of the composition theorem

A high-level sketch of the proof was presented in section 2. Section 5.2.1 contains an outline of the proof. A detailed proof appears in Section 5.2.2.

5.2.1 Proof outline

The proof uses the equivalent formulation of emulation with respect to dummy adversaries (see Claim 10). This formulation considerably simplifies the presentation of the proof. Let π, ϕ and ρ be PPT protocols such that ρ UC-emulates ϕ, and let $\pi^\rho = \pi^{\rho/\phi} = \text{UC}(\pi, \rho, \phi)$ be the composed protocol. We wish to construct an adversary $\mathcal{A}_\pi$ so that no $\mathcal{Z}$ will be able to tell whether it is interacting with π^ρ and the dummy adversary or with π and $\mathcal{A}_\pi$. That is, for any $\mathcal{Z}$, $\mathcal{A}_\pi$ should satisfy

$$\text{EXEC}_{\pi^\rho,\mathcal{D},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}. \tag{2}$$

The general outline of the proof proceeds as follows. The fact that ρ emulates ϕ guarantees that there exists an adversary (called a simulator) $\mathcal{S}$, such that for any environment $\mathcal{Z}_\rho$ we have:

$$\text{EXEC}_{\rho,\mathcal{D},\mathcal{Z}_\rho} \approx \text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}_\rho}. \tag{3}$$

Adversary $\mathcal{A}_\pi$ is constructed out of $\mathcal{S}$. We then demonstrate that $\mathcal{A}_\pi$ satisfies (2). This is done by reduction: Given an environment $\mathcal{Z}$ that violates (2), we construct an environment $\mathcal{Z}_\rho$ that violates (3).

Adversary $\mathcal{A}_\pi$ operates as follows. Recall that $\mathcal{Z}$ expects to interact with parties running π^ρ. The idea is to separate the interaction between $\mathcal{Z}$ and the parties into two parts. To mimic the sending and receiving of messages from the parties of each instance of ρ (and their subroutines), $\mathcal{A}_\pi$ runs an instance of the simulator $\mathcal{S}$. To mimic the sending and receiving of messages from the parties running π, $\mathcal{A}_\pi$ interacts directly with the actual, external parties running π. A bit more specifically, recall that $\mathcal{Z}$ receives the messages sent by the parties of π, by the parties of all instances of ρ, and by all their subsidiaries. In addition, $\mathcal{Z}$ delivers messages to all these entities. (These messages are relayed via the dummy adversary. They include also the corruption instructions.) $\mathcal{A}_\pi$ runs an instance of the simulator $\mathcal{S}$ for each instance of ϕ invoked by a party running π in the system it interacts with. When activated with a message sent by a party or sub-party of π, $\mathcal{A}_\pi$ passes this message to $\mathcal{Z}$, together with the identity of the sender. When activated with message m sent by $\mathcal{Z}$ to a party with identity id that runs ρ, then $\mathcal{A}_\pi$ forwards (m, id) to the corresponding instance of $\mathcal{S}$. If $\mathcal{Z}$ sends a message to be delivered to another party (that runs either π or a subroutine of π), then $\mathcal{A}_\pi$ delivers m to the actual party, $P_{(id)}$. Any message from an instance of $\mathcal{S}$ is passed as output to $\mathcal{Z}$. Figure 7 presents a graphical depiction of the operation of $\mathcal{A}_\pi$.

The validity of $\mathcal{A}_\pi$ is demonstrated, based on the validity of $\mathcal{S}$, via a hybrids argument. While the basic logic of the argument is standard, applying the argument to our setting requires some care. We sketch this argument. (The actual argument is slightly more complex; still, this sketch captures the essence of the argument.) Let m be an upper bound on the number of instances of ρ

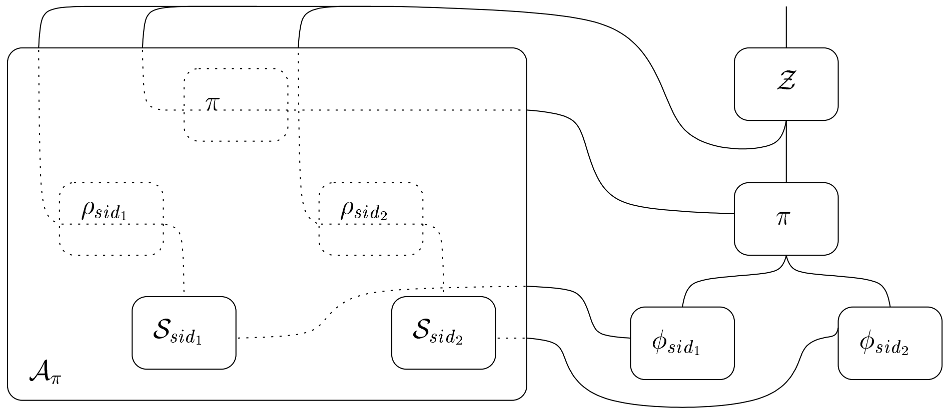

Figure 7: The operation of $\mathcal{A}_\pi$. Inputs from $\mathcal{Z}$ that represent messages of the instance of π are forwarded to the actual instance of π. Inputs directed to an instance of ρ are directed to the corresponding instance of $\mathcal{S}$. Messages from an instance of $\mathcal{S}$ are directed to the corresponding actual instance of ϕ. For graphical clarity we use a single box to represent a multi-party protocol instance.

that are invoked in this interaction. Informally, for $l \le m$ we let π_l denote the protocol where the interaction with the first l instances of ϕ remains unchanged, whereas the rest of the instances of ϕ are replaced with instances of ρ. In particular, protocol π_m is essentially identical to protocol π. Similarly, protocol π_0 is essentially identical to protocol π^ρ.[8]

Now, assume that there exists an environment $\mathcal{Z}$ that distinguishes with probability ϵ between an interaction with $\mathcal{A}_\pi$ and π, and an interaction with $\mathcal{D}$ and $\pi^{\rho/\phi}$. Then there is an $0 < l \le m$ such that $\mathcal{Z}$ distinguishes between an interaction with $\mathcal{A}_\pi$ and π_l, and an interaction with $\mathcal{A}_\pi$ and π_{l-1}. We then construct an environment $\mathcal{Z}_\rho$ that can distinguish with probability ϵ/m between an interaction with $\mathcal{D}$ and parties running a single instance of ρ, and an interaction with $\mathcal{S}$ and ϕ. Essentially, $\mathcal{Z}_\rho$ runs a simulated execution of $\mathcal{Z}$, adversary $\mathcal{A}_\pi$, and parties running π_l, but with the following exception. $\mathcal{Z}_\rho$ uses its actual interaction (which is either with ϕ or with ρ) to replace the parts of the simulated execution that have to do with the interaction with the lth instance of ϕ, denoted ϕ_l. A bit more specifically, whenever some simulated party running π passes an input x to ϕ_l, $\mathcal{Z}_\rho$ passes input x to the corresponding actual party. Outputs generated by an actual party running ρ are treated like outputs from ϕ_l to the corresponding simulated party running π. (Since ρ and ϕ are subroutine respecting, we are guaranteed that the only inputs and outputs between the external protocol instance and $\mathcal{Z}_\rho$ are done via the inputs and outputs of the parties themselves.) Furthermore, whenever the simulated adversary $\mathcal{A}_\pi$ passes input value v to the instance of $\mathcal{S}$ that corresponds to ϕ_l, $\mathcal{Z}_\rho$ passes input v to the actual adversary it interacts with. Any output obtained from the actual adversary is passed to the simulated $\mathcal{A}_\pi$ as an output from the corresponding instance of $\mathcal{S}$. Once the simulated $\mathcal{Z}$ halts, $\mathcal{Z}_\rho$ halts and outputs whatever $\mathcal{Z}$ outputs. Figure 8 presents a graphical depiction of the operation of $\mathcal{Z}_\rho$.

The proof is completed by observing that, if $\mathcal{Z}_\rho$ interacts with $\mathcal{S}$ and ϕ, then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ has the same distribution as the view of $\mathcal{Z}$ when interacting with $\mathcal{A}_\pi$ and π_l. Similarly, if $\mathcal{Z}_\rho$ interacts with $\mathcal{D}$ and parties running ρ, then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ has the same distribution as the view of $\mathcal{Z}$ when interacting with $\mathcal{A}_\pi$ and π_{l-1}.

[8]In the actual proof we consider a different model of computation for each hybrid, rather than considering a different protocol. The reason is that the parties running the protocol may not know which is the (globally) lth instance to be invoked. See details within.

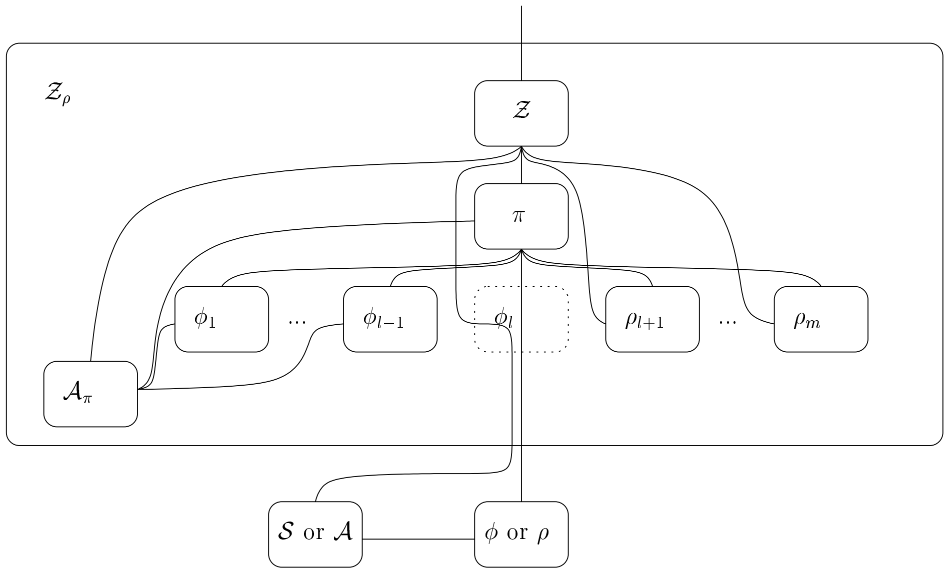

Figure 8: The operation of $\mathcal{Z}_\rho$. An interaction of $\mathcal{Z}$ with π is simulated, so that the first $l-1$ instances of ϕ remain unchanged, the lth instance is mapped to the external execution, and the remaining instances of ϕ are replaced by instances of ρ. For graphical clarity we use a single box to represent a multi-party protocol instance.

5.2.2 A detailed proof

We proceed with a detailed proof of Theorem 13, substantiating the above outline. Most of the terminology and notations are defined in Section 3.

Construction of $\mathcal{A}_\pi$. Let π, ϕ, ρ be protocols, where ρ emulates ϕ, and and let $\pi^\rho = \pi^{\rho/\phi}$ be the composed protocol. The fact that ρ UC-emulates ϕ guarantees that there exists an ideal-process adversary $\mathcal{S}$ such that $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}_\rho} \approx \text{EXEC}_{\rho,\mathcal{Z}_\rho}$ holds for any environment $\mathcal{Z}_\rho$.

Adversary $\mathcal{A}_\pi$ uses $\mathcal{S}$ and is presented in Figure 9. Observe that the handling of corruptions of ITIs (parties and sub-parties of π and ρ) is implicit in the description of $\mathcal{A}_\pi$. Indeed, recall that the corruption requests and the returned information are modeled as special cases of messages and outputs.

Validity of $\mathcal{A}_\pi$. First, note that $\mathcal{A}_\pi$ is PPT. In fact, the polynomial $p(\cdot)$ bounding the running time of $\mathcal{A}_\pi$ can be set to be the polynomial bounding the running time of $\mathcal{S}$, plus a linear polynomial. (Note that $p(\cdot)$ does not depend on the number of instances of ρ. The linear polynomial is needed to take care of relaying the messages of protocol π.)

Next, assume that there exists an environment machine $\mathcal{Z}$ that violates the validity of $\mathcal{A}_\pi$ (that is, $\mathcal{Z}$ violates Equation (2)). We construct an environment machine $\mathcal{Z}_\rho$ that violates the validity of $\mathcal{S}$ with respect to a single run of ρ. (That is, $\mathcal{Z}_\rho$ violates Equation (3).) More specifically, fix some input value z and a value k of the security parameter, and assume that

$$\text{EXEC}_{\pi^\rho,\mathcal{Z}}(k,z) - \text{EXEC}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(k,z) \geq e. \tag{4}$$

We show that

$$\text{EXEC}_{\rho,\mathcal{D},\mathcal{Z}_\rho}(k,z) - \text{IDEAL}_{\phi,\mathcal{S},\mathcal{Z}_\rho}(k,z) \geq e/t \tag{5}$$

Adversary $\mathcal{A}_\pi$

Adversary $\mathcal{A}_\pi$ proceeds as follows, interacting with parties running protocol π and environment $\mathcal{Z}$.

1. When activated with input (m, id, c) (coming from $\mathcal{Z}$), where m is a message, $id = (sid, pid)$ is an identity, and c is a code for an ITM, do:
   (a) If the specified recipient is a party of an instance sid' of ρ, or a subsidiary thereof, then first locate the internally running instance $\mathcal{S}_{(sid',\top)}$ of $\mathcal{S}$ that handles the protocol instance with SID= sid. (That is, either $sid' = sid$ or sid is the SID of a subsidiary of a party with SID=sid'.) If no such instance of $\mathcal{S}$ is found, then internally invoke a new instance of $\mathcal{S}$ with identity $(sid, \top)$. Next, activate this instance of $\mathcal{S}$ with input (m, id, c) and follow its instructions.
   (b) Else (i.e., the specified recipient is not a party or sub-party of an instance of ρ), deliver the message m to the recipient. Using the terminology of Definition 1, this means that $\mathcal{A}_\pi$ executes an external-write request to the incoming message tape of an ITI (c, id).
2. When activated with an incoming message m from some party, do:
   (a) If m was sent by some party $\phi_{(sid,pid)}$ of protocol ϕ, or a subsidiary thereof, then internally activate the instance $\mathcal{S}_{(sid,\top)}$ of $\mathcal{S}$ with incoming message m from $\phi_{(sid,pid)}$, and follow its instructions. (If no such instance of $\mathcal{S}$ exists then invoke it, internally, and label it $\mathcal{S}_{(sid,\top)}$.)
   (b) If m was sent by another party or sub-party of π, then pass output (m, id) to $\mathcal{Z}$.
3. When an instance of $\mathcal{S}$ internally generates a request to deliver a message m to some party, then deliver m to this party. When $\mathcal{S}_{(id)}$ requests to pass an output v to its environment then output v to $\mathcal{Z}$, but with the the exception that $\mathcal{A}_\pi$ mimics the time bounds of a dummy adversary. That is, $\mathcal{A}_\pi$ stops delivering output to $\mathcal{Z}$ as soon as the output length exceeds the overall input length of $\mathcal{A}_\pi$.

Figure 9: The adversary for protocol π.

where $t = t(k, |z|)$ is a polynomial function.

In preparation to constructing $\mathcal{Z}_\rho$, we define the following distributions, and make some observations on $\mathcal{A}_\pi$. Consider an execution of protocol π with adversary $\mathcal{A}_\pi$ and environment $\mathcal{Z}$. Let $t = t(k, |z|)$ be an upper bound on the number of instances of ϕ within π in this execution. (The bound t is used in the analysis only. The parties need not be aware of t. Also, t is polynomial in $k, |z|$ since $\mathcal{Z}$ is PPT.) For $0 \le l \le t$, Let the l-hybrid model for running protocol π denote the extended system of ITMs that is identical to the basic model of computation, with the exception that the control function is modified as follows. (Recall that the control function of an extended system of ITMs determines, among other things, the target ITIs of external-write requests.) The external-write requests to tapes of the first l instances of ϕ to be invoked are treated as usual. The external-write requests to the tapes of all other instances of ϕ are directed to the corresponding instances of parties running ρ. That is, let sid_i denote the SID of the ith instance of ϕ to be invoked in an execution. Then, given an external-write request made by an some ITI to $\phi_{(sid_i,pid)}$ (i.e., to the party running ϕ with identity (sid_i, pid)) for some pid, where $i > l$, the control function writes the requested value to the requested tape of $\rho_{(sid_i,pid)}$; if no such ITI exists then one is invoked. (Note that these modifications apply to external-write requests by *any* ITI, including ITIs that participate in instances of ϕ and ρ, as well as subroutines thereof. We let $\textsc{exec}^l_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ denote the output of this system of ITMs on input z and security parameter k for the environment $\mathcal{Z}$.

Environment $\mathcal{Z}_\rho$

Environment $\mathcal{Z}_\rho$ proceeds as follows, given a value k for the security parameter, input z_ρ, and expecting to interact with parties running a single instance of ρ. We first present a procedure called Simulate(). Next we describe the main program of $\mathcal{Z}_\rho$.

Procedure Simulate(s, l)

1. Expect the parameter s to contain a global state of a system of ITMs representing an execution of protocol π in the l-hybrid model, with adversary $\hat{\mathcal{A}}_\pi$ and environment $\mathcal{Z}$. Continue a simulated execution from state s (making the necessary random choices along the way), until one of the following events occurs. Let ϕ_l denote the lth instance of ϕ that is invoked in the simulated execution, and let sid_l denote the SID of ϕ_l.
   (a) Some simulated party passes input x to a party with identity (sid_l, pid). (That is, the recipient party participates in the lth instance of ϕ.) In this case, save the current state of the simulated system in s, pass input x the actual party with identity (sid_l, pid), and complete this activation.
   (b) The simulated $\mathcal{Z}$ passes input (m, id, c, a) to the simulated adversary $\hat{\mathcal{A}}_\pi$, where $id = (sid_l, pid)$ for some pid. In this case, save the current state of the simulated system in s, deliver the message (m, id, c) to the party running code c with identity id, and complete this activation.
   (c) The simulated environment $\mathcal{Z}$ halts. In this case, $\mathcal{Z}_\rho$ outputs whatever $\mathcal{Z}$ outputs and halts.

Main program for $\mathcal{Z}_\rho$:

1. When activated for the first time, interpret the input z_ρ as a pair $z_\rho = (z, l)$ where z is an input for $\mathcal{Z}$, and $l \in \mathbf{N}$. Initialize a variable s to hold the initial global state of a system of ITMs representing an execution of protocol π in the l-hybrid model, with adversary $\hat{\mathcal{A}}_\pi$ and environment $\mathcal{Z}$ on input z and security parameter k. Next, run Simulate(s, l).
2. In any other activation, let x be the new value written on the subroutine-output tape. Next:
   (a) Update the state s. That is:
      i. If the new value, x, was written by some party $P_{(id)}$ with identity $id = (sid_l, pid)$ then write (x, id) to the subroutine-output tape of the simulated party that invoked $P_{(id_l)}$. ($P_{(id_l)}$ is either a party running ϕ or a party running ρ.)
      ii. If the new value, x, was written by the adversary, then update the state of the simulated adversary $\hat{\mathcal{A}}_\pi$ to include an output x generated by $\mathcal{S}_{(sid_l)}$ (i.e., by the instance of the simulator $\mathcal{S}$ with SID$= sid_l$).
   (b) Simulate an execution of the system from state s. That is, run Simulate(s, l).

Figure 10: The environment for a single instance of ρ.

We also define the following variants of adversary $\mathcal{A}_\pi$ and environment $\mathcal{Z}$. Let $\hat{\mathcal{A}}_\pi$ denote the adversary that is identical to $\mathcal{A}_\pi$, with the following exception. $\hat{\mathcal{A}}_\pi$ expects to have its input include an additional flag, s, that determines whether to consider invoking an instance of $\mathcal{S}$ in this activation. That is, if $a = 1$ then $\hat{\mathcal{A}}_\pi$ operates as $\mathcal{A}_\pi$. If $a = 0$ then $\hat{\mathcal{A}}_\pi$ skips the condition in Step 1a in Figure 9; instead it always runs Step 1b. Let $\mathcal{Z}^{(l)}$ denote the environment machine that is identical to $\mathcal{Z}$, with the following exceptions. Whenever $\mathcal{Z}$ generates a message (m, id, ρ)

to be delivered to the party running ρ with identity id, then $\mathcal{Z}^{(l)}$ passes input (m, id, c, a) to the adversary, where a is set as follows. Let $id = (sid, pid)$. If sid is the SID of one of the first l instances of ρ, then $\mathcal{Z}^{(l)}$ sets $a = 1$. Otherwise, $a = 0$.

We observe that, when $\hat{\mathcal{A}}_\pi$ interacts with $\mathcal{Z}^{(l)}$ and parties running π in the l-hybrid model, then it internally runs at most l instances of the simulator $\mathcal{S}$. (These are the instances that correspond to the first l instances of protocol ϕ with which $\hat{\mathcal{A}}_\pi$ interacts.) The remaining instances of $\mathcal{S}$ are replaced by interacting with the actual parties or sub-parties of the corresponding instances of ρ. Consequently, we have that the output of $\mathcal{Z}^{(t)}$ from an interaction with π and $\hat{\mathcal{A}}_\pi$ in the t-hybrid model is distributed identically to the output of $\mathcal{Z}$ from an interaction with π and $\mathcal{A}_\pi$ in the basic model, i.e. $\text{EXEC}^t_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(t)}} = \text{EXEC}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}$. Similarly, the output of $\mathcal{Z}^{(0)}$ from an interaction with π and $\hat{\mathcal{A}}_\pi$ in the 0-hybrid model is distributed identically to the output of $\mathcal{Z}$ from an interaction with π^ρ in the basic model of computation, i.e. $\text{EXEC}^0_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(0)}} = \text{EXEC}_{\pi,\mathcal{D},\mathcal{Z}}$. Consequently, Inequality (4) can be rewritten as:

$$\text{EXEC}^0_{\pi,\hat{\mathcal{A}}_\rho,\mathcal{Z}^{(0)}}(k,z) - \text{EXEC}^t_{\pi,\hat{\mathcal{A}}_\rho,\mathcal{Z}^{(t)}}(k,z) \geq e. \tag{6}$$

We turn to constructing and analyzing environment $\mathcal{Z}_\rho$. The construction of $\mathcal{Z}_\rho$ is presented in Figure 10. We first note that $\mathcal{Z}_\rho$ is PPT. This follows from the fact that the entire execution of the system is completed in polynomial number of steps. (Indeed, the polynomial bounding the runtime of $\mathcal{Z}_\rho$ can be bounded by the maximum among the polynomials bounding the running times of $\mathcal{Z}$, π^ρ, and π^ϕ.)

The rest of the proof analyzes the validity of $\mathcal{Z}_\rho$, demonstrating (5). It follows from (6) that there exists a value $l \in \{1, ..., m\}$ such that

$$|\text{EXEC}^l_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(l)}}(k,z) - \text{EXEC}^{l-1}_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(l)}}(k,z)| \geq \epsilon/t. \tag{7}$$

However, for every $l \in \{1, .., m\}$ we have

$$\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}_\rho}(k,(z,l)) = \text{EXEC}^l_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(l-1)}}(k,z) \tag{8}$$

and

$$\text{EXEC}_{\pi^\rho,\mathcal{D},\mathcal{Z}_\rho}(k,(z,l)) = \text{EXEC}^{l-1}_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(l-1)}}(k,z). \tag{9}$$

Equations (8) and (9) follow from inspecting the code of $\mathcal{Z}_\rho$ and $\mathcal{A}_\pi$. In particular, if $\mathcal{Z}_\rho$ interacts with parties running ϕ then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ is distributed identically to the view of $\mathcal{Z}$, run within $\mathcal{Z}^{(l)}$, when interacting with π and $\hat{\mathcal{A}}_\pi$ in the l-hybrid model. Similarly, if $\mathcal{Z}_\rho$ interacts with parties running ρ then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ is distributed identically to the view of $\mathcal{Z}$ run within $\mathcal{Z}^{(l)}$, when interacting with π and $\hat{\mathcal{A}}_\pi$ in the $(l-1)$-hybrid model. (Here it is important to note that, since both ρ and ϕ are subroutine respecting, the only communication between the external instance of ρ or ϕ, and the ITIs outside this instance, is the inputs and outputs of the main parties of this instance.)

From Equations (7), (8) and (9) it follows that:

$$|\text{EXEC}_{\rho,\mathcal{D},\mathcal{Z}_\rho}(k,(z,l)) - \text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}_\rho}(k,(z,l))| \geq \epsilon/t \tag{10}$$

as desired.

5.3 Discussion and extensions

Some aspects of the universal composition theorem were discussed in Section 2.3. This section highlights additional aspects, and presents some extensions of the theorem.

On composability with respect to closed environments. Recall that the closed-environment variant of the definition of emulation (Definition 5) considers only environments that take external input that contains no information other than its length, e.g. inputs of the form 1^n for some n. We note that the UC theorem still holds even for this variant: the only difference in the proofs is that, instead of receiving the index l of the hybrid execution as part of the input, the distinguishing environment $\mathcal{Z}_\rho$ chooses l uniformly at random in $1...t$, where t is an upper bound on the number of sessions of ϕ invoked by π in an execution with environment $\mathcal{Z}$.

Composing multiple different protocols. The composition theorem (Theorem 13) is stated only for the case of replacing instances of a *single* protocol ϕ with instances of another protocol. The theorem holds also for the case where multiple different protocols $\phi_1, \phi_2, ...$ are replaced by protocols $\rho_1, \rho_2, ...$, respectively. (This can be seen either by directly extending the current proof, or by defining a single "universal" protocol that mimics multiple different ones.)

A quantitative statement of the UC theorem. We observe that the polynomial bounding the running time of the constructed environment $\mathcal{Z}_\rho$ is no larger than the polynomial bounding the running time of the given environment $\mathcal{Z}$. From this, along with the fact that the distinguishing probability of $\mathcal{Z}_\rho$ decreases proportionally to the number of instances of ρ, we have that if ρ (k, ϵ, g)-UC-emulates ϕ for some value k of the security parameter, then protocol $\pi^{\rho/\phi}$ (k, ϵ', g)-UC-emulates protocol π, where $\epsilon'(p_\mathcal{A}, P_\mathcal{Z})() = t \cdot \epsilon(c \cdot p_\mathcal{A}, c \cdot p_\mathcal{Z})()$, t is a bound on the number of instances of ρ in π^ρ, and c is some small constant. That is, the emulation slack increases by a factor of roughly t, and the simulation overhead remains the same.

An alternative proof of the UC theorem. The above proof of Theorem 13 constructs, in a single step, a simulator $\mathcal{A}_\pi$ that handles all the instances of ρ. An alternative proof proceeds as follows:

1. Prove Theorem 13 for the case where protocol π calls only a single instance of ϕ, or equivalently when only one instance of ϕ, out of potentially many instances called by π, is replaced with an instance of ρ. Call this theorem the "single-instance UC theorem". Proving this theorem is considerably simpler than the current proof; in particular, the hybrids argument is not necessary. Furthermore, in this case the UC theorem preserves both the emulation slack ϵ and the simulation overhead g.

2. To handle replacement of polynomially many concurrent instances of ϕ by instances of ρ, simply apply the "single-instance UC theorem" iteratively, where in each iteration a new instance of ϕ is replaced by an instance of ρ and the rest the instances of ρ, ϕ and π are treated as part of the environment. Then, use the transitivity of UC-emulation to deduce that π^ρ UC-emulates π.

When comparing this proof to the direct proof in Section 5.2, one should take into account the degradation in the quality of the emulation, namely the growth of the emulation slack ϵ and the simulation overhead g. Recall that, by transitivity, the emulation slack increases additively under iterative applications of UC-emulation (see discussion on page 36). This means that the emulation slack grows linearly in the number of instances of the composed protocol; this is the same as in the proof of Section 5.2.

However, the proof of Section 5.2 does better in terms of the simulation overhead: In that proof, there is no increase in the simulation overhead. In contrast, in the alternative proof mentioned here the simulation overhead when composing t instances is the t-wise composition of the overheads incurred by the t applications of the UC theorem. Therefore, the simulation overhead increases proportionally to the number of composed instances. Still, the increase is moderate: By Claim 8, we have that each such overhead is an additive polynomial $p()$ that depends only on protocols ρ and ϕ. This means that the simulation overhead increases by an additive factor of $tp()$.

Nesting of protocol instances. The universal composition operation can be applied repeatedly to perform "nested" replacements of calls to sub-protocols with calls to other sub-protocols. We observe that repeated application of the composition operation preserves security. For instance, if a protocol ρ_1 UC-emulates protocol ϕ_1, and protocol ρ_2 UC-emulates protocol ϕ_2 using calls to ϕ_1, then for any protocol π that uses calls to ϕ_2 it holds that the composed protocol $\pi^{(\rho_2^{\rho_1/\phi_1})/\phi_2} = \textsc{uc}(\pi, \textsc{uc}(\rho_2, \rho_1, \phi_1), \phi_2)$ UC-emulates π.

Security for nested applications of universal composition can be argued by repeated application of the UC theorem, similarly to the alternative proof of the UC theorem, sketched above. As there, the emulation slack and the simulation overhead increase linearly in the number of composed instances, regardless of whether these instances are nested or not. In addition, it is possible to extend the direct proof of Section 5.2 to deal with the case of nested applications of the UC operation with arbitrary polynomial depth, and with no increase in the simulation overhead. We omit further details. The fact that the UC theorem extends to arbitrary polynomial nesting of the UC operation was independently made in [BM04] for their variant of the UC framework.

Beyond PPT. The UC theorem is stated and proven for PPT systems of ITMs, namely for the case where all the involved entities are PPT. It is readily seen that the theorem holds also for other classes of ITMs and systems, as long as the definition of the class guarantees that any execution of any system of ITMs can be "simulated" on a single ITM from the same class.

More precisely, say that a class $\mathcal{C}$ of ITMs is self-simulatable if, for any system (I, C) of ITMs where both I and C (in its ITM representation) are in $\mathcal{C}$, there exists an ITM M in $\mathcal{C}$ such that, on any input and any random input, the output of a single instance of M equals the output of (I, C). (Stated in these terms, Proposition 3 on page 28 asserts that for any concave function $T()$, the class of ITMs that run in time $T()$ is self-simulatable.)

Say that protocol π UC-emulates protocol ϕ with respect to class $\mathcal{C}$ if Definition 5 holds when the class of PPT ITMs is replaced with class $\mathcal{C}$, namely when π, $\mathcal{A}$, $\mathcal{S}$, and $\mathcal{Z}$ are taken to be ITMs in $\mathcal{C}$. Then we have:

Proposition 16 *Let $\mathcal{C}$ be a self-simulatable class of ITMs, and let π, ρ, ϕ be protocols in $\mathcal{C}$ such that ρ UC-emulates ϕ with respect to class $\mathcal{C}$. Then protocol $\pi^{\rho/\phi}$ UC-emulates protocol π with respect to class $\mathcal{C}$.*

Furthermore, if ρ statistically (resp., perfectly) UC-emulates ϕ with respect to class $\mathcal{C}$ of adversaries then $\pi^{\rho/\phi}$ statistically (resp., perfectly) UC-emulates protocol π with respect to class $\mathcal{C}$.

It is stressed, however, that the UC theorem is, in general, *false* in settings where systems of ITMs cannot be simulated on a single ITM from the same class. We exemplify this point for the case where all entities in the system are bound to be PPT, except for the protocol ϕ which is not PPT.[9] More specifically, we present an ideal functionality $\mathcal{F}$ that is not PPT, and a PPT protocol ρ that UC-realizes $\mathcal{F}$ with respect to PPT environments. Then we present an $\mathcal{F}$-hybrid protocol π,

[9] We thank Manoj Prabhakaran and Amit Sahai for this example.

that uses only *two* instances of the ideal protocol for $\mathcal{F}$, and such that $\pi^{\rho/\mathcal{F}}$ does not UC-emulate π. In fact, for *any* PPT ρ' we have that $\pi^{\rho'/\mathcal{F}}$ does not emulate π.

In order to define $\mathcal{F}$, we first recall the definition of pseudorandom ensembles of evasive sets, defined in [GK89] for a related purpose. An ensemble $\mathcal{S} = \{S_k\}_{k\in\mathbf{N}}$ where each $S_k = \{s_{k,i}\}_{i\in\{0,1\}^k}$ and each $s_{k,i} \subset \{0,1\}^k$ is a pseudorandom evasive set ensemble if: (a) $\mathcal{S}$ is pseudorandom, that is for all large enough $k \in \mathbf{N}$ and for all $i \in \{0,1\}^k$ we have that a random element $x \stackrel{\text{R}}{\leftarrow} s_{k,i}$ is computationally indistinguishable from $x \stackrel{\text{R}}{\leftarrow} \{0,1\}^k$. (b) $\mathcal{S}$ is evasive, that is for any non-uniform PPT algorithm A and for any $z \in \{0,1\}^*$, we have that $\text{Prob}_{i\stackrel{\text{R}}{\leftarrow}\{0,1\}^k}[A(z,i) \in s_{k,i}]$ is negligible in k, where $k = |z|$. It is shown in [GK89], via a counting argument, that pseudorandom evasive set ensembles exist.

Now, define $\mathcal{F}$ as follows. $\mathcal{F}$ uses the ensemble $\mathcal{S}$ and interacts with one party only. Given security parameter k, it first chooses $i \stackrel{\text{R}}{\leftarrow} \{0,1\}^k$ and outputs i. Then, given an input $(x, i') \in \{0,1\}^k \times [2^k]$, it first checks whether $x \in s_{k,i}$. If so, then it outputs `success`. Otherwise it outputs $r \stackrel{\text{R}}{\leftarrow} s_{k,i'}$.

Protocol ρ for realizing $\mathcal{F}$ is simple: Given security parameter k it outputs $i \stackrel{\text{R}}{\leftarrow} \{0,1\}^k$. Given an input $x \in \{0,1\}^k$, it outputs $r \stackrel{\text{R}}{\leftarrow} \{0,1\}^k$. It is easy to see that ρ UC-realizes $\mathcal{F}$: Since $\mathcal{S}$ is evasive, then the probability that the input x is in the set $s_{k,i}$ is negligible, thus $\mathcal{F}$ outputs `success` only with negligible probability. Furthermore, $\mathcal{F}$ outputs a pseudorandom k-bit value, which is indistinguishable from the output of ρ.

Now, consider the following $\mathcal{F}$-hybrid protocol π. π runs two instances of $\mathcal{F}$, denoted $\mathcal{F}_1$ and $\mathcal{F}_2$. Upon invocation with security parameter k, it activates $\mathcal{F}_1$ and $\mathcal{F}_2$ with k, and obtains the indices i_1 and i_2. Next, it chooses $x_1 \stackrel{\text{R}}{\leftarrow} \{0,1\}^k$, and feeds (x_1, i_2) to $\mathcal{F}_1$. If $\mathcal{F}_1$ outputs `success` then π outputs `success` and halts. Otherwise, π feeds the value x_2 obtained from $\mathcal{F}_1$ to $\mathcal{F}_2$. If $\mathcal{F}_2$ outputs `success` then π outputs `success`; otherwise it outputs `fail`. It is easy to see that π always outputs success. However, $\pi^{\rho/\mathcal{F}}$ never outputs success. Furthermore, for *any* PPT protocol ρ' that UC-realizes $\mathcal{F}$, we have that $\pi^{\rho'/\mathcal{F}}$ outputs `success` only with negligible probability.

6 UC formulations of some computational models

Recall that the basic model of computation, captured in Figure 5 on page 32, provides only a very rudimentary way for communication between parties. Specifically, the communication is completely controlled by the adversary, in terms of both the contents and the timing of delivered messages. Also, no specific provisions for party corruption are given. In fact, this model is not intended to be a viable model of computation at all. Rather, it is intended to provide a formal basis on top of which more realistic — and more abstract — models of computation can be formulated.

As sketched and motivated in the Introduction (Section 1.3), communication models can be captured within the present framework by way of formulating appropriate ideal functionalities. That is, we say that an ideal functionality $\mathcal{F}$ represents some communication model if protocols in this communication model can be represented as $\mathcal{F}$-hybrid protocols, namely protocols that make use of (potentially multiple instances of) functionality $\mathcal{F}$. By the same approach, a corruption model is captured via a given response pattern to a corruption message from the adversary. (Formally, this response pattern can also be represented as a subroutine, or ideal functionality.)

This section presents a handful of ideal functionalities that are aimed at capturing some salient communication models. Specifically, we address authenticated and secure communication, synchronous communication, and non-concurrent protocol execution. We also formulate within the framework some standard models for party corruption.

In addition to presenting the specific ideal functionalities considered, this section intends to exemplify the use of the framework and to provide the reader with basic techniques for writing ideal functionalities. For this purpose, we also set some general writing conventions for ideal functionalities, and demonstrate their use.

We start with presenting some basic corruption models, in Section 6.1. We then turn to writing ideal functionalities: Section 6.2 presents general writing conventions. Sections 6.3 ad 6.4 present the ideal functionalities for capturing authenticated and secure communication, respectively. Section 6.5 present the ideal functionality capturing synchronous communication, and Section 6.6 presents the ideal functionality capturing non-concurrent protocol execution.

6.1 Some corruption models

The basic model of protocol execution does not specify the behavior of parties upon corruption. Indeed, the composition operation and theorem apply to any type of behavior upon corruption. Specifically, recall that corruptions of party P is captured as a reserved `(corrupt)` message that is delivered to the party by the adversary. (This operation is possible only once the environment has explicitly "agreed" to corrupting P.) Different corruption models are captured via different protocol instructions for the case that a `(Corrupt, ...)` message is received. This modeling provides greater flexibility in defining variants of standard corruption models; in addition, different corruption methods (say, passive, Byzantine, various levels of adaptivity, Fail-Stop) can co-exist in the same system.

The modeling of Byzantine corruption behavior was given in Section 4.1. This section describes how other standard corruption models are captured within the present framework. We address non-adaptive, passive (semi-honest), and proactive corruptions. While these corruption models provide very different security guarantees, from a definitional point of view they are treated in similar ways, and are thus discussed together.

Non-adaptive (static) party corruptions. Definition 9 postulates adaptive party corruptions, namely corruptions that occur as the computation proceeds, based on the information gathered by the adversary so far. Arguably, adaptive corruption of parties is a realistic threat in existing networks. Nonetheless, it is sometimes useful to consider also a weaker threat model, where the identities of the adversarially controlled parties are fixed before the computation starts; this is the case of non-adaptive (or, static) adversaries. See more discussion on the differences between the two models in [C00, CDDIM04].

In the present framework, non-adaptive corruptions can be captured by specifying that any corruption message that is received in any activation other than the first activation of the party is ignored.

Passive (honest-but-curious) party corruptions. Definition 9 postulates active party corruptions, namely corruptions where the adversary obtains total control over the behavior of corrupted parties. Another standard corruption model assumes that even corrupted parties continue to follow their prescribed protocol. Here the only advantage the adversary gains from corrupting parties is in learning the internal states of those parties. We call such adversaries passive.

Passive corruptions can be captured by changing the reaction of a party to a `corrupt` message from the adversary: While a corrupted party still reports to the adversary all of its internal state, it no longer follows the instructions of $\mathcal{A}$.

We remark that one can consider two variants of passive corruptions, depending on whether the adversary is allowed to *modify* the inputs of the corrupted parties prior to the beginning of the computation. The two variants can be captured via appropriate sets of instructions for parties upon corruption.

Transient (mobile) corruptions and proactive security. All the corruption methods so far represent "permanent" party corruptions, in the sense that once a party gets corrupted it remains corrupted throughout the computation. Another variant allows parties to "recover" from a corruption and regain their security. Such corruptions are often called mobile (or, transient). Security against such corruptions is often called proactive security. Transient corruptions can be captured by adding a `(recover` P`)` message from the adversary to P. (As with corruption messages, a recover message can be delivered only after being explicitly "approved" in a message from the environment to the adversary.) Upon receiving a `(recover` P`)` message, P stops reporting its state to the adversary, and stops following the adversary's instructions. (The fact that P may not "know" whether it was corrupted and recovered can be captured by making sure that the rest of the code run by P does not depend on the corruption and recovery messages.)

Physical ("side channel") attacks. A practical and very realistic security concern is protection against "physical attacks" on computing devices, where the attacker is able to gather information on, and sometimes even modify, the internal computation of a device via physical access to it. (Famous examples include the "timing attack" of [K96], the "microwave attacks" of [BDL97, BS97] and the "power analysis" attacks of [CJRR99].) These attacks are often dubbed "side-channel" attacks in the literature. Some formalizations of security against such attacks appear in [MR04, GLMMR04].

This type of attacks can be directly modeled via different reaction patterns of parties to corruption requests. Security against such attacks is then defined as usual (i.e., by realizing an ideal functionality), where the corruption model is chosen appropriately. For instance, the ability of the adversary to observe or modify certain memory locations, or to detect whenever a certain internal operation (such as modular multiplication) takes place, can be directly modeled by having the party send the relevant information to the adversary.

Coercion. In a Coercion attack an external entity tries to influence the input that the attacked party contributes to a computation, without physically controlling the attacked party at the time where the input is being contributed. The typical method is by asking the corrupted party to provide some private information at some later point in time, and by comparing this information with the public transcript of the protocol at the time where the input was contributed. Resilience to coercion is an important requirement in setting where the participants are humans that are susceptible to social pressure, such as in voting schemes.

In the present framework, coercion attacks can be modeled in a straightforward way, along the lines of [CG96]. That is, in a coercion attack the attacked party can run some pre-determined algorithm for modifying its internal state before handing it over to the adversary.

6.2 Writing conventions for ideal functionalities

Specifying the identities of the calling parties. Recall that an instance of an ideal functionality $\mathcal{F}$ with SID sid accepts inputs only from parties $P = (sid_P, pid_P)$ whose SID equals the local SID, i.e. $sid_P = sid$. Similarly, this instance generates outputs only to parties whose SID equals sid. Consequently, when describing an ideal functionality, we allow ourselves to say "receive input from party P" and "generate output for party P", where P specifies only the PID of the corresponding party. The intention is that the input is to be received from the party (sid, P) or sent to party (sid, P), whichever is the case, where sid is the local SID.

Also, it will often be convenient to include within the SID the identity of the initiator (namely, the identity of the party that invokes $\mathcal{F}$), and sometimes even the identities of all participants. Including the initiator's identity in the SID guarantees global uniqueness of the SID, as long as the rest of the SID is locally unique for the initiator. Including the intended identities of other participants provides a simple mechanism for parties to know whether they should participate in this protocol instance. (Of course, this method is applicable only when the initiator knows the identities of the other intended participants in advance.)

Note that the SID is typically treated as a public value, so if it is desired that the identities of the participants be kept secret then some aliasing mechanism should be used (say, the SID includes a list of aliases and the functionality is privately given the identity that corresponds to each alias).

Behavior upon party corruption. Recall that, in the ideal processes, corruption of a parties is modeled as messages sent by the adversary to the ideal functionality. The behavior of the functionality upon receiving such a message is not determined by the model. We say that an ideal functionality $\mathcal{F}$ is standard corruption if, upon receiving a (`corrupt` P) message from the adversary $\mathcal{S}$, $\mathcal{F}$ locally records P as corrupted, and returns to the adversary all the inputs and outputs of P so far. In addition, from this point on, whenever $\mathcal{F}$ gets an input value v from P, it forwards v to the adversary, and receives a "modified input value" v' from the adversary. Also, all output values intended for P are sent to the adversary instead. This captures the standard behavior of the ideal process upon corruption of a party in existing definitional frameworks. The functionalities presented in this section are standard-corruption unless explicitly said otherwise.[10]

Delayed output. Recall that an output from an ideal functionality to a party is read by the recipient immediately, in the next activation. In contrast, we often want to be able to represent the fact that outputs generated by distributed protocols are inevitably delayed due to delays in message delivery. One natural way to relax an ideal functionality along these lines is to have the functionality "ask for the permission of the adversary" before generating an output. More precisely, we say that an ideal functionality $\mathcal{F}$ sends a delayed output v to party P if it engages in the following interaction: Instead of simply outputting v to P, $\mathcal{F}$ first sends to the adversary a note that it is ready to generate an output to P. If the output is public, then the value v is included in the note to the adversary. If the output is private then v is not not mentioned in this note. Furthermore, the note contains a unique identifier that distinguishes it from all other messages sent by $\mathcal{F}$ to the adversary in this execution. When the adversary replies to the note (say, by echoing the unique identifier), $\mathcal{F}$ outputs the value v to P.

[10] We remark that other behavior patterns upon party corruption are sometimes useful in capturing realistic concerns. For instance, *forward secrecy* can be captured by making sure that the adversary does not obtain past inputs or outputs of the party even when the party is corrupted.

Sending public delayed output v to a set $\mathcal{P}$ of parties is carried out as follows: First, $\mathcal{F}$ sends $(v, \mathcal{P})$ to the adversary. Then, whenever the adversary responds with an identity $P \in \mathcal{P}$, $\mathcal{F}$ outputs v to P. If the output is private then the procedure is the same except that $\mathcal{F}$ does not send v to the adversary. Sending delayed output which is partially private and partially public is defined analogously.

Running arbitrary code. It is often convenient to let an ideal functionality $\mathcal{F}$ receive a description of an arbitrary code c from parties, or even directly from the adversary, and then run this code while inspecting some properties of it. One use of such technique is for writing ideal functionalities with only minimal, well-specified requirements from the implementation. For instance, $\mathcal{F}$ may receive from the adversary a code for an algorithm; it will then run this algorithm as long as some set of security or correctness properties are satisfied. If a required property is violated, $\mathcal{F}$ will output an error messages to the relevant parties. Examples of this use include the signature and encryption functionalities as formalized in [CH04, C05]. Another use for this "programming technique" is to enable expressing the requirement that some adversarial processes be carried out in isolation from the external environment the the protocol runs in. An example for this use is the formulation of non-concurrent security in Section 6.6.

At first glance, this technique seems problematic in that $\mathcal{F}$ is expected to run algorithms of arbitrary polynomial running time, whereas its runtime is bounded by some fixed polynomial. We get around this problem by assuming that the entity that provides the code c explicitly invokes an ITI γ that runs the program c; γ then takes inputs from $\mathcal{F}$ and providing outputs to $\mathcal{F}$. This way, the "burden of providing sufficient resources to run c" is shifted to the entity that provides c. Note however that the length of the outputs of c should be bound by the polynomial bounding $\mathcal{F}$, otherwise $\mathcal{F}$ will not be able to read these outputs. Also, recall that the basic model allows $\mathcal{F}$ to know the real code run by this ITI. In particular, $\mathcal{F}$ can make sure that c does not provide any illegitimate output to parties other than $\mathcal{F}$.

6.3 Authenticated Communication

Ideally authenticated message transmission means that a party R will receive a message m from some party S only if S has sent the message m to R. Furthermore, if S sent m to R only t times then R will receive m from S at most t times. These requirements are of course meaningful only as long as both S and R are uncorrupted. In the case of adaptive corruptions, they are meaningful only if both S and R are uncorrupted at the time when R receives the message.

In the present framework, protocols that assume ideally authenticated message transmission can be cast as hybrid protocols with ideal access to an "authenticated message transmission functionality". This functionality, denoted $\mathcal{F}_{\text{AUTH}}$, is presented in Figure 11. $\mathcal{F}_{\text{AUTH}}$ first waits to receive an input $(\texttt{Send}, sid, m)$ from a party with PID S and SID sid. We require that the PIDs of the sender and the receiver are encoded in the SID. (As discussed in Section 6.2, this convention essentially provides each party with a "reserved name-space" for SIDs of instances of $\mathcal{F}_{\text{AUTH}}$ where it is the sender. It also allows the receiver to verify that it is the intended receiver, and guarantees that the receiver identity is fixed in advance, even when the sender is corrupted.) $\mathcal{F}_{\text{AUTH}}$ then generates a public delayed output $(\texttt{Send}, sid, m)$ to R. That is, $\mathcal{F}_{\text{AUTH}}$ first sends this value to the adversary. When the adversary responds, $\mathcal{F}_{\text{AUTH}}$ writes this value to the subroutine output tape of R. In addition, if the adversary instructs to corrupt the sender before the output value was actually delivered to R, then $\mathcal{F}_{\text{AUTH}}$ allows the adversary to provide new, arbitrary message m'. Then, $\mathcal{F}_{\text{AUTH}}$ outputs $(\texttt{Send}, sid, m')$ to R and halts. Corrupting the receiver has no effect.

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

1. Upon receiving an input $(\texttt{Send}, sid, m)$ from party S, do: If $sid = (S, R, sid')$ for some R, then generate a public delayed output $(\texttt{Sent}, sid, m)$ to R and halt. Else ignore the input.
2. Upon receiving $(\texttt{Corrupt-sender}, sid, m')$ from the adversary, and if the $(\texttt{Sent}, sid, m)$ output is not yet delivered to R, then output $(\texttt{Sent}, sid, m')$ to R and halt.

Figure 11: The Message Authentication functionality, $\mathcal{F}_{\text{AUTH}}$

We highlight several points regarding the security guarantees provided by $\mathcal{F}_{\text{AUTH}}$. First, $\mathcal{F}_{\text{AUTH}}$ reveals the contents of the message to the adversary. This captures the fact that secrecy of the message is not guaranteed. Second, the output is delayed, namely $\mathcal{F}_{\text{AUTH}}$ delivers a message only when the adversary responds, even if both the sender and the receiver are uncorrupted. This means that $\mathcal{F}_{\text{AUTH}}$ allows the adversary to delay a message indefinitely, and even to block delivery altogether. Third, $\mathcal{F}_{\text{AUTH}}$ allows the adversary to change the contents of the message, as long as the sender is corrupted *at the time of delivery,* even if the sender was uncorrupted at the point when it sent the message. This provision captures the fact that in general the received value is not determined until the point where the recipient actually generates its output.[11] Fourth, $\mathcal{F}_{\text{AUTH}}$ guarantees "non-transferable authentication": By interacting with $\mathcal{F}_{\text{AUTH}}$, the receiver R does not gain ability to run protocols with a third party V whereby V reliable learns that the message was indeed sent by the sender. In situations where this strong guarantee is not needed, it might suffice to use an appropriately relaxed variant of $\mathcal{F}_{\text{AUTH}}$.

Next, let us highlight two modeling aspects of $\mathcal{F}_{\text{AUTH}}$. First, $\mathcal{F}_{\text{AUTH}}$ deals with authenticated transmission of a *single message.* Authenticated transmission of multiple messages is obtained by using multiple instances of $\mathcal{F}_{\text{AUTH}}$, and relying on the universal composition theorem for security. This is an important property: It allows different instances of protocols that use authenticated communication to use different instances of $\mathcal{F}_{\text{AUTH}}$, thereby making sure that these protocols can be analyzed per instance, independently of other instances. (This modeling also significantly simplifies the analysis of protocols that obtain authenticated communication.) Another modeling aspect is that $\mathcal{F}_{\text{AUTH}}$ generates an output for the receiver without requiring the receiver to provide any input. This means that the SID is determined exclusively by the sender, and there is no need for the sender and receiver to agree on the SID in advance.

On realizing $\mathcal{F}_{\text{AUTH}}$. $\mathcal{F}_{\text{AUTH}}$ is used not only as a formalization of the authenticated communication model. It also serves as a way for specifying the security requirements from authentication protocols. (As discussed earlier, the validity of this dual use comes from the universal composition theorem.) We very briefly summarize some basic results regarding the realizability of $\mathcal{F}_{\text{AUTH}}$.

As a first step, we note that it is *impossible* to realize $\mathcal{F}_{\text{AUTH}}$ in the bare model, except by protocols that never generate any output. That is, say that a protocol is useful if there exists a party that generates output with non-negligible probability for some PPT environment and adversary. Then, we have:

[11] Previous formulations of of $\mathcal{F}_{\text{AUTH}}$ (e.g., the one in [C01]) fails to let the adversary change the delivered message and recipient identity if the sender gets corrupted between sending and delivery. This results in an unrealistically strong security guarantee, that is not intuitively essential and is not provided by reasonable authentication protocols. This oversight was pointed out in several places, including [HMS03, AF04].

Claim 17 ([c04]) *There exist no useful protocols that UC-realize $\mathcal{F}_{\text{AUTH}}$ in the bare model.*

Still, there are a number of ways to realize $\mathcal{F}_{\text{AUTH}}$ in algorithmic ways, given some other abstractions on the systems. Following the same definitional approach, these abstractions are again formulated by way of ideal functionalities. One such abstraction is allowing the parties to communicate in an ideally authenticated way at some preliminary stage. Another abstraction postulates existence of a trusted "registration service", where parties can register public values (e.g., keys) that can be authentically obtained by other parties upon request. These abstractions come in a number of flavors, that significantly influence the feasibility and efficiency of realizing $\mathcal{F}_{\text{AUTH}}$. See more discussion in Appendix A.2.

6.4 Secure Communication

The abstraction of secure communication, often called secure message transmission, usually means that the communication is authenticated, and in addition the adversary has no access to the contents of the transmitted message. It is typically assumed that the adversary learns that a message was sent, plus some partial information on the message (such as, say, its length, or more generally some information on the domain from which the message is taken). In the present framework, having access to an ideal secure message transmission mechanism can be cast as having access to the "secure message transmission functionality", $\mathcal{F}_{\text{SMT}}$, presented in Figure 12. The behavior of $\mathcal{F}_{\text{SMT}}$ is similar to that of $\mathcal{F}_{\text{AUTH}}$ with the following exception. $\mathcal{F}_{\text{SMT}}$ is parameterized by a leakage function $l : \{0,1\}^* \to \{0,1\}^*$ that captures the allowed information leakage on the transmitted plaintext m. That is, the adversary only learns the leakable information $l(m)$ rather than the entire m. (In fact, $\mathcal{F}_{\text{AUTH}}$ can be regarded as the special case of $\mathcal{F}^l_{\text{SMT}}$ where l is the identity function.)

Functionality $\mathcal{F}^l_{\text{SMT}}$

$\mathcal{F}^l_{\text{SMT}}$ proceeds as follows, when parameterized by leakage function $l : \{0,1\}^* \to \{0,1\}^*$.

1. Upon receiving an input $(\texttt{Send}, sid, m)$ from party S, do: If $sid = (S, R, sid')$ for some R then send $(\texttt{Sent}, sid, l(m))$ to the adversary, generate a private delayed output $(\texttt{Sent}, sid, m)$ to R and halt. Else ignore the input.
2. Upon receiving $(\texttt{Corrupt}, sid, P)$ from the adversary, where $P \in \{S, R\}$, disclose m to the adversary. Next, if the adversary provides a value m', and $P = S$, and no output has been yet written to R, then output $(\texttt{Sent}, sid, m')$ to R and halt.

Figure 12: The Secure Message Transmission functionality parameterized by leakage function l.

Like $\mathcal{F}_{\text{AUTH}}$, $\mathcal{F}_{\text{SMT}}$ only deals with transmission of a single message. Secure transmission of multiple messages is obtained by using multiple instances of $\mathcal{F}_{\text{SMT}}$. In addition, like $\mathcal{F}_{\text{AUTH}}$, $\mathcal{F}_{\text{SMT}}$ allows the adversary to change the contents of the message and the identity of the recipient as long as the sender is corrupted *at the time of delivery,* even if the sender was uncorrupted at the point when it sent the message. In addition, following our convention regarding party corruption, when either the sender or the receiver are corrupted, $\mathcal{F}_{\text{SMT}}$ discloses to the adversary all the inputs of the sender since the beginning of the computation.

Stronger variants. One common requirement from secure message transmission protocols is forward secrecy: the transmitted message should remain secret, even if the the sender and receiver of the message are corrupted *after* the protocol execution is completed. A natural way to capture forward secrecy in the present formulation is to modify the behavior upon corruption of either the sender or the receiver, so as to not disclose the plaintext message m to the adversary. Note that here $\mathcal{F}_{\text{SMT}}$ need not be changed at all.

Another common requirement is protection from traffic analysis. Recall that, whenever a party S sends a message to some R, $\mathcal{F}_{\text{SMT}}$ notifies the adversary that S sent a message to R. This reflects the common view that encryption does not hide the fact that a message was sent. (Using common terminology, there is no protection against traffic analysis.) To capture security against traffic analysis, modify $\mathcal{F}_{\text{SMT}}$ so that the adversary does not learn that a message was sent, or alternatively it does not learn the identities of the sender or the receiver.

On realizing $\mathcal{F}_{\text{SMT}}$. Protocols that UC-realize $\mathcal{F}_{\text{SMT}}$ can be constructed, based on public-key encryption schemes that are semantically secure against chosen plaintext attacks, by using each encryption key for encrypting only a single message, and authenticating the communication via $\mathcal{F}_{\text{AUTH}}$. That is, let $E = (gen, enc, dec)$ be an encryption scheme for domain D of plaintexts. (Here gen is the key generation algorithm, enc is the encryption algorithm, dec is the decryption algorithm, and correct decryption is guaranteed for any plaintext in D.) Then, consider the following protocol, denoted π_E. When invoked within S with input $(\texttt{Send}, sid, R, m)$ with $m \in D$, π_E first sends an $(\texttt{init}, sid)$ message to R. R then runs algorithm gen, gets the secret key sk and the public key pk, and sends (sid, pk) back to S. Next, S sends $(sid, enc(pk, m))$ to R and returns. Finally, upon receipt of (sid, c), π_E within R outputs $dec(sk, c)$ and returns.

It can be verified that the above protocol UC-realizes $\mathcal{F}_{\text{SMT}}$ as long as the underlying encryption scheme is semantically secure against chosen plaintext attacks. That is, given a domain D of plaintexts, let l_D be the "leakage function" that, given input x, returns $\perp$ if $x \in D$ and returns x otherwise. Then:

Claim 18 *If E is semantically secure for domain D as in* [GM84, G01] *then π_E UC realizes $\mathcal{F}_{\text{SMT}}^{l_D}$ in the presence of non-adaptive adversaries.*

Furthermore, if E is non-committing (as in [CFGN96]*) then π_E UC-realizes $\mathcal{F}_{\text{SMT}}^{l_D}$ with adaptive adversaries. This holds even if data erasures are not trusted and the adversary sees all the past internal states of the corrupted parties.*

Choosing new keys for each message to be transmitted is of course highly inefficient and does not capture common practice for achieving secure communication. As in the case of $\mathcal{F}_{\text{AUTH}}$, it is possible to realize multiple instances of $\mathcal{F}_{\text{SMT}}$ using a single instance of a more complex protocol, in a way that is considerably more efficient than running multiple independent instances of a protocol that realizes $\mathcal{F}_{\text{SMT}}$. One way of doing this is to use the same encryption scheme to encrypt all the messages sent to some party. Here the encryption scheme should have additional properties on top of being semantically secure. In [CKN03] it is shown that replayable chosen ciphertext security (RCCA) suffices for this purpose for the case of non-adaptive party corruptions. In the case of adaptive corruptions stronger properties and constructions are needed, see further discussion in [N02, CHK05].

6.5 Synchronous communication

A common and convenient abstraction of communication networks is that of *synchronous* communication. Roughly speaking, here the computation proceeds in *rounds,* where in each round each party receives all the messages that were sent to it in the previous round, and generates outgoing messages for the next round. There are of course many variants of the synchronous model. We concentrate here on modeling one basic variant, which provides the following guarantee:

Timely delivery. Each message sent by an uncorrupted party is guaranteed to arrive in the next communication round. In other words, no party will receive messages sent at round r before all uncorrupted parties had a chance to receive the messages sent at round $r - 1$.

This guarantee implies in essence two other ones:

Guaranteed delivery. Each party is guaranteed to receive all the messages that were sent to it.

Authentic delivery. Each message sent by an uncorrupted party is guaranteed to arrive unmodified. Furthermore, the recipient knows the real sender identity of each message.

Typically, the order of activation of parties within a round is assumed to be under the control of the adversary, thus the messages sent by the corrupted parties may depend on the messages sent by the uncorrupted parties in the *same round.*

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

$\mathcal{F}_{\text{SYN}}$ expects its SID to be of the form $sid = (\mathcal{P}, sid')$, where $\mathcal{P}$ is a list of party identities among which synchronization is to be provided. It proceeds as follows.

1. At the first activation, initialize a round counter $r \leftarrow 1$, and send a public delayed output $(\texttt{Init}, sid)$ to all parties in $\mathcal{P}$.
2. Upon receiving input $(\texttt{Send}, sid, M)$ from a party $P \in \mathcal{P}$, where $M = \{(m_i, R_i)\}$ is a set of pairs of messages m_i and recipient identities $R_i \in \mathcal{P}$, record (P, M, r) and output (sid, P, M, r) to the adversary. (If P later becomes corrupted then the record (P, M, r) is deleted.)
3. Upon receiving a message $(\texttt{Advance-Round}, sid, N)$ from the adversary, do: If there exist uncorrupted parties $P \in \mathcal{P}$ for which no record (P, M, r) exists then ignore the message. Else:
   (a) Interpret N as the list of messages sent by corrupted parties in this round. That is, $N = \{(S_i, R_i, m_i)\}$ where each $S_i, R_i \in \mathcal{P}$, m is a message, and S_i is corrupted. (S_i is taken as the sender of message m_i and R_i is the receiver.)
   (b) Prepare for each party $P \in \mathcal{P}$ the list L_P^r of messages that were sent to it in round r.
   (c) Increment the round number: $r \leftarrow r + 1$.
4. Upon receiving input $(\texttt{Receive}, sid)$ from a party $P \in \mathcal{P}$, output $(\texttt{Received}, sid, r, L_P^{r-1})$ to P. (Let $L_P^0 = \perp$.)

Figure 13: The synchronous communication functionality, $\mathcal{F}_{\text{SYN}}$.

Synchronous variants of the UC framework are presented in [N03, HM04a]. Here we provide an alternative way of capturing synchronous communication within the UC framework: We show how synchronous communication can be captured by having access to an ideal functionality $\mathcal{F}_{\text{SYN}}$ that provides the above guarantees. We first describe $\mathcal{F}_{\text{SYN}}$, and then discuss and motivate some aspects of its design. $\mathcal{F}_{\text{SYN}}$ is presented in Figure 13. It expects its SID to include a list $\mathcal{P}$ of parties among which synchronization is to be provided. At the first activation, $\mathcal{F}_{\text{SYN}}$ initializes a round number r to 1, and sends a delayed public output `init` to all parties in $\mathcal{P}$. (The purpose of this output is to notify the parties in $\mathcal{P}$ that a synchronous session with the given SID has started; in particular, it frees the calling protocol from the need to agree on the SID in advance.) Next, $\mathcal{F}_{\text{SYN}}$ responds to three types of inputs: Given input of the form $(\texttt{Send}, sid, M)$ from party $P \in \mathcal{P}$, $\mathcal{F}_{\text{SYN}}$ interprets M as a list of messages to be sent to other parties in $\mathcal{P}$. The list M is recorded together with the sender identity and the current round number, and is also forwarded to the adversary. Given a request $(\texttt{Advance-Round}, sid, N)$ from the adversary, where N is a list of messages sent from the corrupted parties to the uncorrupted ones, $\mathcal{F}_{\text{SYN}}$ first verifies that all uncorrupted parties sent messages in this round. If yes, then $\mathcal{F}_{\text{SYN}}$ advances the round number. If not, then the request is ignored. This provision guarantees that a round does not end before all parties have a chance to send messages. Upon receiving a $(\texttt{Corrupt}, P)$ from the adversary, for some $P \in \mathcal{P}$, $\mathcal{F}_{\text{SYN}}$ marks P as corrupted. If P gets corrupted after it has sent its messages for this round, but before the round advances, then at this time the adversary has the opportunity to "rewrite" the messages that were sent by P at this round. In any case, once the round number advances, the set of messages sent at the round is fixed. Finally, given an input $(\texttt{Receive}, sid)$ from a party $P \in \mathcal{P}$, $\mathcal{F}_{\text{SYN}}$ returns to P all the messages that were sent to P at the previous round, together with the round number. It is stressed that $\mathcal{F}_{\text{SYN}}$ does not deliver messages to a party until being explicitly requested by the party to obtain the messages. This formulation facilitates capturing guaranteed delivery of messages within the present framework; see more discussion below.[12]

To highlight the properties of $\mathcal{F}_{\text{SYN}}$, let us sketch a typical use of $\mathcal{F}_{\text{SYN}}$ by some $\mathcal{F}_{\text{SYN}}$-hybrid protocol π. All parties of π use the same instance of $\mathcal{F}_{\text{SYN}}$. This instance can be invoked by any of the parties, or even the adversary. If the parties know in advance the SID of this instance then they can send messages to it right away. Otherwise, they can wait for the `init` message where the SID is specified. In any case, each party of π first initializes a round counter to 1, and inputs to $\mathcal{F}_{\text{SYN}}$ a list M of first-round messages to be sent to the other parties of π. In each subsequent activation, the party calls $\mathcal{F}_{\text{SYN}}$ with a $(\texttt{Receive}, sid)$ input (where sid is typically derived from the current SID of π). If the round number in the response from $\mathcal{F}_{\text{SYN}}$ is no larger than the local round number then this means that the global round number has not yet advanced, and so the response is ignored. Else, the party obtains the list of messages received in this round, performs its local processing, increments the local round number, and call $\mathcal{F}_{\text{SYN}}$ again with input $(\texttt{Send}, sid, M)$ where M contains the outgoing messages for this round. If the list of incoming messages is empty then π halts.

It can be seen that the message delivery pattern for such a protocol π is essentially the same as in a traditional synchronous network. Indeed, $\mathcal{F}_{\text{SYN}}$ requires that all parties actively participate in the computation in each round. That is, the round counter does not advance until all uncorrupted parties are activated at least once and send a (possibly empty) list of messages for that round. Furthermore, as soon as one uncorrupted party is able to obtain its incoming messages for some round, all uncorrupted parties are able to obtain their messages for that round. Consequently, if the adversary fails to advance the round number then the computation effectively halts altogether.

The present formulation of $\mathcal{F}_{\text{SYN}}$ does not guarantee "fairness", in the sense that the adversary may obtain the messages sent by the uncorrupted parties for the last round while the uncorrupted parties may have not received these messages. (This might happen if the adversary fails to advance the round.) To guarantee fairness, modify $\mathcal{F}_{\text{SYN}}$ so that the adversary learns the messages sent by the uncorrupted parties in a round only after it advances the round.

[12] An alternative formulation of $\mathcal{F}_{\text{SYN}}$ would advance a round only if all uncorrupted parties in $\mathcal{P}$ have requested to *read* their messages for the previous round. This formulation would have the same effect as the present one.

Another point worth elaboration is that each instance of $\mathcal{F}_{\text{SYN}}$ guarantees synchronous message delivery only within the context of the messages sent using that instance. Delivery of messages sent in other ways (e.g., directly or via other instances of $\mathcal{F}_{\text{SYN}}$) may be arbitrarily faster or arbitrarily slower. This allows capturing, in addition to the traditional model of a completely synchronous network where everyone is synchronized, also more general settings such as synchronous execution of a protocol within a larger asynchronous environment, or several protocol executions where each execution is internally synchronous but the executions are mutually asynchronous.

Also note that, even when using $\mathcal{F}_{\text{SYN}}$, the inputs to the parties are received in an "asynchronous" way, i.e. it is not guaranteed that all inputs are received within the same round. Still, a protocol that uses $\mathcal{F}_{\text{SYN}}$ can deploy standard mechanisms for guaranteeing that the actual computation does not start until enough (or all) parties have inputs.

Finally, including the set $\mathcal{P}$ of participants in the SID captures cases where the identities of all participants are known to the initiator in advance. Alternative situations, where the set of participants is not known a priori can be captured by letting parties join in as the computation proceeds, and have $\mathcal{F}_{\text{SYN}}$ update the set $\mathcal{P}$ accordingly.

Potential relaxations. The reliability and authenticity guarantees provided within a single instance of $\mathcal{F}_{\text{SYN}}$ are quite strong: Once a round number advances, all the messages to be delivered to the parties at this round are fixed, and are guaranteed to be delivered upon request. It is of course possible to relax $\mathcal{F}_{\text{SYN}}$ by, say, allowing the adversary to stop delivering messages or to modify messages sent by corrupted parties even in "mid-round".

Another potential relaxation of $\mathcal{F}_{\text{SYN}}$ is to relax the "timeliness" guarantee. Specifically, it may only be guaranteed that messages are delivered within a given number, δ, of rounds from the time they are generated. The bound δ may be either known in advance or alternatively unknown and determined dynamically (e.g., specified by the adversary when the message is sent). The case of known delay δ corresponds to the "timing model" of [DNS98, G02, LPT04]. The case of unknown delay corresponds to the model of *non-blocking asynchronous communication model* where message are guaranteed to be delivered, but with unknown delay (see, e.g., [BCG93, CR93]).

On composing $\mathcal{F}_{\text{SYN}}$-hybrid protocols. Recall that each instance of $\mathcal{F}_{\text{SYN}}$ represents a single synchronous system, and different instances of $\mathcal{F}_{\text{SYN}}$ are mutually asynchronous, even when they run in the same system. This means that different protocol instances that wish to be mutually synchronized would need to use the same instance of $\mathcal{F}_{\text{SYN}}$. In particular, if we have a complex, multi-component protocol that assumes a globally synchronous network, then all the components of this protocol would need to use the same instance of $\mathcal{F}_{\text{SYN}}$.

A priori, this observation may seem to prevent the use of the universal composition operation for constructing such protocols, since this operation does not allow the composed protocols to have any joint subroutines. We note, however, that protocol instances that use the same instance of $\mathcal{F}_{\text{SYN}}$ can be composed using a tool called universal composition with joint state (see Appendix A.2). This tool allows analyzing each such protocol instance as if it is the only instance using $\mathcal{F}_{\text{SYN}}$, while still asserting security of the system where multiple protocol instances use the same instance of F_{SYN}. A key observation here is that one can straightforwardly realize multiple independent instances of $\mathcal{F}_{\text{SYN}}$ using a single instance of $\mathcal{F}_{\text{SYN}}$, whose set of participants is the union of the sets of participants of the realized instances.

6.6 Non-concurrent Security

One of the main features of the UC framework is that it guarantees security even when protocol instances are running concurrently in an adversarially controlled manner. Still, sometimes it may be useful to capture within the UC framework also security properties that are not necessarily preserved under concurrent composition, and are thus realizable by simpler protocols or with milder setup assumptions.

This section provides a way to express such "non-concurrent" security properties within the present framework. Specifically, we present an ideal functionality, $\mathcal{F}_{\text{NC}}$, that guarantees to the calling protocol instance that no other parties are activated as long as the current instance is running. other protocol instances are running concurrently in the system. Thus, modeling a protocol as an $\mathcal{F}_{\text{NC}}$-hybrid protocol is paramount to analyzing this protocol in a system where there are no protocol instances running concurrently.

Functionality $\mathcal{F}_{\text{NC}}$ is presented in Figure 14. It first expects to receive a code of an adversary, $\hat{\mathcal{A}}$. It then behaves as adversary $\hat{\mathcal{A}}$ would in the non-concurrent security model. That is, $\mathcal{F}_{\text{NC}}$ runs $\hat{\mathcal{A}}$ and follows its instructions with respect to receipt and delivery of messages between parties. As soon as $\hat{\mathcal{A}}$ terminates the execution, $\mathcal{F}_{\text{NC}}$ reports the current state of $\hat{\mathcal{A}}$ back to the external adversary, and halts. (Recall the convention from Section 6.2 regarding running arbitrary code.)

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

1. When receiving message $(\texttt{Start}, sid, \hat{\mathcal{A}})$ from the adversary, where $\hat{\mathcal{A}}$ is the code of an ITM (representing an adversary), invoke $\hat{\mathcal{A}}$ and change state to `running`.
2. When receiving input $(\texttt{Send}, sid, m, Q)$ from party P, and if the state is `running`, activate $\hat{\mathcal{A}}$ with incoming message (m, Q) from party P. Then:
   (a) If $\hat{\mathcal{A}}$ instructs to deliver a message (m', P') to party Q' then output (sid, m', P') to Q'.
   (b) If $\hat{\mathcal{A}}$ halts without delivering a message to any party then send the current state of $\hat{\mathcal{A}}$ to the adversary and halt.

Figure 14: The non-concurrent communication functionality, $\mathcal{F}_{\text{NC}}$.

We remark that, in addition to analyzing "pure" non-concurrent security of protocols, $\mathcal{F}_{\text{NC}}$ can also be used to analyze systems where some of the components may be running concurrently with each other, where other components cannot. Here the "non-composable" components can be written as $\mathcal{F}_{\text{NC}}$-hybrid protocols. For some concrete examples see Section A.2.

Non-concurrent security of protocols that assume some idealized communication model (such as, say, authenticated or synchronous communication) can be captures by using $\mathcal{F}_{\text{NC}}$ in conjunction with the ideal functionality that captures the desired model. For instance, to capture non-concurrent security of synchronous protocols, let the protocol use $\mathcal{F}_{\text{SYN}}$ as described in Section 6.5, with the exception that $\mathcal{F}_{\text{SYN}}$ interacts with $\mathcal{F}_{\text{NC}}$ instead of interacting directly with the adversary. ($\mathcal{F}_{\text{NC}}$, in turn, interacts with the adversary as described above.) We call such protocols $\mathcal{F}_{\text{NC}}$-$\mathcal{F}_{\text{SYN}}$-hybrid protocols.

Equivalence with the definition of [C00]**.** Recall the security definition of [C00], that guarantees that security is preserved under non-concurrent composition of protocols. (See discussion in Section 1.1.) More specifically, recall that the [C00] notion, which we call non-concurrent security, is the same as UC security with the following exception: In the case of non-concurrent security, the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ are prohibited from interacting (i.e., they cannot send inputs and outputs to each other) from the moment where the first protocol message is sent until the moment where the last protocol message is delivered. More specifically, say that an environment is non-concurrent if it does not provide any input to the adversary other than the input provided to the adversary at its first activation, and furthermore does not provide any inputs to the protocol parties after receiving the first output of the adversary. Then:

Definition 19 *Let π and ϕ be PPT protocols. We say that π* NC-emulates *ϕ if for any PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any non-concurrent PPT environment $\mathcal{Z}$, we have $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.*

Recall that the [C00] framework concentrates on synchronous communication networks (or in other words on $\mathcal{F}_{\text{SYN}}$-hybrid protocols). Furthermore, it considers only the task of secure function evaluation. Still, this section considers general protocols and tasks; treatment of $\mathcal{F}_{\text{SYN}}$-hybrid protocols for secure function evaluation is obtained as a special case.

An important feature of non-concurrent security is that it is easier to realize. In particular, known protocols (e.g., the protocol of [GMW87], see also [G04]) for realizing a general class of ideal functionality with any number of faults, assuming authenticated communication as the only set-up assumption, can be shown secure in this model. This stands in contrast to the impossibility results regarding the realizability of the same functionalities in the UC framework, even with authenticated communication.

We argue that having access to $\mathcal{F}_{\text{NC}}$ is essentially equivalent to running in the non-concurrent security model described above. More specifically, say that a protocol π is NC if all protocol messages are sent via $\mathcal{F}_{\text{NC}}$. Then, an NC protocol π UC-realizes some ideal functionality $\mathcal{F}$ if and only if π realizes $\mathcal{F}$ with non-concurrent security. More generally:

Proposition 20 *Let π and ϕ be protocols, where π is NC. Then π UC-emulates ϕ if and only if π NC-emulates ϕ.*

Notice that Proposition 20, together with the UC theorem, provide an alternative (albeit somewhat roundabout) formulation of the non-concurrent composition theorem of [C00].

Proof: Clearly if π UC-emulates ϕ then π NC-emulates ϕ. For the other direction, assume that π NC-emulates ϕ. This means that for any adversary $\mathcal{A}$ there exists an adversary (simulator) $\mathcal{S}$ such that $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ for all non-concurrent environments $\mathcal{Z}$. To show that π UC-emulates ϕ, consider an interaction of π with the dummy adversary $\mathcal{D}$. Since π is an $\mathcal{F}_{\text{NC}}$-hybrid protocol, the environment $\mathcal{Z}$ must now provide some code $\hat{\mathcal{A}}$ to $\mathcal{F}_{\text{NC}}$. From this point on, $\mathcal{Z}$ sees no outgoing messages from any of the participants of π. Furthermore, it only receives one message from $\mathcal{F}_{\text{NC}}$, describing the "final" state of $\hat{\mathcal{A}}$. Once this message is given, no party generates any further output to $\mathcal{Z}$.

Now, consider the simulator $\hat{\mathcal{S}}$ such that $\text{EXEC}_{\phi,\hat{\mathcal{S}},\mathcal{Z}} \approx \text{EXEC}_{\pi,\hat{\mathcal{A}},\mathcal{Z}}$ for any non-concurrent $\mathcal{Z}$. (Existence of $\hat{\mathcal{S}}$ follows from the assumption that π NC-emulates ϕ.) We claim that $\text{EXEC}_{\phi,\hat{\mathcal{S}},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{D},\mathcal{Z}}$ even when $\mathcal{Z}$ is not necessarily non-concurrent. This is so since the interaction of $\mathcal{D}$ with π is limited in the same way as that of a non-concurrent environment. In other words, when interacting with $\mathcal{D}$ and an $\mathcal{F}_{\text{NC}}$-hybrid protocol π, any environment is effectively bound to be non-concurrent. $\square$

Acknowledgments

Much of the motivation for undertaking this project, and many of the ideas that appear here, come from studying secure key-exchange protocols together with Hugo Krawczyk. I thank him for this long, fruitful, and enjoyable collaboration. I am also grateful to Oded Goldreich who, as usual, gave me both essential moral support and invaluable technical and presentational advice.

Many thanks also to the many people with whom I have interacted over the years on definitions of security and secure composition. A very partial list includes Martin Abadi, Michael Backes, Mihir Bellare, Ivan Damgaard, Marc Fischlin, Shafi Goldwasser, Rosario Gennaro, Shai Halevi, Dennis Hofheinz, Yuval Ishai, Ralf Küsters, Eyal Kushilevitz, Yehuda Lindell, Phil MacKenzie, Tal Malkin, Cathy Meadows, Silvio Micali, Daniele Micciancio, Moni Naor, Rafi Ostrovsky, Rafael Pass, Birgit Pfitzmann, Tal Rabin, Charlie Rackoff, Phil Rogaway, Victor Shoup, Paul Syverson and Michael Waidner. In particular, it was Daniele who proposed to keep the basic framework minimal and model subsequent abstractions as ideal functionalities within the basic model.

References

[AFG98] M. Abadi, C. Fournet, and G. Gonthier. Secure implementation of channel abstractions. In LICS'98, pages 105–116, 1998.

[AG97] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *4th ACM Conference on Computer and Communications Security*, 1997, pp.36-47. Fuller version available at http://www.research.digital.com/SRC/ abadi.

[AR00] M. Abadi and P. Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). *J. Cryptology* 15(2): 103-127 (2002). Preliminary version at *International Conference on Theoretical Computer Science IFIP TCS 2000*, LNCS, 2000.

[AF04] M. Abe and S. Fehr. Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography. *Crypto '04*, 2004.

[A04] J. Almansa. The Full Abstraction of the UC Framework. BRICS, Technical Report RS-04-15 University of Aarhus, Denmark, August 2004. Also available at eprint.iacr.org/2005/019.

[AB01] J. An and M. Bellare. Does encryption with redundancy provide authenticity? *Eurocrypt 2001, LNCS 2045*, 2001.

[BH04] M. Backes and D. Hofheinz. How to Break and Repair a Universally Composable Signature Functionality. *7th Information Security Conference (ISC)*, LNCS 3225 pp.61–72. 2004.

[BPW03] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *10th ACM conference on computer and communications security (CCS)*, 2003. Extended version at the eprint archive, http://eprint.iacr.org/2003/015/.

[BPW03a] M. Backes, B. Pfitzmann, and M. Waidner. Symmetric Authentication Within a Simulatable Cryptographic Library. *8th European Symposium on Research in Computer Security, ESORICS '03*, LNCS 2808 pp. 271–290, 2003. Extended version at the eprint archive, http://eprint.iacr.org/2003/145.

[BPW04] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *1st Theory of Cryptography Conference (TCC)*, LNCS 2951 pp. 336–354, Feb. 2004.

[BPW04a] M. Backes, B. Pfitzmann, and M. Waidner. Secure Asynchronous Reactive Systems. Eprint archive, http://eprint.iacr.org/2004/082, March 2004.

[BP04] M. Backes and B. Pfitzmann. Symmetric Encryption in a Simulatable Dolev-Yao Style Cryptographic Library. *17th IEEE Computer Security Foundations Workshop (CSFW)*, pp. 204–218, 2004. Extended version at the eprint archive, http://eprint.iacr.org/2004/059.

[B01] B. Barak. How to go Beyond the Black-Box Simulation Barrier. In *42nd FOCS*, pp. 106–115, 2001.

[B+05] B. Barak, R. Canetti, Y. Lindell, R. Pass and T. Rabin. Secure Computation Without Authentication. In *Crypto'05*, 2005.

[BCNP04] B. Barak, R. Canetti, J. B. Nielsen, R. Pass. Universally Composable Protocols with Relaxed Set-Up Assumptions. *36th FOCS*, pp. 186–195. 2004.

[BGGL04] B. Barak, O. Goldreich, S. Goldwasser and Y. Lindell. Resettably-Sound Zero-Knowledge and its Applications. *42nd FOCS*, pp. 116-125, 2001.

[BLR04] B. Barak, Y. Lindell and T. Rabin. Protocol Initialization for the Framework of Universal Composability. Eprint archive. eprint.iacr.org/2004/006.

[BS05] B. Barak and A. Sahai, How To Play Almost Any Mental Game Over the Net - Concurrent Composition via Super-Polynomial Simulation. *FOCS*, 2005.

[B91] D. Beaver. Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. J. Cryptology, (1991) 4: 75-122.

[B96] D. Beaver. Adaptive Zero-Knowledge and Computational Equivocation. *28th Symposium on Theory of Computing (STOC)*, ACM, 1996.

[B97] D. Beaver. Plug and play encryption. *CRYPTO 97*, 1997.

[BH92] D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Eurocrypt '92*, LNCS No. 658, 1992, pages 307–323.

[BCK98] M. Bellare, R. Canetti and H. Krawczyk. A modular approach to the design and analysis of authentication and key-exchange protocols. *30th Symposium on Theory of Computing (STOC)*, ACM, 1998.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway. Relations among notions of security for public-key encryption schemes. *CRYPTO '98*, 1998, pp. 26-40.

[BR93] M. Bellare and P. Rogaway. Entity authentication and key distribution. *CRYPTO'93*, LNCS. 773, pp. 232-249, 1994.

[BR93*a*] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *1st Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[BCG93] M. Ben-Or, R. Canetti and O. Goldreich. Asynchronous Secure Computations. *25th Symposium on Theory of Computing (STOC),* ACM, 1993, pp. 52-61.

[BGW88] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. *20th Symposium on Theory of Computing (STOC),* ACM, 1988, pp. 1-10.

[BKR94] M. Ben-Or, B. Kelmer and T. Rabin. Asynchronous Secure Computations with Optimal Resilience. *13th PODC,* 1994, pp. 183-192.

[BM04] M. Ben-Or, D. Mayers. General Security Definition and Composability for Quantum & Classical Protocols. arXiv archive, http://arxiv.org/abs/quant-ph/0409062.

[BHLMO05] M. Ben-Or, M. Horodecki, D. Leung, D. Mayers, and J. Oppenheim. The Universal Composable Security of Quantum Key Distribution. *2nd Theoretical Cryptographic Conference (TCC),* 2005.

[BS97] E. Biham, A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. *CRYPTO '97,* pp. 513–525. 1997.

[B04] B. Blanchet. Automatic proof of strong secrecy for security protocols. *IEEE Security and Privacy Conference,* pages 86–102. 2004.

[B82] M. Blum. Coin flipping by telephone. IEEE Spring COMPCOM, pp. 133-137, Feb. 1982.

[BFM89] M. Blum, P. Feldman and S. Micali. Non-interactive Zero-Knowledge and its applications. *20th STOC,* 1988.

[BDL97] D. Boneh, R. A. DeMillo, R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). *Eurocrypt '97,* pp. 37–51. 1997.

[BCC88] G. Brassard, D. Chaum and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *JCSS,* Vol. 37, No. 2, pages 156–189, 1988.

[BAN90] M. Burrows, M. Abadi and R. Needham. A logic for authentication. DEC Systems Research Center Technical Report 39, February 1990. Earlier versions in *the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, 1988, and *the Twelfth ACM Symposium on Operating Systems Principles,* 1989.

[CL05] J. Camenisch, A. Lysyanskaya. A Formal Treatment of Onion Routing. *Crypto*, 2005.

[C95] R. Canetti. Studies in Secure Multi-party Computation and Applications.*Ph.D. Thesis,* Weizmann Institute, Israel, 1995.

[C98] R. Canetti. Security and composition of multi-party cryptographic protocols. ftp://theory.lcs.mit.edu/pub/tcryptol/98-18.ps, 1998.

[C00] R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology,* Vol. 13, No. 1, winter 2000.

[C01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Version of October 2001. Available at http://eccc.uni-trier.de/eccc-reports/2001/TR01-016/revision01.ps.

[C04] R. Canetti. Universally Composable Signature, Certification, and Authentication. *17th Computer Security Foundations Workshop (CSFW)*, 2004. Long version at eprint.iacr.org/2003/239.

[C05] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Version of December 2005. Available at http://eccc.uni-trier.de/eccc-reports/2001/TR01-016.

[C06] Ran Canetti. Security and composition of cryptographic protocols: A tutorial. SIGACT News, Vol. 37, Nos. 3 & 4, 2006. Available also at the Cryptology ePrint Archive, Report 2006/465.

[C07] R. Canetti. Obtaining Universally Composable Security: Towards the Bare Bones of Trust. ASIACRYPT 2007, pp. 88-112.

[C+05] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Using Probabilistic I/O Automata to Analyze an Oblivious Transfer Protocol. MIT Technical Report MIT-LCS-TR-1001, August 2005.

[CDDIM04] R. Canetti, I. Damgaard, S. Dziembowski, Y. Ishai, T. Malkin. On Adaptive vs. Non-adaptive Security of Multiparty Protocols. *J. Cryptology,* 17(3): 153-207. 2004.

[CDPW07] R. Canetti, Y. Dodis, R. Pass and S. Walfish. Universally Composable Security with Pre-Existing Setup. *4th theory of Cryptology Conference (TCC),* 2007.

[CFGN96] R. Canetti, U. Feige, O. Goldreich and M. Naor. Adaptively Secure Computation. *28th Symposium on Theory of Computing (STOC),* ACM, 1996. Fuller version in MIT-LCS-TR 682, 1996.

[CF01] R. Canetti and M. Fischlin. Universally Composable Commitments. Crypto '01, 2001.

[CG96] R. Canetti and R. Gennaro. Incoercible multi-party computation. *37th Symp. on Foundations of Computer Science (FOCS)*, IEEE, 1996.

[CGH04] R. Canetti, O. Goldreich and S. Halevi. The Random Oracle Methodology, Revisited. *Journal of the ACM,* v.51 n.4, p.557-594, July 2004. Preliminary version in *30th STOC,* 1998.

[CG99] R. Canetti and S. Goldwasser. A practical threshold cryptosystem resilient against adaptive chosen ciphertext attacks. *Eurocrypt '99,* 1999.

[CHH00] R. Canetti, S. Halevi and A. Herzberg. How to Maintain Authenticated Communication. *Journal of Cryptology,* Vol. 13, No. 1, winter 2000. Preliminary version at *16th Symp. on Principles of Distributed Computing (PODC),* ACM, 1997, pp. 15-25.

[CHK05] R. Canetti, S. Halevi and J, Katz. Adaptively Secure Non-Interactive Public-Key Encryption. *2nd theory of Cryptology Conference (TCC),* 2005.

[CHKLM05] R. Canetti, S. Halevi, J. Katz, Y. Lindell, P. Mackenzie. Universally Composable Password-Based Key Exchange. Eurocrypt '05, 2005. Extended version at http://eprint.iacr.org/2005/196.

[CH04] R. Canetti and J. Herzog. Universally Composable Symbolic Analysis of Cryptographic Protocols (The case of encryption-based mutual authentication and key exchange). Eprint archive, http://eprint.iacr.org/2004/334.

[CK01] R. Canetti and H. Krawczyk. Analysis of key exchange protocols and their use for building secure channels. Eurocrypt '01, 2001. Extended version at http://eprint.iacr.org/2001/040.

[CK02] R. Canetti and H. Krawczyk. Universally Composable Key Exchange and Secure Channels . In *Eurocrypt '02*, pages 337–351, 2002. LNCS No. 2332. Extended version at http://eprint.iacr.org/2002/059.

[CK02a] R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-based Key-Exchange Protocol. *Crypto '02*, 2002. Extended version at http://eprint.iacr.org/2002/120.

[CKN03] R. Canetti, H. Krawczyk, and J. Nielsen. Relaxing Chosen Ciphertext Security of Encryption Schemes. *Crypto '03*, 2003. Extended version at the eprint archive, eprint.iacr.org/2003/174.

[CKL03] R. Canetti, E. Kushilevitz, Y. Lindell. On the Limitations of Universally Composable Two-Party Computation without Set-up Assumptions. *J. Cryptology* 19(2): 135-167 (2006). (Preliminary version at Eurocrypt'03.)

[CKOR00] R. Canetti, E. Kushilevitz, R. Ostrovsky and A. Rosen. Randomness vs. Fault-Tolerance. *J. Cryptology* Vol. 13, No. 1, winter 2000. Preliminary version at *16th Symp. on Principles of Distributed Computing (PODC)*, ACM, 1997,

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, A. Sahai. Universally composable two-party and multi-party secure computation. *34th STOC*, pp. 494–503, 2002.

[CPS07] R. Canetti, R. Pass, and A. Shelat. Cryptography from sunspots: How to use an imperfect reference string. 48th Foundations of Computer Science (FOCS) 2007.

[CR93] R. Canetti and T. Rabin. Optimal Asynchronous Byzantine Agreement. *25th STOC*, 1993, pp. 42-51.

[CR03] R. Canetti and T. Rabin. Universal Composition with Joint State. *Crypto'03*, 2003.

[CJRR99] S. Chari, C. S. Jutla, J. R. Rao, P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. *CRYPTO '99*, pp. 398–412. 1999.

[CGMA85] B. Chor, S. Goldwasser, S. Micali and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. *26th FOCS*, 1985, pp. 383-395.

[CGP99] E. Clarke, O. Grunberg and E. Peled. *Model Checking*, MIT Press, 1999.

[CGS02] C. Crepeau, D. Gottesman and A. Smith. Secure Multi-Party Quantum Computation. *34th STOC*, 2002.

[DG03] I. Damgaard, J. Groth. Non-interactive and reusable non-malleable commitment schemes. *34th STOC*, pp. 426–437. 2003.

[DN00] I. Damgaard and J. B. Nielsen. improved non-committing encryption schemes based on general complexity assumption. CRYPTO 2000, pp. 432–450. 2000.

[DN02] I. Damgaard and J. B. Nielsen. Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor. CRYPTO 2002, pp. 581–596. 2002.

[D05] A. Datta, Security Analysis of Network Protocols: Compositional Reasoning and Complexity-theoretic Foundations. PhD Thesis, Computer Science Department, Stanford University, September 2005.

[DDMRS06] A. Datta, A. Derek, J. C. Mitchell, A. Ramanathan and A. Scedrov. Games and the Impossibility of Realizable Ideal Functionality. *3rd theory of Cryptology Conference (TCC)*, 2006.

[DKMR05] A. Datta, R. Küsters, J. C. Mitchell and A. Ramanathan. On the Relationships between Notions of Simulation-based Security. *2nd theory of Cryptology Conference (TCC)*, 2005.

[DOW92] W. Diffie, P. van Oorschot and M. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2, 1992, pp. 107–125.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory* IT-22, November 1976, pp. 644–654.

[DDOPS01] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano and A. Sahai. Robust Non-Interactive Zero-Knowledge. *CRYPTO 01,* 2001.

[DIO98] G. Di Crescenzo, Y. Ishai and R. Ostrovsky. Non-interactive and non-malleable commitment. *30th STOC,* 1998, pp. 141-150.

[DHR00] Y. Dodis, S. Halevi, T. Rabin. A Cryptographic Solution to a Game Theoretic Problem. *CRYPTO '00*, pp. 112–130. 2000.

[DM00] Y. Dodis and S. Micali. Secure Computation. *CRYPTO '00,* 2000.

[DDN00] D. Dolev. C. Dwork and M. Naor. Non-malleable cryptography. *SIAM. J. Computing,* Vol. 30, No. 2, 2000, pp. 391-437. Preliminary version in *23rd Symposium on Theory of Computing (STOC),* ACM, 1991.

[DY83] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory,* 2(29), 1983.

[DLMS04] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

[DNRS99] C. Dwork, M. Naor, O. Reingold, and L. Stockmeyer. Magic functions. *J. ACM* 50(6): 852-921 (2003). Preliminary version in *40th FOCS,* pages 523–534. IEEE, 1999.

[DNS98] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In *30th STOC*, pages 409–418, 1998.

[EG83] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *Proceedings, 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 34–39. IEEE, November 1983.

[EGL85] S. Even, O. Goldreich and A. Lempel, A randomized protocol for signing contracts, *CACM*, vol. 28, No. 6, 1985, pp. 637-647.

[FHG98] F. J. T. Fabrega, J. C. Herzog, J. D. Guttman. Strand Spaces: Why is a Security Protocol Correct? *IEEE Symposium on Security and Privacy,* May 1998.

[F91] U. Feige. Ph.D. thesis, Weizmann Institute of Science, 1991.

[FS90] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd STOC*, pages 416–426, 1990.

[FM97] P. Feldman and S. Micali, An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement, *SIAM Journal on Computing, Vol. 26, No. 4,* 1997, pp. 873–933.

[FF00] M. Fischlin and R. Fischlin, Efficient non-malleable commitment schemes, *CRYPTO '00, LNCS 1880*, 2000, pp. 413-428.

[F06] M. Fischlin. Round-Optimal Composable Blind Signatures in the Common Reference String Model. Crypto 2006, LNCS Volume 4117, pp. 60-77.

[GM00] J. Garay and P. MacKenzie, Concurrent Oblivious Transfer, *41st FOCS*, 2000.

[GMY04] J. Garay and P. MacKenzie and K. Yang. Efficient and Universally Composable Committed Oblivious Transfer and Applications. *Theory of Cryptography Conference (TCC),* LNCS 2951. 2004.

[GMY04a] J. A. Garay, P. MacKenzie and K. Yang. Efficient and Secure Multi-Party Computation with Faulty Majority and Complete Fairness. Eprint archive, eprint.iacr.org/2004/009.

[GL03] R. Gennaro, Y. Lindell. A Framework for Password-Based Authenticated Key Exchange. *ACM Trans. Inf. Syst. Secur.* 9(2): 181-234 (2006). Preliminary version in *EUROCRYPT 2003,* pp. 524–543. 2003.

[GLMMR04] R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali and T. Rabin. Tamper Proof Security: Theoretical Foundations for Security Against Hardware Tampering. *Theory of Cryptography Conference (TCC),* LNCS 2951. 2004.

[GM95] R. Gennaro and S. Micali. Verifiable Secret Sharing as Secure Computation. *Eurocrypt 95, LNCS 921,* 1995, pp. 168-182.

[GRR98] R. Gennaro, M. Rabin and T Rabin. Simplified VSS and Fast-track Multiparty Computations with Applications to Threshold Cryptography, *17th PODC,* 1998, pp. 101-112.

[G01] O. Goldreich. *Foundations of Cryptography.* Cambridge Press, Vol 1 (2001).

[G02] O. Goldreich. Concurrent Zero-Knowledge With Timing, Revisited. *34th STOC,* 2002.

[G04] O. Goldreich. *Foundations of Cryptography.* Cambridge Press, Vol 2 (2004).

[GK88] O. Goldreich and A. Kahan. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Jour. of Cryptology*, Vol. 9, No. 2, pp. 167–189, 1996.

[GK89] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM. J. Computing,* Vol. 25, No. 1, 1996.

[GMW87] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game. *19th Symposium on Theory of Computing (STOC),* ACM, 1987, pp. 218-229.

[GO94] O. Goldreich and Y. Oren. Definitions and properties of Zero-Knowledge proof systems. *Journal of Cryptology*, Vol. 7, No. 1, 1994, pp. 1–32. Preliminary version by Y. Oren in *28th Symp. on Foundations of Computer Science (FOCS)*, IEEE, 1987.

[GL90] S. Goldwasser, and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. *CRYPTO '90, LNCS 537*, 1990.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, Vol. 28, No 2, April 1984, pp. 270-299.

[GMRa89] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Comput.*, Vol. 18, No. 1, 1989, pp. 186-208.

[GMRi88] S. Goldwasser, S. Micali, and R.L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, April 1988, pages 281–308.

[GO92] S. Goldwasser and R. Ostrovsky, "Invariant Signatures and Non-Interactive Zero-Knowledge Proofs Are Equivalent," *Crypto '92*, 1992, pp. 228-245.

[G04] J. Groth. Re-randomizable and Replayable Adaptive Chosen Ciphertext Secure Cryptosystems. In proceedings of *TCC'04*, 2004.

[HKN04] S. Halevi, P. Karger and D. Naor. Enforcing Confinement in Distributed Storage and a Cryptographic Model for Access Control. Eprint archive, eprint.iacr.org/2005/169.

[HM00] M. Hirt and U. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation. *Journal of Cryptology*, Vol 13, No. 1, 2000, pp. 31-60. Preliminary version in *16th Symp. on Principles of Distributed Computing (PODC)*, ACM, 1997, pp. 25–34.

[H85] C. A. R. Hoare. Communicating Sequential Processes. International Series in Computer Science, Prentice Hall, 1985.

[HMS03] D. Hofheinz and J. Müler-Quade and R. Steinwandt. Initiator-Resilient Universally Composable Key Exchange. *ESORICS*, 2003. Extended version at the eprint archive, eprint.iacr.org/2003/063.

[HM04] D. Hofheinz and J. Müller-Quade. Universally Composable Commitments Using Random Oracles. *Theory of Cryptography Conference (TCC)*, LNCS 2951 pp. 58-74. 2004.

[HM04a] D. Hofheinz and J. Müller-Quade. A Synchronous Model for Multi-Party Computation and the Incompleteness of Oblivious Transfer. Eprint archive, http:/eprint.iacr.org/2004/016, 2004.

[HMS03a] D. Hofheinz, J. Müller-Quade and R. Steinwandt. On Modeling IND-CCA Security in Cryptographic Protocols. *4th Central European Conference on Cryptology, WARTACRYPT'04*, pp.47–49. Full version at http://eprint.iacr.org/2003/024.

[HMU05] D. Hofheinz, J. Müller-Quade and D. Unruh. Polynomial Runtime in Simulatability Definitions. Manuscript, 2005.

[HU05] D. Hofheinz and D. Unruh. Comparing Two Notions of Simulatability. *2nd theory of Cryptology Conference (TCC)*, pp. 86-103, 2005.

[IK03] R. Impagliazzo and B. M. Kapron. Logics for Reasoning about Cryptographic Constructions. *44th FOCS*, pp. 372-383. 2003.

[IR89] R. Impagliazzo, S. Rudich. Limits on the Provable Consequences of One-Way Permutations. *22nd STOC*, pp. 44-61. 1989.

[IPSEC] The IPSec working group of the IETF. See http://www.ietf.org/html.charters/ipsec-charter.html

[ILM05] S. Izmalkov, M. Lepinski and S. Micali. Universal Mechanism Design. *46th FOCS*, 2005.

[K07] J. Katz. Universally Composable Multi-party Computation Using Tamper-Proof Hardware. EUROCRYPT 2007, pp. 115-128.

[KMM94] R. Kemmerer, C. Meadows and J. Millen. Three systems for cryptographic protocol analysis. *J. Cryptology*, 7(2):79-130, 1994.

[KZ04] A. Kiayias and H. Zhou. Equivocal Blind Signatures and Adaptive UC-Security. *5th Theory of Cryptology Conference (TCC)*, 2004.

[K89] J. Kilian Uses of Randomness in Algorithms and Protocols, Chapter 3, The ACM Distinghished Dissertation 1989, MIT press.

[K96] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. *CRYPTO '96*, pp. 104–113. 1996.

[K01] H. Krawczyk. The order of encryption and authentication for protecting communications (Or: how secure is SSL?). *CRYPTO 01*, 2001.

[K06] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. CSFW 2006, pp. 309-320.

[LMPS04] M. Lepinski, S. Micali, C. Peikert, A. Shelat. Completely fair SFE and coalition-safe cheap talk. *23rd PODC*, pp. 1-10. 2004.

[L03] Y. Lindell. Bounded-concurrent secure two-party computation without setup assumptions. *35th STOC*, pp. 683–692, 2003.

[L03a] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. *43rd FOCS*, pp. 394–403. 2003.

[L04] Y. Lindell. Lower Bounds for Concurrent Self Composition. *1st Theory of Cryptology Conference (TCC)*, pp. 203–222. 2004.

[LLR02] Y. Lindell, A. Lysyanskaya and T. Rabin. On the composition of authenticated Byzantine agreement. *34th STOC*, 2002.

[LPT04] Y. Lindell, M. Prabhakaran, Y. Tauman. Concurrent General Composition of Secure Protocols in the Timing Model. Manuscript, 2004.

[LMMS98] P. Lincoln, J. Mitchell, M. Mitchell, A. Scedrov. A Probabilistic Poly-time Framework for Protocol Analysis. *5th ACM Conf. on Computer and Communication Security*, 1998, pp. 112-121.

[LMMS99] P. Lincoln, J. Mitchell, M. Mitchell, A. Scedrov. Probabilistic Polynomial-time equivalence and security analysis. *Formal Methods Workshop,* 1999. Available at ftp://theory.stanford.edu/pub/jcm/papers/fm-99.ps.

[LO96] G. Lowe. Breaking and fixing the Needham-Schröder public-key protocol using CSP and FDR. *2nd International Workshop on Tools and Algorithms for the construction and analysis of systems,* 1996.

[LY96] N. Lynch. *Distributed Algorithms.* Morgan Kaufman, San Francisco, 1996.

[LSV03] N. Lynch, R. Segala and F. Vaandrager. Compositionality for Probabilistic Automata. *14th CONCUR,* LNCS vol. 2761, pages 208-221, 2003. Fuller version appears in MIT Technical Report MIT-LCS-TR-907.

[MMS03] P. Mateus, J. C. Mitchell and A. Scedrov. Composition of Cryptographic Protocols in a Probabilistic Polynomial-Time Process Calculus. *CONCUR,* pp. 323-345. 2003.

[M94] C. Meadows. A model of computation for the NRL protocol analyzer. *Computer Security Foundations Workshop (CSFW),* IEEE Computer Security Press, 1994.

[M94a] C. Meadows. Formal verification of cryptographic protocols: A survey. *Asiacrypt '94,* LNCS 917, 1995, pp. 133-150.

[MP04] C. Meadows, Dusko Pavlovic. Deriving, Attacking and Defending the GDOI Protocol. *ESORICS 2004,* pp. 53-72 2004.

[MPR06] S. Micali, R. Pass, A. Rosen. Input-Indistinguishable Computation. FOCS 2006, pp. 367-378.

[MRV99] S. Micali, M. Rabin, and S. Vadhan. Verifiable Random Functions. *40th Annual Symposium on Foundations of Computer Science,* 1999.

[MR04] S. Micali and L. Reyzin. Physically Observable Cryptography. *1st Theory of Cryptography Conference (TCC)*, LNCS 2951, 2004.

[MR91] S. Micali and P. Rogaway. Secure Computation. unpublished manuscript, 1992. Preliminary version in *CRYPTO '91, LNCS 576,* 1991.

[MW04] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In the *1st Theory of Cryptography Conference (TCC)*, LNCS 2951, pp. 133–151. 2004.

[M89] R. Milner. Communication and Concurrency. Prentice Hall, 1989.

[M99] R. Milner. Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press, 1999.

[MMS98] J. Mitchell, M. Mitchell, A. Schedrov. A Linguistic Characterization of Bounded Oracle Computation and Probabilistic Polynomial Time. *39th FOCS*, 1998, pp. 725-734.

[NMO05] W. Nagao, Y. Manabe and T. Okamoto. A Universally Composable Secure Channel Based on the KEM-DEM Framework. *2nd theory of Cryptology Conference (TCC),* 2005.

[NY90] M. Naor and M. Yung. Public key cryptosystems provably secure against chosen ciphertext attacks". *22nd STOC*, 427-437, 1990.

[NS78] R. Needham and M. Schröder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[N02] J. B. Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. *CRYPTO*, pp. 111–126. 2002.

[N03] J. B. Nielsen. On Protocol Security in the Cryptographic Model. PhD thesis, Aarhus University, 2003.

[P04] R. Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. *36th STOC*, pp. 232–241. 2004.

[P06] R. Pass. A Precise Computational Approach to Knowledge. PhD Thesis, MIT. 2006.

[PR03] R. Pass, A. Rosen. Bounded-Concurrent Secure Two-Party Computation in a Constant Number of Rounds. *44th FOCS*, 2003

[PR05a] R. Pass, A. Rosen. New and improved constructions of non-malleable cryptographic protocols. *STOC*, pp. 533-542, 2005.

[PR05b] R. Pass, A. Rosen. Concurrent and Non-Malleable Commitments. *FOCS*, 2005.

[P05] A. Patil. On symbolic analysis of cryptographic protocols. Master's thesis, Massachusetts Institute of Technology, May 2005.

[P88] L. C. Paulson. Isabelle: the next seven hundred theorem provers (system abstract). 9th International Conf. on Automated Deduction, LNCS 310, pp. 772773, 1988. More details at http://www.cl.cam.ac.uk/Research/HVG/Isabelle/

[PW94] B. Pfitzmann and M. Waidner. A general framework for formal notions of secure systems. Hildesheimer Informatik-Berichte 11/94, Universitat Hildesheim, 1994. Available at http://www.semper.org/sirene/lit.

[PSW00] B. Pfitzmann, M. Schunter and M. Waidner. Secure Reactive Systems. IBM Research Report RZ 3206 (#93252), IBM Research, Zurich, May 2000.

[PSW00a] B. Pfitzmann, M. Schunter and M. Waidner. Provably Secure Certified Mail. IBM Research Report RZ 3207 (#93253), IBM Research, Zurich, August 2000.

[PW00] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. *7th ACM Conf. on Computer and Communication Security*, 2000, pp. 245-254.

[PW01] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. IEEE Symposium on Security and Privacy, May 2001. Preliminary version in http://eprint.iacr.org/2000/066 and IBM Research Report RZ 3304 (#93350), IBM Research, Zurich, December 2000.

[PS04] M. Prabhakaran, A. Sahai. New notions of security: achieving universal composability without trusted setup. *36th STOC*, pp. 242–251. 2004.

[PS05] M. Prabhakaran, A. Sahai. Relaxing Environmental Security: Monitored Functionalities and Client-Server Computation. *2nd theory of Cryptology Conference (TCC),* 2005.

[R81] M. Rabin. How to exchange secrets by oblivious transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[RB89] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multi-party Protocols with Honest Majority. *21st Symposium on Theory of Computing (STOC),* ACM, 1989, pp. 73-85.

[RS91] C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. *CRYPTO '91,* 1991.

[RK99] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *Eurocrypt99*, LNCS 1592, pages 415–413.

[RK05] R. Renner and R. König. Universally Composable Privacy Amplification Against Quantum Adversaries. *2nd theory of Cryptology Conference (TCC),* 2005.

[R06] Alon Rosen. *Concurrent Zero-Knowledge.* Series on Information Security and Cryptography. Springer-Verlag, 2006.

[R+00] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe and B. Roscoe. *The Modeling and Analysis of Security Protocols: the CSP Approach.* Addison-Wesley, 2000.

[SL95] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing,* Vol. 2. No. 2, pp 250-273, 1995.

[Sh99] V. Shoup. On Formal Models for Secure Key Exchange. manuscript, 1999. Available at: http://www.shoup.org.

[Si05] M. Sipser. *Introduction to the Theory of Computation.* Second edition, Course Technology, 2005.

[So99] D. Song. Athena: an Automatic Checker for Security Protocol Analysis. *Proc. of 12th IEEE Computer Security Foundation Workshop,* June 1999.

[S+06] C. Sprenger, M. Backes, D. A. Basin, B. Pfitzmann, and M. Waidner. Cryptographically sound theorem proving. In *CSFW*, pages 153–166. IEEE Computer Society, July 2006.

[W04] D. Wikström. A Universally Composable Mix-Net. *1st Theory of Cryptography Conference (TCC)*, LNCS 2951. 2004.

[W04a] D. Wikström. Universally Composable DKG with Linear Number of Exponentiations. Eprint archive eprint.iacr.org/2004/124.

[Y82] A. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd Annual Symp. on Foundations of Computer Science (FOCS),* pages 80–91. IEEE, 1982.

[Y82A] A. Yao. Protocols for Secure Computation. In *Proc. 23rd Annual Symp. on Foundations of Computer Science (FOCS),* pages 160–164. IEEE, 1982.

[Y86] A. Yao, How to generate and exchange secrets, In *Proc. 27th Annual Symp. on Foundations of Computer Science (FOCS),* pages 162–167. IEEE, 1986.

A Related work

This section surveys some works that are closely related to this one. Some of the reviewed works are prior art. Others were done concurrently to the first version of this work or subsequently to it. For clarity, the presentation is organized primarily by topics, rather than by work or by chronological order. Works that are relevant to more than one topic may be mentioned in multiple places. Section A.1 discusses other general frameworks for defining cryptographic security of protocols. Works on formulating definitions of security within this framework and designing protocols that realize them, as well as the impossibility thereof, are reviewed in Section A.2. Section A.3 reviews works that extend and relax the framework. Finally, Section A.4 discusses works that use the framework to perform symbolic and automated security analysis of protocols.

A.1 Other simulation-based security frameworks

Two works that essentially "laid out the field" of general security definitions for cryptographic protocols are the work of Yao [Y82], which stated for the first time the need for a general "unified" framework for expressing the security requirements of cryptographic tasks and for analyzing cryptographic protocols; and the work of Goldreich, Micali and Wigderson [GMW87], which put forth the "trusted-party paradigm", namely the approach of defining security via comparison with an ideal process involving a trusted party (albeit in a very informal way).

Another work that greatly influenced the formation of notions of security is the work of Dolev, Dwork and Naor [DDN00]. This work points out some important security concerns that arise when cryptographic protocols run concurrently within a larger system. In particular, making sure that the concerns pointed out in [DDN00] are addressed is central to the present framework.

The first rigorous definitional framework is due to Goldwasser and Levin [GL90], and was followed shortly by the frameworks of Micali and Rogaway [MR91] and Beaver [B91]. In particular, the notion of "reducibility" in [MR91] directly underlies the notion of protocol composition in many subsequent works including the present one. Beaver's framework was the first to directly formalize the idea of comparing a run of a protocol to an ideal process. (However, the [MR91, B91] formalisms only address security in restricted settings; in particular, they do not deal with computational issues.) [GL90, MR91, B91] are surveyed in [C00] in more detail.

The frameworks of [GL90, MR91, B91] concentrate on *synchronous* communication. Also, although in [GMW87] the trusted-party paradigm was put forth for reactive functionalities, the three frameworks concentrate on the task of *secure function evaluation.* An extension to *synchronous* communication networks is formulated in [BCG93]. A system model and notion of security for reactive functionalities is sketched in Pfitzmann and Waidner [PW94].

Canetti [C95] provides the first ideal-process based definition of computational security against resource bounded adversaries. [C00] strengthens the framework of [C95] to handle secure composition. In particular, [C00] defines a general composition operation, called *modular composition,* which is a non-concurrent version of universal composition. That is, only a single protocol instance can be active at each point in time. (See more details in Section 6.6.) In addition, security of protocols in that framework is shown to be preserved under modular composition. This work also contains sketches on how to strengthen the definition to support concurrent composition. The UC framework implements these sketches in a direct way. A closely related formulation appears in [G04].

The framework of Hirt and Maurer [HM00] provides a rigorous treatment of reactive functionalities. Dodis and Micali [DM00] build on the definition of Micali and Rogaway [MR91] for unconditionally secure function evaluation, where ideally private communication channels are assumed. In that setting, they prove that their notion of security is preserved under a general concurrent composition operation similar to universal composition. They also formulate an additional composition operation (called *synchronous composition*) that provides stronger security guarantees, and show that their definition is closed under that composition operation in cases where the scheduling of the various instances of the protocols can be controlled. However, their definition applies only to settings where the communication is ideally private. It is not clear how to extend this definitional approach to settings where the adversary has access to the communication between honest parties.

The framework of Pfitzmann, Schunter and Waidner [PSW00, PW00] is the first to rigorously address *concurrent* universal composition in a computational setting. (This work is based on the sketches in [PW94]). They define security for reactive functionalities in a synchronous setting and prove that security is preserved when a *single* instance of a subroutine protocol is composed concurrently with the calling protocol. An extension of the [PSW00, PW00] framework to asynchronous networks appears in [PW01].

At high level, the notion of security in [PSW00, PW00, PW01], called reactive simulatability, is similar to the one here. In particular, the role of their "honest user" can be roughly mapped to the role of the environment as defined here. However, there are several differences. They use a finite-state machine model of computation that builds on the I/O automata model of [Ly96], as opposed to the ITM-based model used in this work. On the one hand, their model provides a richer set of methods for scheduling events in an execution. On the other hand, they postulate a closed system where the number of participants is constant and fixed in advance. Furthermore, the number of *protocol instances* run by the parties is constant and fixed in advance, thus it is impossible to argue about the security of systems where the number of protocol instances may be a non-constant function of the security parameter (even if this number is known in advance). That model also automatically provides each pair of parties with a dedicated and labeled communication channel; thus it does not facilitate arguing about situations where dedicated channels are not available a-priori. Other technical differences include the notion of polynomial time computation and the generation of outputs.

Backes, Pfitzmann and Waidner [BPW04] extend the framework of [PW01] to deal with the case where the number of parties and protocol instances depends on the security parameter. In that framework, they prove that reactive simulatability is preserved under universal composition. The [BPW04a] formulation returns to the original approach where the number of entities and protocol instances is fixed irrespective of the security parameter. Nielsen [N03] and Hofheinz and Müller-Quade [HM04a] formulate synchronous variants of the UC framework.

Lincoln, Mitchell, Mitchell and Scedrov [LMMS98, LMMS99] develop a process calculus, based on the π-calculus of Milner [M89, M99], that incorporates random choices and computational limitations on adversaries. (In [MMS98] it is demonstrated how to express probabilistic polynomial time within such a process calculus.) In that setting, their definitional approach has a number of similarities to the simulation-based approach taken here: They define a computational variant of *observational equivalence*, and say that a real-life process is secure if it is observationally equivalent to an "ideal process" where the desired functionality is guaranteed. This is indeed similar to requiring that no environment can tell whether it is interacting with the ideal process or with the protocol execution. However, their ideal process must vary with the protocol to be analyzed, and they do not seem to have an equivalent of the notion of an "ideal functionality" which is associated only with the task and is independent of the analyzed protocol. This makes it harder to formalize the security requirements of a given task.

Mateus, Mitchell and Scedrov [MMS03] and Datta, Küsters, Mitchell, and Ranamanathan [DKMR05] (see also [D05]) extend the [LMMS98, LMMS99] framework to express simulatability as defined here, cast in a polytime probabilistic process calculus, and demonstrate that the universal composition theorem holds in their framework. They also rigorously compare certain aspects of the present framework (as defined in [C01]) and reactive simulatability (as defined in [BPW04a]). Tight correspondence between the [MMS03] notion of security and the one defined here is demonstrated in Almansa [A04].

Canetti et al. [C+05] extend the probabilistic I/O automata of Lynch, Segala and Vaandrager [SL95, LSV03] to a framework that allows formulating security of cryptographic protocols along the lines of the present UC framework. This involves developing a special mechanism, called the *task schedule,* for curbing the power of non-deterministic scheduling; it also requires modeling resource-bounded computations. The result is a framework that represents the concurrent nature of distributed systems in a direct way, that allows for analyzing partially-specified protocols (such as, say, standards), that allows some scheduling choices to be determined non-deterministically during run-time, and at the same time still allows for meaningful UC-style security specifications.

Küsters [K06] formulates an ITM-based model of computation that allows for defining UC-style notions of security. The model contains new constructs that facilitate both flexible addressing of messages and a flexible notion of resource-bounded computation in a distributed environment. This work also adapts abstract notations from the process calculus literature to his ITM-based model, allowing for succinct and clear presentation of composition theorems and proofs. The present version of this work uses ideas from [K06].

A.2 On formulating tasks and analyzing protocols within the UC framework

This section briefly reviews the body of work carried out on modeling and analyzing protocols within the UC framework. This includes works that formulate the security requirements of tasks and the trust assumptions on the underlying system as ideal functionalities within the UC framework, that present protocols that realize such functionalities, and that demonstrate the *impossibility* of realizing functionalities in certain cases. For clarity, this section is sub-divided by topics.

General multi-party computation with honest majority. Some known constructions for general secure function evaluation turn out to be universally composable. In particular, the [BGW88] constructions, as well as the simplification of [GRR98]), result in protocols that UC-realize practically any ideal functionality, as long as less than a third of the parties are corrupted, assuming synchronous, authenticated and secure communication channels. (Such channels can be captured via a variant of $\mathcal{F}_{\text{SYN}}$ that provides message secrecy along the lines of $\mathcal{F}_{\text{SMT}}$.) Using [RB89], any corrupted minority is tolerable. Composing these protocols with protocols for realizing $\mathcal{F}_{\text{SMT}}$, one obtains protocols that use open synchronous communication. Asynchronous communication can be handled using the techniques of [BCG93, BKR94]. These facts are stated in early versions of this work [C01] with a sketch of proof. We leave full proof out of scope for this work. It is also demonstrated there how to deal with the case where any number of protocol participants is corrupted; This is done by using a set of "helper parties", or "servers," where it is assumed that only a minority of the *servers* are corrupted.

Public-Key Encryption and Digital Signatures. An ideal functionality for capturing the security properties of digital signatures, $\mathcal{F}_{\text{SIG}}$, is proposed in the first version of this work [C01]. UC-realizing $\mathcal{F}_{\text{SIG}}$ is claimed to be equivalent to existential unforgeability against chosen message attacks as in [GMRi88]. The original formulation turned out to have several flaws, that were discovered and corrected in several iterations, in [CK02, CR03, BH04, C04]. The formulation in [C04] keeps to the approach of the original formulation and the equivalence with the [GMRi88] notion. A simpler, albeit somewhat less general formulation of $\mathcal{F}_{\text{SIG}}$ is presented in [C05].

An ideal functionality, $\mathcal{F}_{\text{PKE}}$, aimed at capturing the security properties of public-key encryption, is formulated in [C01]. It is erroneously claimed there that realizing $\mathcal{F}_{\text{PKE}}$ is a strictly weaker property than security against chosen ciphertext attacks (CCA security) as in [RS91, DDN00]. In [CKN03] it is shown that realizing $\mathcal{F}_{\text{PKE}}$ is in fact *equivalent* to CCA security, in the case where the party corruptions are *static.* (This fact was observed independently in [HMS03a].) They also formulate a relaxed variant of $\mathcal{F}_{\text{PKE}}$, called $\mathcal{F}_{\text{RPKE}}$ (for replayable public-key encryption), and demonstrate that this variant is sufficient for most applications of CCA-secure encryption.

Nielsen [N02] demonstrates that realizing $\mathcal{F}_{\text{PKE}}$ in the presence of adaptive party corruptions is *impossible.* This holds even in the case where parties are assumed to be able to effectively erase past data. [CHK05] proposes a relaxation of $\mathcal{F}_{\text{PKE}}$, where even the legitimate receiver is able to decrypt ciphertexts only during a given time window, and show how to realize this relaxed functionality under certain number-theoretic assumptions. A simplified formulation, as well as additional discussion appears in [C05].

Authenticated and Secure Communication; Key Exchange; UC with Joint State. In [C04] it is shown how to realize authenticated communication (namely, how to UC-realize $\mathcal{F}_{\text{AUTH}}$) via protocols that use $\mathcal{F}_{\text{SIG}}$ plus an ideal "authenticated bulletin board service" where parties can register public values together with their identities. This service is formulated via an ideal functionality, $\mathcal{F}_{\text{REG}}$, that all protocol participants are assumed to have access to. (Essentially, $\mathcal{F}_{\text{REG}}$ is aimed at representing the basic functionality provided by a certification authority.) The [C04] construction is essentially the simple "signature-based authenticator" constructions from [BCK98], and is focused on authenticating a single message using a single instance of $\mathcal{F}_{\text{SIG}}$.

Canetti and Krawczyk [CK02] define UC key exchange and secure sessions, and demonstrate connections with their earlier notion [CK01] which is not based on emulating an ideal process. They also prove security of some basic key exchange protocols and some natural ways to obtain secure communication sessions given an ideal key exchange protocol. That work also introduces a general technique, called *non-information oracles,* for relaxing the security requirements of ideal functionalities in the presence of adaptive corruptions. In [CK02a], it is shown that the "cryptographic core" of the key exchange protocol of the IPSEC security standard [IPSEC] is a UC key exchange protocol. Hofheinz et al. [HMS03] point out some flaws in the [CK02] formulation of the ideal key exchange functionality, and demonstrate how these flaws can be fixed so that the security analyses of the protocols in [CK02] remain valid. The key exchange functionality in [C05] takes these points into account.

Nagao, Manabe and Okamoto [NMO05] show how to realize UC secure channels using a variant of the "hybrid encryption" methodology for public-key encryption. This is an interesting alternative to the traditional way of obtaining secure communication sessions via key-exchange.

UC with Joint State. The treatment of authentication and key exchange protocols brings up the following issue. Many such protocols allow parties to have a single "long term authentication" module that is used to authenticate multiple messages in multiple sessions. This module typically involves a registration mechanism with a third party (e.g. a certification authority or a dedicated authentication server), plus an algorithmic part (e.g. a signature, public-key encryption, or message authentication scheme). This structure provides greater efficiency, since it allows a party to register only once, instead of once per message or once per authentication session. Also, a single instance of a signature or public-key encryption scheme per party may suffice for multiple authentication sessions.

From an analytical point of view, this structure means that all the authentication and key exchange sessions that use the same long-term module have "joint state" and therefore have to be analyzed jointly, as one piece. Indeed, authentication mechanisms are traditionally analyzed in this way, which results in relatively complex analysis.

The UC framework provides a way to simplify the analysis by considering only a single authentication session in isolation. Security of the composite system is then deduced using a general composition theorem. Note that, in of itself, the UC theorem is of no help here, since the UC operation allows composing only protocol instances that do not have any joint modules (subroutines). Still, the UC with Joint State (JUC) theorem [CR03] demonstrates that the UC theorem can be applied even to protocols that use a joint subroutine, as long as the joint subroutine satisfies a certain criterion. Both the signature-based and encryption-based long-term authentication modules satisfy this criterion [CR03, CH04].

It should be noted however that, even when using the JUC mechanism, the process of registering keys with a trusted third party is treated as a subroutine of the authentication protocol. Consequently, the analysis does not address security concerns that arise when the registering entity provides information to protocols other than the analyzed protocol. See more discussion on this issue at the end of this sub-section.

Two-party functionalities and general multi-party computation with honest minority. Ideal functionalities that capture a a number of salient two-party tasks have been formulated. Specifically, an ideal functionality $\mathcal{F}_{\text{COM}}$, that captures ideal commitment, an ideal functionality, $\mathcal{F}_{\text{ZK}}$, that captures ideal "Zero-Knowledge proof of knowledge", and an ideal functionality that captures two-party coin tossing are formulated in [CF01]. Ideal functionalities that capture Oblivious Transfer and multi-party generalizations of the above two-party functionalities are formulated in [CLOS02].

In the case of more basic notions of security of protocols, such as the notions of [C00] and [G04], practically any ideal functionality can be securely realized, as long as the parties are provided with authenticated communication and a broadcast channel. Furthermore, this holds for any number of corrupted parties, and even for two-party functionalities.

In the UC framework the situation is different: None of the known two-party protocols for any of the above tasks is UC secure. In fact, many interesting tasks, including all of those mentioned above, turn out to be unrealizable if honest majority is not guaranteed, even if authenticated communication is provided. Specifically, it is shown in [CF01] that a natural and basic formalization of $\mathcal{F}_{\text{COM}}$ cannot be realized in the UC framework, as long as either one of the parties can be corrupted, even if ideally authenticated communication is provided. Similar impossibility results are proven in [C01] for the ideal coin tossing functionality, the ideal Zero-Knowledge functionality, and the ideal Oblivious Transfer functionality. These results are extended in Canetti, Kushilevitz and Lindell [CKL03] to demonstrate unrealizability by plain protocols of almost all "non-trivial" deterministic two-party functions and many probabilistic two-party functions, and in Datta et al. [DDMRS06] to demonstrate impossibility of realizing *any* "ideal commitment functionality", namely any functionality that guarantees the basic correctness, binding and secrecy properties of commitment in a perfect way.

These sweeping impossibility results motivated two research directions. One direction is to look for more relaxed notions of security, that might continue to provide composable security but still be realizable in the plain model. Works towards this direction are described in Section A.3. The other direction is to keep requiring UC security, but construct protocols that make use of trusted set-up. In other words, design and analyze protocols under the assumption that some components of the system behave in a trusted way. As usual, these assumptions are modeled by letting the protocols use ideal functionalities that represent the trust assumptions.

The first trusted set-up assumption that was used in this context is the assumption that the protocol participants have access to a common string that is guaranteed to be taken from a pre-defined distribution. This is modeled via an ideal functionality that chooses a reference string from the specified distribution and makes the string available to all protocol participants. Indeed, this trusted set-up assumption turns out to be essentially equivalent to the well-known *common random string (CRS) model* of [BFM89]. To highlight this fact, we call the coin-tossing functionality $\mathcal{F}_{\text{CRS}}$. The protocol of [CF01] is based on general assumptions (specifically, existence of claw-free trapdoor permutations) and is non-interactive, but is inefficient in that the size of the commitment string is roughly the security parameter times the length of the committed string. The protocol of [CLOS02] uses only standard trapdoor permutations. These schemes can be made secure against adaptive adversaries, assuming existence of enhanced trapdoor permutations, and can be made to use a uniformly distributed reference string assuming dense cryptosystems. The protocol of Damgaard and Nielsen [DN00] is interactive, uses reference string whose length is linear in the number of parties, but the size of the commitment is roughly the same as the size of the committed string. It is based on the Paillier assumption. Damgaard and Groth show how to make do with a reference string that is independent of the number of parties, while retaining the other efficiency parameters. Hofheinz et. al. construct a surprisingly simple UC commitment protocol in the random oracle model [HM04].

In [CLOS02] these results are extended to realizing practically any ideal functionality, including reactive functionalities and multi-party functionalities, via $\mathcal{F}_{\text{CRS}}$-hybrid protocols. These results hold even with respect to adaptive corruption of parties and even when data cannot be effectively erased. We remark that the [CLOS02] construction can be slightly modified to do without the common random string and remain secure against adversaries that corrupt only a minority of the parties.

Damgaard and Groth [DG03] also show how to construct, in a black-box way, a key exchange protocol from a UC commitment protocol. Using the separation results of Impagliazzo and Rudich [IR89], we conclude that it is unlikely that UC commitment can be constructed based only on one-way functions in a black-box way.

In [B$^+$05], the results of [CLOS02] are generalized to a setting where the adversary controls the network and *no authenticated set-up is available.* Somewhat surprisingly it is shown that, even in this completely unauthenticated setting, if the parties have access to a common reference string then essentially the only possible adversarial behavior is to "partition" the network to several disjoint "clusters", where each "cluster" realizes the original functionality in the usual authenticated and secure manner. (If only *bounded concurrency* is desired, e.g. if a bound on number of concurrent sessions is known a priori, then the same effect can be obtained with no setup whatsoever.)

Other primitives. Garay, Mackenzie and Yang construct UC protocols for Committed Oblivious Transfer [GMY04a]. Interactive Zero-Knowledge protocols for any language in NP are constructed in [CF01], and non-interactive ones are constructed in [CLOS02], both under general assumptions and in the CRS model. The interactive ZK protocols are secure even in the presence of adaptive party corruptions without data erasures. In contrast, the non-interactive ZK protocols work only in the presence of static party corruptions, or alternatively make essential use of data erasures.

One-to-many variants of UC commitment and Zero-Knowledge are defined and realized in [CLOS02]. Here a single committer (resp. prover) commits to a value (resp., proves a statement) to a set of recipients. The secrecy (resp., Zero-Knowledge) guarantee remains the same, and the binding (resp., soundness) guarantee now applies to all the recipients. These primitives are at the heart of the general feasibility result in [CLOS02] for multi-party functionalities. A general construction of one-to-many Zero-Knowledge from standard Zero-Knowledge is described in [P04].

An ideal functionality that captures password-based key exchange is formulated in [CHKLM05]. It is then shown to be UC-realized by a variant of the [GL03] protocol (in the CRS model). We remark that formulating an ideal functionality that adequately captures password-based key exchange turns out to be non-trivial; in fact, the final formulation is quite different than the first formulation that comes to mind.

Prabhakaran and Sahai [PS05] provide, using an approach related to the "non-information oracles" of [CK02], relaxed formulations of some two-party functionalities (including commitment and ZK). Their relaxed functionalities are realizable by more efficient protocols.

A UC definition of a mix-net for the purpose of anonymous voting protocols is formulated and realized in Wikström [W04]. His protocol uses ideal calls to the Zero-Knowledge ideal functionality, $\mathcal{F}_{\text{ZK}}$, but is otherwise quite efficient. A UC notion of anonymous routing was defined and realized in Camenisch and Lysyanskaya [CL05].

UC definitions of threshold key generation for the purpose of threshold signature and encryption schemes are formulated and realized in Abe and Fehr [AF04] and independently in Wikström [W04a]. UC formulations of blind signatures are formulated an realized in [F06, KZ04]

Other trusted set-up models. A number of alternative trusted set-up assumptions have been considered. We give here a brief overview. See [C07] for a more detailed survey. Canetti et. al. [CPS07] consider the case where the reference string is taken from a distribution that is not known or is adversarially controlled, and investigate the minimal guarantees on this distribution that suffice for realizing functionalities of interest. They show how to realize $\mathcal{F}_{\text{COM}}$ as long as the reference string is taken from a distribution that's efficiently samplable and in addition satisfies some weak criteria relating the min-entropy and the description length of (or, amount of non-uniformity in) the sampling algorithm. They also show that these requirements from the reference string are minimal, in the sense that if they are relaxed in any way then realizing $\mathcal{F}_{\text{COM}}$ becomes impossible.

An alternative relaxation of the CRS model is considered in Barak et. al. [BCNP04]. They demonstrate how to realize $\mathcal{F}_{\text{COM}}$ (and hence practically any ideal functionality, using, say, [CLOS02]) when F_{CRS} is replaced with an ideal functionality, $\mathcal{F}_{\text{KRK}}$, that allows parties to register public keys in a way that lets the corrupted parties (and only them) know their secret keys. That is, when an uncorrupted part registers, a public key is provided by $\mathcal{F}_{\text{KRK}}$ and the corresponding secret key remains unknown even to the registering party. When a corrupted party registers, it is expected to explicitly provide a secret key from which the public key is computed. Then, any protocol participant can obtain from F_{KRK} the public key of any other participant. This formulation is reminiscent of the CRS model, in the sense that protocols designed in it cannot assume knowledge of secret keys associated with the public information, yet the adversary interacting with the ideal functionality is assumed to know the secret keys associated with corrupted parties. Still, this formulation allows realizations require weaker trust assumptions. For instance, it is realizable in a setting where there are multiple registration servers and neither server is trusted by all. Here each protocol participant needs to put full trust only in the sever with which it registers. Other servers need not be trusted to keep information secret; they only need to be trusted to compute public keys correctly. In [CDPW07] the protocol of [BCNP04] is extended to handle adaptive corruptions.

Katz [K07] proposes a trusted set-up assumption of a different flavor. The idea is to assume that there exist computing devices that can be programmed by one party and then handed over to another party to be executed in a "tamper-proof" way, and without a direct communication link with external entities. No other trust assumptions are made; in particular, the protocol participants need not obtain any trusted information from external entities. Formally, this trust assumption is captured via an ideal functionality, $\mathcal{F}_{\text{WRAP}}$, that allows one party to specify code that is then executed by another party. $\mathcal{F}_{\text{WRAP}}$ does not provide the executing party, B, with any information about the code M that it is running, nor can B influence that code. Still, M can maintain local state between calls and make private random choices. These features represent the "tamper-proofness", or "virtual black box" guarantees that are given to party A. On the other hand, $\mathcal{F}_{\text{WRAP}}$ does not provide any information to party A. This guarantees to party B that M communicates only with B and has no "side channels" for communicating with A or other parties. It is stressed again that $\mathcal{F}_{\text{WRAP}}$ can be realized without requiring the parties to trust any third parties. Still, Katz shows that it suffices for realizing $\mathcal{F}_{\text{COM}}$, under standard number-theoretic assumptions.

All the trusted set-up assumptions described so far model the trusted information as a subroutine of the analyzed protocol. This means that the information is made available only to the participants of a single protocol. Consequently, in order to implement such a model, one has to generate a fresh instance of the trusted information (i.e., a fresh reference string, a fresh set of public keys, or a fresh hardware device) for each instance of a protocol that uses it. Furthermore, this has to be done in a way that makes the trusted information not directly available to the participants of other protocol instances. While such implementations are of course possible, this is a severe limitation. In particular, this modeling stands in contrast with the prevalent intuitive perception of the reference string and public key infrastructure as "global" constructs that may be chosen in advance and made available to all throughout the lifetime of the system.

Trusted set-up assumptions that involve globally available information are considered in [CDPW07]. The idea is to model the trusted information as an entity (ITI) $\mathcal{F}$ that provides the information to all other entities (ITIs) in the system. This means that $\mathcal{F}$ exists in the system regardless of the analyzed protocol; in particular, it exists even when the analyzed protocol is replaced by another protocol via universal composition. (This in turn requires an extension to the notion of protocol emulation and the the UC theorem; see more details in Section A.3.)

The natural extension of $\mathcal{F}_{\text{CRS}}$ to an ideal functionality that gives the reference string all entities in the system turns out to be provably insufficient for realizing $\mathcal{F}_{\text{COM}}$ [CDPW07]. In fact, *no* set-up assumption that involves only information that's globally available suffices. In contrast, a variant of the key registration model where the public key is made available to all entities in the system suffices for realizing $\mathcal{F}_{\text{COM}}$. In fact, the same protocols (by [BCNP04] for static corruptions and [CDPW07] for adaptive corruptions) work. Furthermore, an extension of the [CDPW07] protocol works even in a variant of the CRS model, called *augmented CRS (ACRS),* where there is a short, globally available reference string, plus secret information that is made available to each corrupted party.

Similar issues arise when trying to directly model the common situation where the public keys registered by parties for the purpose of *authentication* are disclosed by the registering party (the "certificate authority") to third parties that do not take part in the authentication protocol. Here the traditional modeling of the registering process as a subroutine of the authentication protocol is inadequate. Modeling authentication protocols in this setting is a fascinating research topic.

Another approach for putting trust in the physical behavior of some parts of the system is to make assumptions on the relative delays of messages in the network. More precisely, assume that all messages sent are eventually delivered unmodified within some time bound, and in addition there is a bound on the amount of relative "drift" between local clocks of parties in the system. In Lindell et. al. [LPT04] it is shown how to UC-realize $\mathcal{F}_{\text{CRS}}$ (hence also $\mathcal{F}_{\text{COM}}$) in such a setting. Also here, the parties need not trust any piece of information that comes from external parties or sources. (We note that in [LPT04] the trust assumptions are presented as a modification to the basic framework, along the lines of the "timing model" of [DNS98, G02]; still, one can equivalently present them as an ideal functionality withing the basic UC framework.)

A.3 Relaxations and enhancements of the UC framework

We review works that propose alternatives to and variants of the UC framework, either for the purpose of strengthening the security guarantee, or alternatively for relaxing it in order to facilitate realizations.

Relaxations. The above impossibility results naturally bring up the question of whether one can relax the UC framework so that it will be possible to naturally express security requirements of general tasks in a way that is realizable by protocols in the plain model (or easier to realize in general), but still guarantees reasonable composability and security.[13] It turns out, however, that such an approach will necessarily result in a definition of security that either does not provide general composability guarantees, or alternatively is too weak to guarantee even stand-alone security according to a basic simulation-based definition such as, say, the one in [C00, G04]. Specifically, we have that any protocol that realizes some functionality according to these notions and continues to do so even when composed with arbitrary protocols, is guaranteed to realize the same functionality with UC security [L03, C06]. This means that one cannot meaningfully relax the security requirements of the UC framework without ending up with a definition of security that is either weaker than the basic definition of stand-alone security or is not preserved under universal composition.

In [L04] these negative results are shown to hold even for the case of self composition, where it is guaranteed that all protocol executions in the network are instances of the same protocol, as long as the *inputs* to the computation can be chosen adaptively while the computation proceeds.

Still, some meaningful relaxations of the UC framework have been proposed. Pass and Rosen [PR03, P04, PR05a, PR05b] show general realizability in the plain model for the cases when there is an a-priori known bound on the number of protocol instances running concurrently, or alternatively some "synchronizing event" that limits the concurrency, or alternatively the inputs to the computation are chosen in advance. Prabhakaran and Sahai [PS04] propose a family of extensions of the UC framework; furthermore, they demonstrate a specific extended framework where security is preserved under universal composition, and where any functionality is realizable *in the plain model.* It thus follows that the [PS04] notion does not always imply even the basic notion of [C00, G04]. Nonetheless, [PS04] provides evidence that this notion is strong enough in a number of interesting cases.

[13]It should be stressed that providing secure composability does not in itself guarantee that the basic security properties are satisfied. To exemplify this point, consider the "definition of security" that allows all protocols to be "secure". This definition is certainly preserved under universal composition, but it does not guarantee any security.

Essentially, the idea in the [PS04] extension is to enhance the ideal-process adversary (namely, the simulator) by giving it access to super-polynomial computational resources (called "angels"), while restricting these resources in ways that depend on the current global state of the system. The rationale here is that many ideal functionalities are formulated in a way that guarantees security even against computationally unbounded adversaries. In such cases, security will be guaranteed even if the simulator has super-polynomial computational power. Next, in order to guarantee universal composability, the environment and real-life adversary are also given access to the same angels. This last step only strengthens the definition, so the same security guarantees still hold.

However, it should be kept in mind that, when using this extended notion, the universal composition theorem is meaningful only when the "high level protocol" π that calls the secure protocol already provides a strong level of security. To see that, recall the statement of the UC theorem: Let ρ be a protocol that realizes an ideal functionality $\mathcal{F}$, and let π be a protocol that makes calls to $\mathcal{F}$. Then, for any adversary $\mathcal{A}$ that "attacks" π^ρ there exists an adversary $\mathcal{S}$ that mounts essentially the same "attack" against the original π. When $\mathcal{A}$ and $\mathcal{S}$ are polynomially bounded, as in the UC framework, this statement can be interpreted as saying: *"for and any feasible attack against π^ρ there is a comparably successful feasible attack against π".* This statement applies to any protocol π, regardless of its security. In contrast, in the extended framework the adversaries are enhanced with angels that provide super-polynomial computational power. Consequently, here the UC theorem says: *"for any (potentially enhanced) attack against π^ρ there is a comparably successful* enhanced *attack against π."* This means that if the original π does not withstand enhanced attacks then no security is guaranteed with respect to π^ρ, *even against feasible, non-enhanced attacks.* (When the angels can be implemented using only slightly super-polynomial resources, the distinction between the two notions becomes smaller and more quantitative, rather than qualitative.)

Finally, we note that much of this discussion applies also to other notions of security that allow for simulation using quasi-polynomial resources, as in [P04, BS05].

A non simulation-based notion of security for multi-party function evaluation was shown to be generally realizable in the plain model without trusted set-up other than authenticated communication [MPR06]. However, its composability properties are not known.

A remark regarding terminology: The study of relaxed variants of the UC framework highlights the fact that this work uses the same terminology (UC) for two very different objects: a definition of security and a composition operation. Indeed, there may be multiple valid definitions of security that are preserved under universal composition. For this reason, "UC security" is called "environmental security" in some places, e.g. [G04, PS04]. We prefer the term "UC security" since it best communicates the main motivation behind this restrictive notion of security. Also, as the bounds in [L03, L04, C06] indicate, this is in a sense the "basic definition" to use whenever one desires simulation-based security that is preserved under universal composition.

Enhancements. The basic UC framework considers only protocols that are "well-structured" in the sense that no subroutine of an instance of the protocol is a subroutine of another protocol instance. (Recall that a protocol π is a subroutine of protocol π' if some party of π takes inputs from some party of π' or provides outputs to it.) This restriction greatly simplifies the modeling and analysis; but it comes at the price of not being able to capture systems where the same subroutine is used by multiple protocol instances. (Essentially, in the basic UC framework all these instances would have to be considered as a single instance of some more complex protocol.)

It turns out, however, that systems where multiple protocol instances use the same subroutine instance occur quite commonly. A relatively "mild" situation is where all the protocols that use the common subroutine are controlled by the protocol designer, such as in the case of multiple instances of a secure communication session protocols that use the same instance of a signature scheme. Here the problem can be often solved by using the UC with Joint State (JUC) theorem mentioned above, without extending the basic model.

A more intricate situation occurs when the same module can be used as a subroutine of *arbitrary* protocols, even protocols that are not known at the time of analysis and even protocols that may depend on the analyzed protocol. Prevalent examples include a common random string or a public key infrastructure. To handle this case, the notion of UC-emulation and the UC theorem are extended in [CDPW07] to handle protocols that use "globally available" subroutines. The main motivation for the extended notion, called generalized UC (GUC) security, to model globally available set-up information, as described in the previous sub-section.

Garay, Mackenzie and Yang [GMY04] extend the UC framework to express fairness properties, and construct a "fair" protocol for realizing general functionalities.

Halevi, Karger and Naor [HKN04] extend the UC framework to address information-flow and confinement concerns within systems with multiple levels of data secrecy.

Extensions of the UC framework to the model of quantum computation was considered in a number of works. In [CGS02] an extension of the closely related [C00] notion to quantum computation is developed. In [BM04] the general model of computation and the UC theorem are extended to the quantum setting. In [BHLMO05, RK05] the notion of UC quantum key-exchange is defined and shown to be realizable by known protocols. The work of [BM04] also has applications to the 'classical" UC framework. In particular, it is proven that the UC theorem can be extended to handle polynomially many applications of the UC operation, i.e. "polynomial depth" of nesting of subroutine calls.

A.4 Applications to symbolic and automated protocol analysis

Dolev and Yao [DY83] propose to analyze security of cryptographic protocols in a model where the cryptographic primitives in use are represented as abstract constructs that allow parties to perform only a restricted set of "symbolic operations". Effectively, this modeling postulates absolute and unconditional security properties of the cryptographic primitives. This allows designing "abstract protocols" whose security properties can often be expressed and asserted in an absolute sense, with no asymptotics, no error probabilities, no computational limitations on the power of the adversaries involved, and no computational hardness assumptions. In fact, such protocols are often entirely deterministic. This line of modeling and analysis (which is often called the "Dolev-Yao modeling") was carried out in numerous works, including [BAN90, M94a, KMM94, LO96, AG97, AFG98, FHG98, $R^{+}00$, SO99, MP04]. Although the approach is quite general, Dolev-Yao style modeling was applied primarily to protocols that are geared towards obtaining encryption and authentication of communication, using encryption and digital signatures as their cryptographic primitives. It is attractive in that it allows for relatively simple and mechanical analysis; in fact, it is readily amenable to automation. Indeed, several tools that automate this style of analysis exist, e.g. [P88].

However, Dolev-Yao style symbolic analysis does not in itself capture potential security weaknesses that come up when the symbolic operation is replaced by a real cryptographic primitive whose security is guaranteed only in a computational and probabilistic sense. Consequently, analytical works that use the Dolev-Yao abstraction cannot on their own be used to assert security of protocols. In other words, these analytical works do not by themselves provide *cryptographic soundness.*

Several approaches have been proposed for obtaining Dolev-Yao style symbolic analysis that is cryptographically sound. One way was proposed in Micciancio and Warinschi [MW04] (based on Abadi and Rogaway [AR00]): Provide direct mappings between symbolic protocols and fully specified ones, and demonstrate that the fully specified protocols demonstrate some security properties if and only if the corresponding symbolic protocols demonstrate some symbolic properties. This approach is indeed simple and attractive. However, since it does not provide any secure composability guarantees, one is forced to perform the symbolic analysis directly on an entire system, which often contains a potentially unbounded number of protocol instances. Consequently, automated analysis using this approach encounters undecidability results even for very simple protocols [EG83, DLMS04].

An alternative approach, proposed in [PW00, C01], is to interpret the Dolev-Yao symbolic operations as ideal functionalities in a composable security framework. When appropriately formulated, these ideal functionalities would allow the protocols that use them to be analyzable without having to address computational issues, in a similar way to the Dolev-Yao modeling. Cryptographic soundness would then follow from an appropriate composability property (such as universal composition).

One way for substantiating this approach was carried out in a sequence of works by Backes, Pfitzmann and Waidner, including [BPW03, BPW03a, BP04, S+06]. They formulate, within their reactive simulatability framework, such an ideal functionality (which they call a composable cryptographic library (CCL)), and show how security properties of protocols can be asserted using the CCL. By the composition theorem, these properties remain intact even when the CCL is replaced by a cryptographic protocol that realizes it. Still, even the abstract, "symbolic" analysis has to be done within their full-fledged cryptographic framework. More importantly, it is inherent in the formulation of the CCL that the abstract analysis must still be done directly on an entire system which may contain multiple sessions of multiple protocols, thus encountering the same undecidability results mentioned above. More discussion of the formulation of CCL appears in [C04].

Canetti and Herzog [CH04] propose a different approach for carrying out computationally sound Dolev-Yao style symbolic analysis. As in [MW04], they provide mappings between fully-specified protocols and symbolic protocols in a simplified model. However, the security properties they assert for the fully-specified protocols are formulated within the UC framework. Furthermore, they are formulated in terms of a *single session* of the analyzed protocol. This allows using the universal composition theorem; in particular, it allows applying the symbolic analysis to only *a single session* of a protocol rather than to an entire multi-instance system. This circumvents the above-mentioned undecidability results. Indeed, they show how their symbolic criteria can be verified in a fully automated way using an existing tool (specifically, that of Blanchet [B04]). The analysis in [CH04] applies to protocols that use public-key encryption as their only cryptographic primitive. Patil [P05] extends this work to handle protocols that use digital signatures in addition to public-key encryption.