# Universally Composable Security: A New Paradigm for Cryptographic Protocols*

Ran Canetti†

January 28, 2005

**Abstract**

We propose a new paradigm for defining security of cryptographic protocols, called universally composable security. A salient property of definitions that follow this paradigm is that they guarantee security even when the analyzed protocol runs alongside an unbounded number of unknown (even maliciously designed) protocols, or more generally when the protocol is used as a component of an arbitrary distributed system. This property is essential for maintaining security of cryptographic protocols in complex and unpredictable environments, such as the global Internet. In addition, it allows for very modular design and analysis of protocols.

We formulate a general framework that allows writing universally composable definitions of security for practically any cryptographic task. We then exemplify the expressive power of this framework by capturing within it a number of standard communication models and cryptographic primitives that were traditionally defined in a variety of different ways.

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

This version is missing Section 7, which is still under construction. A complete version should be available shortly. Please check http://eprint.iacr.org/2000/067.

*This is an updated version. While the overall spirit and the structure of the definitions and results in this paper has remained the same, many important details have changed. We point out and motivate the main differences as we go along. Earlier versions of this work appeared in October 2001, under the same title, and in December 2000 under the title "A unified framework for analyzing security of protocols". These earlier versions can be found at the ECCC archive, TR 01-16 (http://eccc.uni-trier.de/eccc-reports/2001/TR01-016); however they are not needed for understanding this work and have only historic significance.

†IBM T.J. Watson Research Center. Email: canetti@watson.ibm.com.

# 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 concern in assessing the security of cryptographic protocols is their robustness 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, an important criterion for cryptographic definitions of security is whether they guarantee robustness to the execution environment, or more generally whether they guarantee security when the protocol is used as a component within a larger system.

In contrast, cryptographic primitives (or, *tasks*) were traditionally first defined as stand-alone protocol problems 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, as well as simple 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, CGGM00, BGGL04]; Key Exchange, where the original notions did not suffice for providing secure sessions [BR93, BCK98, Sh99, CK01]; Oblivious Transfer [R81, EGL85, GM00].

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 treat the protocol as stand-alone but guarantee *secure composition.* That is, here definitions of security inspect only a single instance of the protocol *"in vitro"*. Security in complex settings, where a protocol instance may run concurrently with many other protocol instances, on arbitrary inputs and in an adversarially controlled way, 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. Furthremore, it guarantees security in arbitrary protocol environments, even ones which have not been explicitly considered.

In order to make such an approach (and in particular, such a composition theorem) meaningful, we first need to have a general framework for representing cryptographic protocols and the security requirements of cryptographic tasks. Indeed, 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 some of these works and their relation to the present one in Section 1.4.

This work proposes a framework for representing and analyzing cryptographic protocols. Within this framework, we formulate a general and uniform methodology for expressing the security requirements of practically any cryptographic task in a clear, concise and intuitively satisfying way. Furthremore, we define (extending prior work) a very general method for composing protocols, and show that definitions of security generated 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] In a nutshell, universal composition can be viewed as a generalization of the natural "subroutine substitution" composition operation for sequential algorithms to a distributed setting with multiple concurrently running protocols. 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 security under concurrent composition and non-malleability are satisfied in a strong sense, i.e. with respect to arbitrarily many instances of either the same protocol or other protocols.

The rest of the Introduction is organized as follows. Section 1.1 provides a brief overview of the proposed framework, the composition operation, and the security preservation theorem. Section 1.2 and 1.3 sketch how several popular models of communication and cryptographic tasks can be captured within this framework. Finally, Section 1.4 reviews related work, including both prior work and work that was done following the publication of the first version of this work.

## 1.1 The UC security framework

We briefly sketch the proposed framework and highlight some of its properties. A more comprehensive overview is postponed to Section 2. 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 discussed in Section 1.4.

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. See more details in Section 1.4. 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 in [C00] consists of a set of interactive Turing machines (ITMs) representing the parties running the protocol, plus an ITM representing the adversary. The adversary controls a subset of the parties, which in general may be chosen adaptively throughout the execution. 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 all parties and of the adversary 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 $\pi$ 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 $\pi$ 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 $\pi$ securely evaluates $f$ then the parties running $\pi$ 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 $\pi$ is generatable by an adversary that only gets the inputs and outputs of the corrupted parties. (We refer the reader to [C00] for more discussions on the implications of and motivation for this definitional approach.) 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.

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 $\pi$ 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 $\pi$ 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 $\pi$ securely realizes function $f$ with respect to this type of "interactive environment" then we say that $\pi$ 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 information flow is essential for capturing the threats of a multi-instance execution environment. 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. (In particular, $\mathcal{S}$ 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 be 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 stress that this is an independent extension of the model that is unrelated to the previous one. 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 above notion of security is equivalent to the seemingly stronger variant where the ideal adversary $\mathcal{S}$ is restricted to black-box access to the adversary $\mathcal{A}$. It is also equivalent to the seemingly weaker variant where $\mathcal{S}$ may depend on the environment $\mathcal{Z}$. (We remark that in other frameworks these variants result in seemingly different formal requirements.)

Another contribution of this work is in defining a basic model for a system of interacting Turing machines (ITMs). This model extends the definitions of [GMRa89, G01], which concentrate on pairs of ITMs. The new definitions are aimed at capturing open distributed systems where multiple parties run multiple different protocols, and where multiple instances of each protocol coexist. Furthermore, neither the number or identities of the participants in each protocol instance, nor the number of protocol instances running concurrently, are known in advance. We also put forth a notion of probabilistic polynomial time for interacting Turing machines, which behaves well within our model.

**Universal Composition.** Consider the following method for composing two protocols into a single composite protocol. Let $\rho$ be a protocol that UC-realizes some ideal functionality $\mathcal{F}$, according to the above definition. In addition, let $\pi$ be some arbitrary protocol (we think of $\pi$ as a "high level design" protocol) where, in addition to interacting in the usual way, the parties make ideal calls to multiple instances of $\mathcal{F}$. That is, protocol $\pi$ can be regarded as a "hybrid protocol" that uses both standard communication and instances of the ideal process for $\mathcal{F}$. Indeed, we say that $\pi$ is an $\mathcal{F}$-hybrid protocol. It is stressed that the different instances of $\mathcal{F}$ are running at the same time without any global coordination. They are distinguished via special *session identifiers,* generated by $\pi$.

Now, construct the composed protocol $\pi^\rho$ by starting with protocol $\pi$, and replacing each call to a new instance of $\mathcal{F}$ with an invocation of a fresh instance of $\rho$. Similarly, a message sent to an existing instance of $\mathcal{F}$ is replaced with an input value given to the corresponding instance of $\rho$, and any output of an instance of $\rho$ is treated as a message received from the corresponding instance of $\mathcal{F}$. It is stressed that, since protocol $\pi$ may use an unbounded number of instances of $\mathcal{F}$ at the same time, we have that in protocol $\pi^\rho$ there may be an unbounded number of instances of $\rho$ which are running concurrently on related inputs.

The universal composition theorem states that running protocol $\pi^\rho$ with no access to $\mathcal{F}$ has essentially the same effect as running the $\mathcal{F}$-hybrid protocol $\pi$. 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 $\pi^\rho$, or with $\mathcal{A}_\mathcal{F}$ and parties running $\pi$. In particular, if $\pi$ securely realizes some ideal functionality $\mathcal{G}$ then $\pi^\rho$ securely realizes $\mathcal{G}$.

***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. An example of a context where this interpretation is put to use is the proof of security in [CKOR00]. Note that in this interpretation the protocol designer knows in advance which protocol instances are running together and can control the way protocols are scheduled.

The above interpretation is indeed very useful. Here, however, we propose a different use of the composition theorem: We use it as a tool for gaining confidence in the sufficiency of a definition of security in a given protocol environment. Indeed, protocols that satisfy a UC definition are guaranteed to maintain their security 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 instances of the protocol (or other protocols altogether) that may be running concurrently in the system in an adversarially coordinated manner. This is a strong guarantee.

## 1.2 Capturing various communication models

There are many communication and adversary models for designing and analyzing cryptographic protocols. In some cases the models represent abstractions that are later instantiated by other protocols; in other cases the models represent real physical assumptions on the underlying communication network. Examples include several flavors of synchrony and reliability guarantees on the

communication, several flavors of authenticity and secrecy guarantees, and several levels of restricting the behavior of corrupted parties. While security definitions typically follow the same approach in all models, precise and workable formulations are traditionally very much model-specific. Consequently, we have multiple variants of a definitional framework, one for each specific model. This is a tedious state of affairs, which is also somewhat limited in its applicability to special and new settings.

We take a different approach toward defining various models of computation: We keep the basic framework simple and unchanged. Different communication models are captured by letting the parties to call (multiple instances of) an appropriate ideal functionality that captures the guarantees provided by the relevant model. This way, the same ideal functionality can be viewed either as representing a physical assumption, or as an abstraction that is later realized by other protocols. (The validity of the latter interpretation is guaranteed by the UC theorem.) Another advantage of this approach is that the basic model and the composition theorem need not be re-stated and re-proven for each new communication model.

More specifically, the communication model provided by the "bare" UC framework is very basic: all messages generated by the parties are handed to the adversary, and the adversary delivers arbitrary messages of its choosing to parties. This communication method provides no guarantees whatsoever regarding the timeliness, authenticity or secrecy of message delivery. In fact, this model is hardly workable at all. On top of this model, we define ideal functionalities that capture several popular and more abstract communication models, including authenticated communication, secure communication, and synchronous communication. (Essentially, an ideal functionality $\mathcal{F}$ captures a communication model if writing a protocol in this model turns out to be equivalent to writing the protocol as an $\mathcal{F}$-hybrid protocol.) We also show how to capture, within the present framework, security properties that are not necessarily preserved under composition. This allows analyzing within a single framework protocols where some of the components can be composed concurrently while other components require non-concurrent composition.

### 1.3 UC definitions of some tasks

We formulate and study universally composable definitions of a number of standard cryptographic tasks. In fact, much of the definitional work is already done by the general framework described above. All that is left to do on the definitional side is to formulate ideal functionalities that capture the security requirements of these tasks.

As seen below, some of these tasks are simply to realize some of the (ideal functionalities that represent the) communication models mentioned in the previous section. In fact, the present formalization removes any formal distinction between "realizing a model of computation" and "realizing a cryptographic task". Both goals are cast as "realizing an ideal functionality".

Throughout, we define the tasks for a *single instance,* even when the expected use is for multiple concurrently running instances. This allows for relatively simple formulations of ideal functionalities, as well as simpler constructions protocol analysis. Security in a multi-party, multi-instance setting is guaranteed via the universal composition theorem.

The task of *message authentication* is considered first, by formulating an appropriate ideal functionality, called $\mathcal{F}_{\text{AUTH}}$. We remark that, following the above methodology, the 'authenticated communication variant of the UC framework' is defined as the model where each transmission of a message is replaced by the appropriate instance of $\mathcal{F}_{\text{AUTH}}$. In other words, protocols that use authenticated communication are captured as $\mathcal{F}_{\text{AUTH}}$-hybrid protocols. Let us sketch the formulation of $\mathcal{F}_{\text{AUTH}}$. $\mathcal{F}_{\text{AUTH}}$ first expects to be invoked with a request by some party, $T$, to transmit a message

$m$ to another party, $R$. It then forwards $(T, R, m)$ to the adversary, and waits for a response. Once the adversary returns an "ok" message, $\mathcal{F}_{\text{AUTH}}$ forwards $(T, m)$ to $R$ and halts. (If $T$ is corrupted then $\mathcal{F}_{\text{AUTH}}$ allows the adversary to modify the contents of $m$.) This formulation guarantees that if $R$ received a message $m$ from an uncorrupted $T$ then $T$ indeed sent that message. It does not guarantee secrecy (since the adversary sees the message), nor does it guarantee that the message will be delivered. Note that $\mathcal{F}_{\text{AUTH}}$ is defined with respect to a authenticating a *single message* between two parties. Authentication of multiple messages in a multi-party setting can be obtained by using multiple instances of a protocol that realizes $\mathcal{F}_{\text{AUTH}}$.

The task of providing secure (i.e., authenticated *and secret*) transmission of individual messages is captured in a similar manner, with the exception that now the ideal functionality (called the *secure message transmission functionality,* $\mathcal{F}_{\text{SMT}}$) does not disclose the message $m$ to the adversary, unless either the sender or the receiver are corrupted. The 'secure communication variant of the UC framework' is formalized as the model where each secret transmission of a message is replaced by the appropriate instance of $\mathcal{F}_{\text{SMT}}$. We show that standard semantically secure encryption (or alternatively non-committing encryption for adaptive adversaries) is sufficient in order to realize $\mathcal{F}_{\text{SMT}}$, assuming authenticated communication (i.e., when using$\mathcal{F}_{\text{AUTH}}$). We note that here each message is encrypted using a different public/private key pair.

Next we formulate ideal functionalities that capture the tasks of *secure sessions* and *key exchange.* Secure sessions is an extension of $\mathcal{F}_{\text{SMT}}$ to the case where a sequence of messages between a pair of parties are secured together. The main advantage of this functionality over the previous ones is that it allows for more efficient realizations, via key-exchange combined with symmetric cryptography using the generated keys. The key exchange functionality $\mathcal{F}_{\text{KE}}$ essentially provides pairs of parties with ideally chosen random values. In particular, protocols that securely realize $\mathcal{F}_{\text{KE}}$ are guaranteed to satisfy the security notion of [CK01]. Further relations between $\mathcal{F}_{\text{KE}}$ and the [CK01] notion are studied in [CK02].

Next the tasks of *public-key encryption* and *digital signatures* are addressed. Securely realizing the signature ideal functionality turns out to be essentially equivalent to existential unforgeability against chosen message attacks as in Goldwasser Micali and Rivest [GMRi88]. In the case of public-key encryption (where many messages may be encrypted by different parties using the same public key), securely realizing the proposed functionality turns out to be equivalent to security against adaptive chosen ciphertext attacks [RS91, DDN00, BDPR98].

We then proceed to formulate ideal functionalities representing "classic" two-party primitives such as *coin-tossing, commitment, zero-knowledge,* and *oblivious-transfer.* These primitives are treated as two-party protocols in a multi-party setting. As usual, the composition theorem guarantees that security is maintained under concurrent composition, either with other instances of the same protocol or with other protocols, and within any application protocol. In particular, non-malleability with respect to an *arbitrary* set of protocols is guaranteed. See section 1.4 for discussion on the realizability of these primitives, and on realizing other primitives within the UC framework.

## 1.4 Related work

This section briefly surveys some works that are closely related to this one. Some of the reviewed work was done concurrently to the first version of this work or subsequently to it. We apologize for any omissions and mis-representations. If you notice any, please let us know.

For clarity, we organize the presentation by topics, rather than by chronological order. We first discuss other general frameworks for defining cryptographic security of protocols. Next we

discuss work on connections with formal and symbolic analysis of protocols. Next work on the realizability of UC definitions of security and the minimality of the security requirements of the UC framework is reviewed, including potential relaxations and set-up assumptions. This is followed by a quick review of some extensions of the UC framework. Work on defining and realizing specific cryptographic primitives within the UC framework is discussed last.

**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 expressed 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 approach of defining security via comparison with an ideal process involving a trusted party (albeit in a very informal way).

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, [MR91, B91] only address unconditional security and do not deal with computational issues.) [GL90, MR91, B91] are surveyed in [C00] in more detail.

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 6.5.) In addition, security of protocols in that framework is shown to be preserved under modular composition. Early versions (e.g. [C98]) also sketch how the framework can be extended to handle a concurrent version of modular composition as well as reactive functionalities. The UC framework implements these sketches in a direct manner.

The framework of Hirt and Maurer [HM00] is the first to rigorously address the case 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 and interesting 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 realistic settings where the adversary has access to the communication between honest parties.

The framework of Pfitzmann, Steiner and Waidner [PSW00, PW00] is the first to rigorously address *concurrent* composition in a computational setting. (This work is based on [PW94], which contains a basic system model but no notions of security.) They provide a definition of security for reactive functionalities and prove that security is preserved when a *single* instance of a subroutine protocol is composed concurrently with the calling protocol.

All the work mentioned above assumes *synchronous* communication. Security for asynchronous communication networks was first considered in [BCG93, C95]. An extension of the [PSW00, PW00]

framework to asynchronous networks appears in [PW01].

At high level, the notion of security in [PSW00, PW00, PW01] 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. First, they use a finite-state machine model of computation that builds on the I/O automata model of [L96] with some additional definitional and notational constructs. Next, 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 depends on the security parameter or is unknown in advance. (These are typically the cases where protocol composition is most challenging and subtle.) They also postulate fixed *identities* of parties and protocol instances (sessions). (The modeling here is quite different in these respects. See more discussion in Sections 2.1 and 3.4.) Other technical differences include the order of activations, the scheduling of messages, and the generation of outputs. See [DKMR05] for a comparison of these aspects of the respective frameworks.

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 their notion of security (called 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. A related line of ideal-process based definitional work [LMMS98, LMMS99, DKMR05] is discussed below in the context of formal protocol analysis.

Last but not least, we mention the pioneering work of Dolev, Dwork and Naor [DDN00]. While this work does not define a general security framework, it points out some important security concerns that arise when running several cryptographic protocols within a larger system. Making sure that the concerns pointed out in [DDN00] are addressed plays a central role in the present framework.

**Connections with formal and automated protocol analysis.** There are several well-known frameworks for formally analyzing certain characteristics of distributed systems and protocols, such as the CSP model of Hoare [H85], the $\pi$ calculus of Milner et. al. [MPW92] and the I/O automata of Lynch [L96]. These frameworks and others are very attractive, first in that they provide a uniform way of analyzing distributed protocols, and more importantly in that they naturally lend to automation of the analysis. However, while these frameworks have mechanisms for expressing adversarial behavior of some parts of the system such as adversarial message scheduling and faulty processors, they do not have mechanisms to express computational bounds on processes and adversaries, nor can they express randomized protocols. In contrast, cryptographic protocols are typically secure only against computationally bounded adversaries, and use randomness in an essential way. Lynch and Segala [SL95] propose an extension of I/O automata that can express probabilistic protocols. Still, their model does not address computational issues.

One way to address these analytical limitations was proposed somewhat implicitly in Needham and Schroeder [NS78] and more explicitly in Dolev and Yao [DY83]: Represent the cryptographic primitives in use as "ideal boxes" that provide absolute secrecy and authenticity guarantees regarding the transmitted data. Such a modeling resulted in "idealized protocols" that use no randomness and whose security can be analyzed using existing tools within the model, without having to model computationally bounded adversaries or computational assumptions. This line of modeling and

analysis (which is often called the "Dolev-Yao style abstraction") was carried out in numerous works, including [BAN90, M94a, KMM94, L96, AG97, AFG98, R+00, So99, FHG98, MP04]. It is attractive in that it allows for relatively simple analysis. More importantly, it is readily amenable to automation. Indeed, several automated tools that provide this style of analysis exist. In all, Dolev-Yao style analysis was instrumental in finding flaws in many protocols, including widely used and standardized ones.

However, the Dolev-Yao style abstraction ignores potential security weaknesses that come up when the "ideal box" 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 efforts have been made towards bridging this analytical gap. One approach, taken by Lincoln, Mitchell, Mitchell and Scedrov [LMMS98, LMMS99], is to directly extend the formal models to capture probabilistic protocols and computationally bounded adversaries (see also [MMS98, IK03]). Specifically, they introduce a variant of the $\pi$-calculus that incorporates random choices and computationally limitations on adversaries. In that setting, their approach has a number of similarities to ours. They define a notion 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. 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. Furthermore, they do not state a composition theorem in their model.

Datta, Küsters, Mitchell, and Ranamanathan [DKMR05] extend the [LMMS98, LMMS99] framework to express simulatability as defined here, cast in a polytime probabilistic process calculus, and demonstrate that the UC theorem holds in their framework. They also rigorously compare certain aspects of UC security (as defined in [C01]) and reactive simulatability (as defined in [BPW04a]), such as the relations between standard simulatability and black-box simulatability.

Another approach towards bridging the analytical gap created by the Dolev-Yao abstraction is to provide a Dolev-Yao style abstraction that is also cryptographically sound. Abadi and Rogaway [AR00] were the first to propose such a mechanism: They devised a formalism for arguing about 'formal indistinguishability' of expressions based on formal encryption in an abstract and unconditional way. At the same time, it is guaranteed that if two formal expressions are 'formally indistinguishable' then the corresponding distribution ensembles, obtained by replacing the formal encryption by a semantically secure symmetric encryption scheme, are computationally indistinguishable in the standard sense. This methodology was extended in several ways, most notably in Micciancio and Warinschi [MW04] to the case of mutual authentication protocols based on public-key encryption schemes.

An alternative approach for obtaining cryptographically sound Dolev-Yao style abstraction is to represent the Dolev-Yao "ideal boxes" as ideal functionalities in a composable security framework. When appropriately formulated, these ideal boxes would allow the protocols that use them to be analyzable without having to address computational issues, and sometimes even deterministically. Cryptographic soundness would then follow from an appropriate composition theorem (such as the UC theorem). This approach was proposed in [PW00, C01]. In a sequence of works, including [BPW03, BPW03a, BP04], Backes, Pfitzmann and Waidner formulate such an ideal box (i.e., an ideal functionality) within their reactive simulatability framework, and show how certain proper-

ties of several known protocols can be asserted using their ideal functionality (which they call a "composable cryptographic library"). Still, even the abstract analysis has to be done within their cryptographic framework. More discussion of the [BPW03] formalization appears in [C04].

Canetti and Herzog [CH04] propose a different way for formulating Dolev-Yao style ideal boxes within a composable security framework. They show how to translate protocols in the UC framework into formal protocols in a variant of an existing abstract process algebra for which automated analysis tools exist (see e.g. [L96, R+00, So99]). It is guaranteed that the concrete protocol is a UC mutual authentication (resp., UC key exchange) protocol *if and only if* the corresponding abstract protocol satisfies an abstract property that is verifiable within the abstract algebra. This further simplifies the analysis since the formal analysis is performed in a simplified model and addresses only a single instance of the analyzed protocol. For concreteness, [CH04] concentrate for protocols for mutual authentication and key exchange, based on public-key encryption as the only cryptographic primitive.

**On the realizability and minimality of UC definitions of security.** Definitions of security in the proposed framework are more stringent than other definitions and are not always realizable. It is thus natural to ask which cryptographic tasks are realizable in a way that guarantees composability, and under which set-up and computational assumptions. An important and intimately related question is whether it is possible to define security of protocols in a more relaxed way, that would be realizable by simpler protocols and with milder set-up assumptions, and at the same time would still guarantee reasonable security and universal composability.

We first note that some known protocols for general secure function evaluation are, in fact, universally composable. For instance, the [BGW88] protocol (both with or without the simplification of [GRR98]), together with encrypting each message using non-committing encryption [CFGN96], is universally composable as long as less than a third of the parties are corrupted. Using [RB89], any corrupted minority is tolerable. The asynchronous setting can be handled using the techniques of [BCG93, BKR94]. These facts were stated in [C01] with a sketch of proof. We leave full proof out of scope for this work. [C01] also demonstrates how to use these protocols in a straightforward way to realize practically *any* ideal functionality, even reactive ones and even "two-party functionalities" (i.e., functionalities where only two parties have inputs and outputs). This is done by having the parties use a set of "helper parties", or "servers," where it is assumed that only a minority of the servers are corrupted.

However, things are different when no honest majority of the parties is guaranteed, and in particular in the case where only two parties participate in the protocol, and either one of the parties may be corrupted. In this case it was shown in [CF01] that a natural and basic formalization of ideal commitment cannot be realized in the UC framework by plain protocols, even if ideally authenticated communication is provided. (We alternately use the terms *plain protocol* and *protocol in the plain model* to denote a protocol that do not use any ideal functionality, except for the authenticated communication functionality, $\mathcal{F}_{\text{AUTH}}$.) Similar impossibility results were 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 impossibility of realizing almost all "non-trivial" deterministic two-party functions and many probabilistic two-party functions by plain protocols. This in particular implies that none of the known two-party protocols for any of the above tasks is UC secure. This is mainly due to the fact that one of the most common proof-techniques for cryptographic protocols, namely black-box simulation with rewinding of the adversary, does not in general work in the present framework. (Indeed, here the ideal adversary has to interact with the environment which cannot

be "rewound".)

Still, all is not lost: It was shown in [CF01, DN02, DG03] how to realize the commitment and zero-knowledge functionalities via protocols that use ideal access to the coin-tossing ideal functionality. Interestingly, having ideal access to the coin-tossing functionality turns out to be essentially a reformulation of the well-known *common random string model* of [BFM89]. To highlight this fact, we call the coin-tossing functionality $\mathcal{F}_{\text{CRS}}$. 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] protocol can be slightly modified to run in the plain model and remain secure against adversaries that corrupt only a minority of the parties. Security of the [CF01, CLOS02] constructions is based on standard general hardness assumptions, whereas [DN00, DG03] use specific number theoretic assumptions.

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.

Barak et. al. [BCNP04] demonstrate that all the above protocols can be modified to work under a number of alternative set-up assumptions other than having access to trusted common randomness. Specifically, they show that it is enough to have "registration authorities" where parties register "public keys" in a way that guarantees that the corresponding secret keys exist and are "extractable". In particular, no single authority or string has to be trusted by all parties.

In [B+04], Barak et. al. generalize the results of [CLOS02] 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 CRS then they can force the adversary to essentially "partition" the network to several disjoint "clusters", where each "cluster" realizes the original functionality in the usual authenticated and secure manner.

The above results naturally bring up the question of whether one can relax the UC framework so that it will still guarantee security and universal composability, but will be realizable by plain protocols (or will be easier to realize in general). Here, Lindell [L03a, L04] shows that essentially any notion of security for a given task (namely, for a given ideal functionality) that implies the basic security notion of [C00], and preserves security under universal composition, implies UC security. Roughly speaking, this means that one cannot meaningfully relax the security requirements of the UC framework without ending up with a definition of security that is weaker than the basic definition of stand-alone security. (We remark that Lindell uses the term "general concurrent composition" to denote universal composition.) Still, some meaningful relaxations of the UC framework exist, as described below.[2]

We remark these negative results 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. In contrast, in the case of self composition with bounded concurrency, i.e. when there is an a-priori known bound on the number of protocol instances running concurrently, securely realizing general tasks is possible [L03, PR03, P04].

[2] It should be stressed that providing secure composability does not necessarily 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.

***Extensions of the UC framework.*** Several extensions of the UC framework were proposed. Prabhakaran and Sahai [PS04] propose a family of variants of the UC framework; furthermore, they demonstrate a specific relaxed variant where security is preserved under universal composition, and where any functionality is realizable *in the plain model.* Indeed, the resulting notion is weaker than the [C00] notion of security. Nonetheless, [PS04] provides evidence that this notion is strong enough in a number of interesting cases. Specifically, it is shown that if a protocol realizes some functionality according to a [PS04] variant of the UC framework, then this protocol realizes a somewhat weakened version of this functionality in the unmodified UC framework, when given ideal access to another (rather mild) ideal functionality.

Lindell, Prabhakaran and Tauman [LPT04] propose an extension of UC security to the "timing model" of [DNS98, G02], where message delay and clock drifts are assumed to be bounded by known values. In that model they show that any functionality can be realized by protocols whose security is maintained under universal composition, given that *all* protocols in the network obey certain timing bounds. No set-up assumptions, other than authenticated communication, are needed.

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] develops an extension of the closely related [C00] notion to quantum computation. In [BM04] the general model of computation is extended and the UC theorem is re-proven. 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 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. [G01, PS04]. We prefer the term "UC security" since it best communicates the main motivation behind this restrictive notion of security. Also, as the results of Lindell indicate, this is in a sense the "basic definition" when one needs security that is preserved under universal composition.

**Defining and realizing specific primitives within the UC framework.** The UC framework has been used to capture the security requirements of a number of cryptographic primitives and to analyze a number of protocols. Here we briefly review this body of work.

A tool that is used throughout is the *universal composition with joint state (JUC)* theorem of [CR03]. This theorem asserts that, under certain conditions, the UC theorem can be applied even in cases where the composed protocol instances have some joint state and randomness. This powerful tool enables modular analysis in many cases where we would otherwise be forced to analyze an entire complex system as a single atomic unit. (See more details in Section 5.3. some examples appear below.)

***Key exchange and secure communication.*** 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 [CK02A], they prove that the "cryptographic core" of the key exchange protocol of the IPSEC security standard [IPSEC] is a UC key exchange protocol. Hofheinz et. al. 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.

[CHKLM04] define UC password-based key exchange protocols, and show that a variant of the [GL03] protocol is secure under their definition. Formulating an ideal functionality that adequately captures the subtleties of password-based key exchange turns out to be a non-trivial task; in particular their formulation is quite different than the first formulation that comes to mind.

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.

***Public key encryption and signatures.*** An ideal functionality for capturing the security properties of digital signatures, $\mathcal{F}_{\text{SIG}}$, was proposed in the first version of this work [C01], and was 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 in several iterations, in [CK02, CR03, BH04, C04]. The formulation from [C04] (included in this work for completeness) keeps the approach of the original formulation and the equivalence with the [GMRi88] notion. [C04] also shows how to realize authenticated communication via protocols that use $\mathcal{F}_{\text{SIG}}$ plus ideal "registration services" where parties can register their public verification keys. We note that the [C04] analysis is focused on authenticating a single message using a single instance of $\mathcal{F}_{\text{SIG}}$. Validity in the case where multiple messages are authenticated using the same instance of $\mathcal{F}_{\text{SIG}}$ (i.e., using a single signature/verification key) per party is obtained via the JUC theorem.

[C01] formulates an ideal functionality, $\mathcal{F}_{\text{PKE}}$, that is aimed at capturing the security properties of public-key encryption. It was also claimed that realizing $\mathcal{F}_{\text{PKE}}$ is a strictly weaker property than security against chosen ciphertext attacks (CCA security) as in [RS91, DDN00]. [CKN03] show 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 replayable encryption, and demonstrate that this variant is in fact sufficient for most applications of CCA-secure encryption. [CH04] point out a weakness in previous formulations of $\mathcal{F}_{\text{PKE}}$ and show how to fix this weakness while maintaining all the properties of $\mathcal{F}_{\text{PKE}}$ that were claimed in the previous works.

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 can effectively erase past data. [CHK05] proposes a relaxation of $\mathcal{F}_{\text{PKE}}$, where even the legitimate receiver is able to decrypt ciphertexts only in a given time window, and show how to realize this relaxed functionality under certain number-theoretic assumptions.

***Two-party protocols.*** UC commitment protocols in the common reference string (CRS) model were constructed in [CF01, DN00, DG03]. The protocol of [CF01] is based on general assumptions 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 scheme of Damgaard and Nielsen

[DN00] is interactive, uses reference string that 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 protocols in the random oracle model [HM04].

Garay, Mackenzie and Yang construct UC protocols for committed Oblivious Transfer [GMY04a]. Interactive Zero-Knowledge protocols for any language in NP were constructed in [CF01], and non-interactive ones in [CLOS02], both under general assumptions and in the CRS model. Barak et. al. [BCNP04] demonstrate how to obtain UC non-interactive Zero-Knowledge in their "key registration model" discussed above. 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.

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 have the attractive property that they are realizable by more efficient protocols.

***Multi-party protocols.*** 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 above-mentioned general feasibility result in [CLOS02] for multi-party functionalities.

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

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 Wikstrom [W04a].

**Organization of the rest of this paper.** Section 2 presents an overview of the framework, definition of security, and composition theorem. The basic model for representing multiparty protocols is presented and motivated in Section 3. The general definition of security is presented in Section 4. The composition theorem and its proof are presented in Section 5. Capturing other models of computation within the basic model is discussed in Section 6. Section 7 presents and discusses a number of ideal functionalities that capture some known cryptographic primitives. Section 8 suggests some directions for future research. Throughout, we point out the differences from earlier versions of this work as we go along. (An exception is Section 2, which for clarity postpones this discussion to later sections.)

## 2 Overview of the framework

This section presents an overview of the framework, the definition of security, and the composition theorem. The presentation builds on the intuition provided in the Introduction (Section 1.1). First, in Section 2.1, we sketch the basic the computational model, which is geared towards representing multiple interacting computer programs. Next, in Section 2.2, we sketch the definition of protocol execution, the "ideal process" for realizing tasks, and the notion of security. Finally, in Section 2.3,

we sketch the composition theorem and its proof. (These sections roughly correspond to the material in Sections 3, 4 and 5, respectively.) Throughout, the goal is to present and discuss the main definitional ideas, with minimal amount of formalism. Additional discussion, regarding definitional details that are not mentioned in this section, appears in subsequent sections. To avoid duplication and to preserve the readability of this overview, we postpone the discussion on the differences from previous version of this work to subsequent sections, where the same material is covered in full detail.

## 2.1 The underlying computational model

Following [GMRa89, G01], the programs run by parties in a communication network are represented as interactive Turing machine (ITMs). The main reason for choosing ITMs as the underlying model of distributed computation is that they provide a natural basis of considering resource bounded, probabilistic computation, together with adversarial scheduling. See more discussion on this point in Section 3.4.

Due to the complexity of modeling and arguing about protocols in multi-party, multi-protocol, multi-instance settings, we chose to define the underlying "mechanics" of inter-ITM communication separately from the security model and definition. For this purpose, we formalize the notion of a system of ITMs, which represents a network of communicating computer programs. It is stressed that the definition of a system of ITMs provides only the mechanics of communication, without any notion of security. Still, Formulating this definition involves a number of choices. Indeed, this notion may be of interest even regardless of the notions of security which are the focus of the rest of this work.

Formally, a system $(I, C)$ of ITMs consists of an initial ITM $M$ and a control function $C$. An execution of a system starts by running $I$ on some external input. In addition to its local computation, $I$ may *invoke* other instances of ITMs and write information on some of their tapes. Once an ITM instance is invoked, it executes its code, and potentially invokes other ITM instances or writes to tapes of other instances. Overall, an execution of a system consists of a sequence of activations, where in each activation a single ITM instance is active (i.e., running), and may write to the tapes of only one other instance. Once this instance enters a special "waiting" state, the instance whose tapes were written to becomes active. (We remark that this extremely simple order of activations is in fact quite powerful, in that it allows us to express practically any security, liveness, and fairness requirement.) The control function $C$ determines which tapes of which instances can be written to by each instance. Figuratively, $C$ can be viewed as a central "routing ITM" which forwards information from one ITM instance to another. An execution ends when the initial ITM halts. Its output is the output of the initial ITM.

Let us highlight some properties of the notion of a system of ITMs. First, it provides a clear separation between the traditional notion of an ITM, which corresponds to a "static object", namely an algorithm or a program, and the notion of an ITM instance, which corresponds to a "run-time object", namely an instance of a program running on some specific data. In particular, the same ITM (program) may have multiple instances in an execution of a system.

Another feature is the ability to dynamically generate ITM instances "on the fly", in a potentially adversarial way, throughout the computation. This ability seems essential for adequate modeling of modern computer networks and systems. However, it introduces a number of definitional questions, such as how the programs and identities of newly generated instances are determined. We choose to let the 'invoking instance" determine these values at runtime. This is an "algorithmic approach", which provides great flexibility and power to the protocol designer. It also means that

the program of the invoking instance should contain sufficient instructions for determining the code of the invoked instance.

A related definitional question is how an ITM instance specifies which ITM instance it wishes to address in an "external write" instruction. For this purpose we let each ITM instance have an identity that is determined at invocation time (by the invoking instance) and is unchangeable. In addition, the mechanism of invoking ITM instances guarantees that identities are *globally unique,* thus preventing addressing ambiguities. The fact that identities are determined dynamically during the execution, and are readable by the ITM instances themselves, is an important feature: On the one hand, it provides a powerful tool for protocol design and analysis. On the other hand, it allows capturing within the model attacks that use adversarially chosen identities of parties.

The dynamic nature of the present definition of a system of ITMs raises another question: How to delineate, or isolate, a single "protocol instance" within an execution of a system? Indeed, the traditional notion that defines a protocol instance as a pre-determined set of ITMs, often with pre-determined identities, does not apply here. The natural informal answer to this question would probably be "a set of ITM instances in an execution of a system are a protocol instance if they, or rather their invokers, decide so". We formalize this answer by adding extra structure to the identities of ITM instances. Specifically, we let each identity consist of two separate fields: the session id (SID) and the party id (PID). A set of ITM instances at a certain moment in an execution of a system of ITMs are a protocol instance if they have the same program and the same SID. The PIDs are used to differentiate between ITM instances within a protocol instance; they can also be used to associate ITM instances with "clusters", such as physical computers in a network. This definition allows for protocol instances where the set of participants is not bounded or known a priori, and furthermore changes dynamically. Such modeling seems essential for capturing many protocols in modern computer networks.

Finally, to better capture the characteristics of multi-party, multi-instance systems, the present model provides two methods for inter-ITM communication. Communication via the *communication tapes* of parties represents "untrusted communication" where the identity and program of the writing entity is not necessarily known to the recipient. Communication via the *subroutine output tapes* represents more trusted communication, where the recipient trusts the identity and program of the writing entity. Typically, the communication tapes would model communication over a network, while subroutine input/output tapes would model local subroutine calls. This distinction is conceptually important; it also facilitates the description of the model of protocol execution, presented in the next section. A graphical depiction of an ITM appears in Figure 1.

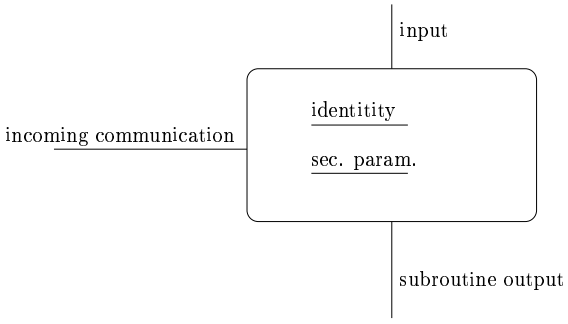


Figure 1: An interactive Turing machine. The tapes writable by other ITMs are the input, incoming communication, and subroutine output tapes. The identity and security parameter tapes are written to only at invocation time.

**Defining polynomial-time computation.** Another non-trivial choice is how to define polynomial time (or, more generally, resource bounded) computation in this highly dynamic setting. A definition should make sure that each component, as well as the overall system, is not allowed to over-consume resources, and at the same time should not restrict components in "artificial ways" that would affect the meaningfulness of notions of security. (Some pitfalls are described within, as well as in [DKMR05, HMU05].)

In a nutshell, our definition proceeds as follows. We say that an ITM $M$ is locally probabilistic polynomial time (PPT) if, at any point during its execution, its overall running time so far is bounded by a polynomial in the security parameter and the overall length of input (i.e., the number of bits written on the input tape), and in addition the number of bits written on the input tapes of other ITMs, plus the number of other ITMs that $M$ writes to, is less than the the length of $M$'s input so far. Note that incoming messages, which are written on the incoming communication tape, do not increase the bound on the running time. This notion extends the notion of PPT in [G01, Vol I, Ch. 4.2.1]. In particular, it guarantees that each execution of a system of PPT ITMs is completed within a number of steps that is polynomial in the initial input to the system. This property seems natural in cryptographic modeling. It is also essential for the composition theorem to hold.

This definition of PPT ITMs has some additional technical advantages. For instance, it allows demonstrating that some natural variants of the definition of security are equivalent; it also avoids other technical difficulties that encumber other notions. See more details and discussion in Sections 3 and 4. (We also remark that PPT adversaries are defined in a slightly more liberal way; details within.)

## 2.2 Defining security of protocols

As sketched in Section 1.1, protocols that securely carry out a given task (or, realize a functionality) are defined in three steps. First, the process of executing a protocol in the presence of an adversary and in a given computational environment is formalized. Next, an "ideal process" for realizing the functionality is formalized. A protocol is said to securely realize the functionality if the process of running the protocol amounts to "emulating" the ideal process for that functionality. We sketch these steps in more detail.

**The model of protocol execution.** We sketch the model for executing multi-party protocols in the presence of and adversary and in a given execution environment (or, "context"). The model provides only a "bare minimum" for inter-ITM communication; we thus often refer to it as the bare model. Specifically, it postulates a completely asynchronous, unauthenticated, and unreliable network, where the communication is adversarially observable and controlled. Furthermore, parties have no a-priori information on other participants (such as their identities or public keys). As seen below, more abstract and "idealized" communication models are defined on top of the bare model. In addition to simplifying the presentation, this approach provides greater power to assertions, such as the composition theorem, made on the bare model.

The model of protocol execution is parameterized by three ITMs: $\pi$, representing the protocol to be executed; $\mathcal{A}$, an adversary; and $\mathcal{Z}$, an environment. Intuitively, the adversary represents adversarial activities that are directly aimed as the protocol execution under consideration, including attacks on protocol messages and corruption of protocol participants. The environment represents all the other protocols running in the system and the adversaries thereof, including the protocols that provide inputs to, and obtain outputs from, the protocol execution under consideration. Formally, given $(\pi, \mathcal{A}, \mathcal{Z})$, the model of execution is defined as a system of ITMs with a specific control

function, and where the initial ITM is $\mathcal{Z}$. While the description below is quite informal, formal specification of the control function can be readily extracted.

Recall that an execution consists of a sequence of activations, where in each activation a single ITM instance is active and may write to the tapes of one other ITM instance, which becomes the next instance to be activated. The environment $\mathcal{Z}$ is activated first. The first instance to be invoked by $\mathcal{Z}$ is the adversary $\mathcal{A}$. In all other activations, $\mathcal{Z}$ may pass information to $\mathcal{A}$ (this information can be thought of as "instructions" or "questions"), or provide inputs to parties in an instance of $\pi$. That is, all ITM instances invoked by $\mathcal{Z}$ throughout the computation (except for $\mathcal{A}$) are required to have the program $\pi$; furthermore, all are required to have the same SID (which is determined by $\mathcal{Z}$).

Once the adversary is activated, it may either deliver a message to some party by writing this message to the party's incoming communication tape, or corrupt a party, or report some information to $\mathcal{Z}$. We do not make any restrictions on the delivered messages. In particular, they need not be related to any of the messages generated by the parties. Upon corrupting a party, the environment is notified, and the adversary gains access to and control over the internal actions of the corrupted party, according to the specific corruption model. (Formally, the corruption operation is modeled as special type of message delivered to the party; the party's response to this message is determined by a special part of the party's program.)

Once a party of $\pi$ is activated, either due to an input given by the environment or due to a message delivered by the adversary, it follows its program and possibly writes outgoing messages on the incoming communication tape of $\mathcal{A}$, or writes outputs to the subroutine output tape of $\mathcal{Z}$. In addition, parties of $\pi$ may invoke other ITM instances as subroutines, provide inputs to them, and obtain outputs from them. Once a subroutine of a party of $\pi$ (or a subroutine thereof) is activated, it follows its program in the same way. It is stressed that the parties can only send messages to $\mathcal{A}$; they cannot send messages directly to other entities.

The output of the execution is the output of the environment. Without loss of generality we assume that this output consists of only a single bit. A graphical depiction of the model of protocol execution appears in Figure 2.

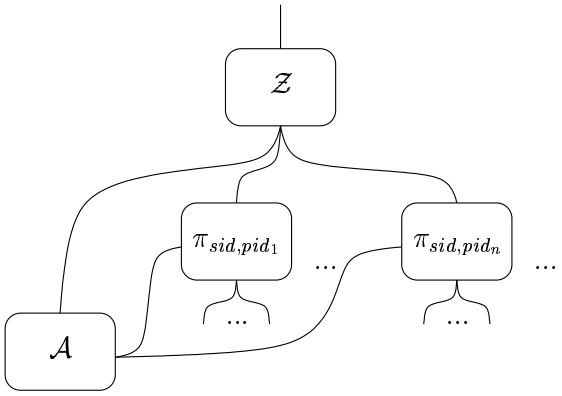


Figure 2: The model of protocol execution. The environment $\mathcal{Z}$ writes the inputs and reads the outputs of the parties running the protocol, while the adversary $\mathcal{A}$ controls the communication. In addition, $\mathcal{Z}$ and $\mathcal{A}$ interact freely via $\mathcal{A}$'s input tape and $\mathcal{Z}$'s subroutine output tape, and the parties may invoke subroutines (sub-parties). The Parties are identified via their SIDs and PIDs; all the SIDs in a protocol instance are identical.

***Discussion.*** Several remarks are in order at this point. First notice that, throughout the process of protocol execution, the environment $\mathcal{Z}$ has access only to the inputs and outputs of the parties. It does not have direct access to the communication among the parties. In contrast, the adversary $\mathcal{A}$ only has access 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, among other things, the protocol that provides inputs to and obtains outputs from the present instance of $\pi$, while $\mathcal{A}$ represents an adversary that attacks the protocol without having access to the local (and potentially secret) inputs and outputs.

Nonetheless, the order of events allows $\mathcal{Z}$ and $\mathcal{A}$ to exchange information freely between each two activations of some party. It may appear at first glance that no generality is lost by assuming that $\mathcal{A}$ and $\mathcal{Z}$ disclose their entire internal states to each other. A close 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, keeping $\mathcal{A}$ and $\mathcal{Z}$ separate is crucial for the notion of security to make sense.

Also, note that the only external input to the process of protocol execution is the input of $\mathcal{Z}$. This input represents an initial state of the system and in particular 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 it a non-uniform ITM.)

Another point to be highlighted is that the model of execution allows the adversary to corrupt individual ITM instances, which represent individual program instances (or, "processes") within computers. This allows for finer granularity and greater expressibility in defining security of protocols; in particular it allows considering security of a protocol instance even when other instances running within the same computer are compromised.

**Ideal functionalities and ideal protocols.** Security of protocols is defined via comparing the protocol execution in the real-life model to an *ideal process* for carrying out the task at hand. 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 the ideal functionality that captures the desired functionality, or the specification, of the task by way of specifying a set of instructions for a "trusted party". The ideal functionality is modeled as a special ITM $\mathcal{F}$ that serves as a "joint subroutine" of multiple ITM instances.

More specifically, the ideal protocol $\text{IDEAL}_{\mathcal{F}}$ for a given ideal functionality $\mathcal{F}$ proceeds as follows. Upon receiving an input $v$, protocol $\text{IDEAL}_{\mathcal{F}}$ instructs the party to forward $v$ as input to the instance of $\mathcal{F}$ whose SID is the same as the local SID. Any output coming from $\mathcal{F}$ is copied to the local output. (As usual, the parties of an instance of $\text{IDEAL}_{\mathcal{F}}$ are distinguished using their PIDs. The corresponding instance of $\mathcal{F}$ has a special null PID.) We often call the parties of $\text{IDEAL}_{\mathcal{F}}$ dummy parties for $\mathcal{F}$.

$\mathcal{F}$ constains instructions on how to generate outputs to parties based on their inputs. It is stressed that $\mathcal{F}$ models reactive computation, in the sense that it maintains local state and new inputs may be received after prior outputs have been generated. 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 channel"of direct communication between $\mathcal{F}$ and $\mathcal{A}$ serves several purposes. By letting $\mathcal{F}$ specify appropriate message exchanges with $\mathcal{A}$, it is possible to capture the "allowed influence" of the adversary on the outputs of the parties. It is also possible to capture the "allowed leakage" of

information on the inputs and outputs of the parties to the adversary, and the "allowed delay" in output delivery. Furthermore, corruption of parties of the ideal protocol is captured as a request from $\mathcal{A}$ to $\mathcal{F}$, and the information that $\mathcal{A}$ is allowed to obtain upon corruption is captured in the response message from $\mathcal{F}$ to $\mathcal{A}$. A graphical depiction of the ideal protocol appears in Figure 3.

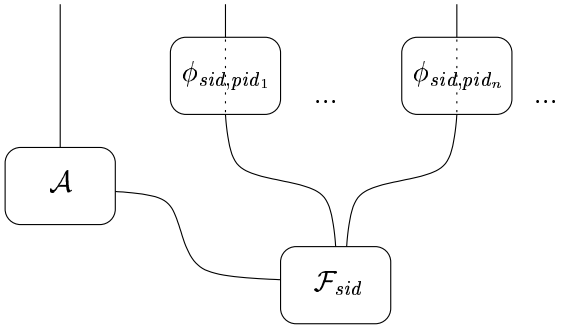


Figure 3: The ideal protocol $\phi$ for an ideal functionality $\mathcal{F}$. The parties of $\phi$ are "dummy parties": they relay inputs to the instance of $\mathcal{F}$ with the same SID, and relay outputs of $\mathcal{F}$ to their output tapes. The adversary $\mathcal{A}$ communicates only with $\mathcal{F}$.

**Protocol emulation.** Before presenting the notion of realizing an ideal functionality, we present the notion of *protocol emulation* in more general terms that apply to any two protocols. Informally, protocol $\pi$ emulates protocol $\phi$ if, from the point of view of any environment, protocol $\pi$ is "just as good" as $\phi$, in the sense that interacting with $\pi$ and some adversary is indistinguishable from interacting with $\phi$ and some other adversary. More precisely:

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

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 $\mathcal{A}$. Also, we call the emulated protocol $\phi$ as a reminder that in the definition of realizing a functionality (see below), $\phi$ takes the role of the ideal protocol for some ideal functionality $\mathcal{F}$.

We note that, in the present framework, the specific order of quantifiers in the notion of emulation is of no significance. That is, either letting the adversary $\mathcal{S}$ depend on $\mathcal{Z}$, or alternatively requiring that $\mathcal{S}$ be fixed for any $\mathcal{A}$ (with black-box access to $\mathcal{A}$), result in equivalent definitions. We choose the present order of quantifiers since it seems to be the most intuitive; it is also the most relaxed order that allows proving the composition theorem.

The above notion of emulation, as well as the rest of this work, is geared towards capturing *computational security,* namely the case where all involved parties, including all adversarial entities, have polynomially bounded computational resources. This is captured by restricting $\mathcal{Z}$, $\mathcal{A}$, and $\mathcal{S}$ to be PPT. Still, the present notion of emulation is easily extendable to capture unconditional security. Specifically, statistical security is captured by allowing $\mathcal{Z}$ and $\mathcal{A}$ to be computationally unbounded. Perfect security is captured by requiring in addition that the distinguishing probability

of $\mathcal{Z}$ is zero. In both cases it is prudent to require that $\mathcal{S}$ be polynomial in the complexity of $\mathcal{A}$ (see [C00] for discussion and rationale).

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

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

Indeed, as in more basic ideal-model based definitions such as the one in [C00], it is guaranteed that if $\pi$ UC-realizes $\mathcal{F}$ then the parties running $\pi$ will generate outputs that are indistinguishable from the outputs provided by $\mathcal{F}$ on the same inputs. Furthermore, any information gathered by an adversary that interacts with $\pi$ is obtainable by an adversary that only interacts with $\mathcal{F}$. (See [C00] for more discussion.) In addition, the definition here guarantees that security is preserved under a very general composition operation, described below.

**Hybrid protocols.** Before moving to present the universal composition operation and theorem, we define a special type of protocols that play a central role in the presentation of the composition theorem and in the rest of this work. In these protocols, in addition to communicating via the adversary in the usual way, the parties also make calls to instances of ideal functionalities. We call these protocols hybrid protocols, since they are hybrids between "real protocols" and ideal protocols. (Figuratively speaking, these instances of ideal functionalities can be thought of as "ideal services" that are available to parties in the network.)

More precisely, calling an ideal functionality is done by invoking the ideal protocol for that functionality. That is, an $\mathcal{F}$-hybrid protocol is a protocol that includes subroutine calls to $\text{IDEAL}_{\mathcal{F}}$, the ideal protocol for $\mathcal{F}$. Note that a hybrid protocol may invoke an unbounded number of instances of the ideal protocol, where each instance of the ideal protocol uses its own instance of $\mathcal{F}$. As usual, these instances, which may run concurrently, are identified via their SIDs.

## 2.3 On the composition theorem

**The composition operation.** As in the case of protocol emulation, we present the composition operation and theorem in terms of general protocols. The special case of ideal functionalities and ideal protocols follows as a special case. Let $\pi$ be a protocol that uses subroutine calls to some protocol $\phi$, and let $\rho$ be a protocol that UC-emulates $\phi$. The composed protocol, denoted $\pi^{\rho/\phi}$, is the protocol in which each invocation of $\phi$ is replaced by an invocation of $\rho$. That is, protocol $\pi$ is modified so that each instruction to provide an input to some instance of $\phi$ is is replaced with an instruction to give the same input to an instance of $\rho$ with the same identity, and each output received from an instance of $\rho$ is treated as an output received from an instance of $\phi$ with the same identity. It is stressed that an execution of $\pi$ may involve an unbounded number of concurrent copies of $\phi$. Similarly, an execution of $\pi^{\rho/\phi}$ may involve an unbounded number of concurrent copies of $\rho$. When the replaced protocol $\rho$ is an ideal protocol for some functionality $\mathcal{F}$ we denote the composed protocol by $\pi^{\mathcal{F}/\rho}$. A graphical depiction of the composition operation appears in Figure 4.

**The composition theorem.** In its general form, the composition theorem says that if protocol $\rho$ emulates protocol $\phi$ then, for any protocol $\pi$, the composed protocol $\pi^{\rho/\phi}$ emulates $\pi$. This can

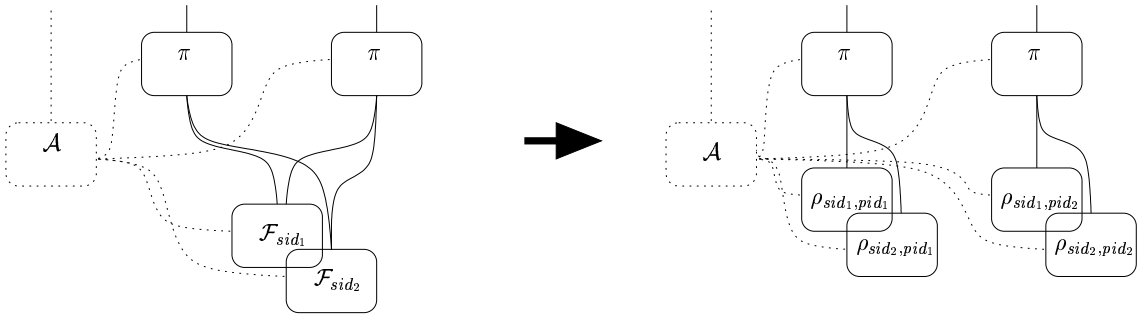


Figure 4: The universal composition operation, for the case where the replaced protocol is an ideal protocol for $\mathcal{F}$. Each instance of $\mathcal{F}$ (left figure) is replaced by an instance of $\rho$ (right figure). The solid lines represent inputs and outputs. The dotted lines represent communication. The "dummy parties" for $\mathcal{F}$ are omitted from the left figure for graphical clarity.

be interpreted as asserting that replacing calls to $\phi$ with calls to $\rho$ does not affect the behavior of $\pi$ in any distinguishable way.

A first, immediate corollary of the general theorem states that if protocol $\rho$ UC-realizes an ideal functionality $\mathcal{F}$, and $\pi$ is an $\mathcal{F}$-hybrid protocol, then the composed protocol $\pi^{\mathcal{F}/\rho}$ UC-emulates $\pi$.

Another corollary states that if $\pi$ UC-realizes an ideal functionality $\mathcal{G}$, then so does $\pi^{\rho/\phi}$.

***On the need to use the session IDs.*** Recall that in our setting the SID of each instance of $\mathcal{F}$ (resp., of $\rho$) is available for code (program) of $\mathcal{F}$ (resp. $\rho$). It is interesting to note that this property is essential. Indeed, if the SIDs are not available to the instances of $\rho$ then the composition theorem does not necessarily hold. This fact is exemplified in [LLR02] for the basic tasks of broadcast and Byzantine agreement.

***Standard concurrent "self composition" as a special case.*** The traditional notion of concurrent composition of protocols usually considers a system where many identical instances of a given protocol $\rho$ are running concurrently on adversarially controlled inputs and with an adversarially controlled scheduling of message delivery. (Here there are no other protocols except for the instances of $\rho$; following [L04], we call this type of composition *self composition*.) To see how this notion of concurrent composition is captured by the above composition operation, assume that protocol $\rho$ securely realizes functionality $\mathcal{F}$, and consider the following protocol $\pi$ in the $\mathcal{F}$-hybrid model. Whenever a party receives a message saying "Please activate instance $s$ of $\mathcal{F}$ with input $x$," the party generates input $x$ to the instance of $\text{IDEAL}_{\mathcal{F}}$ with SID $s$.

This way, the composed protocol, $\pi^{\mathcal{F}/\rho}$, allows the adversary to create a scenario that is equivalent to the traditional scenario of adversarially controlled concurrent self-composition. Consequently, the fact that $\pi^{\mathcal{F}/\rho}$ emulates $\pi$ means that protocol $\rho$ preserves its security under concurrent composition.

**On the proof of the composition theorem.** We briefly outline the main ideas in the proof of the composition theorem. Let $\phi$ be a protocol, let $\pi$ be a protocol that makes subroutine calls to $\phi$, let $\rho$ be a protocol that emulates $\phi$ and let $\pi^{\rho/\phi}$ be the composed protocol. Let $\mathcal{A}$ be a real-life adversary, geared towards interacting with parties running $\pi^{\rho/\phi}$. We wish to construct an adversary $\mathcal{A}_\pi$ in the $\mathcal{F}$-hybrid model such that no $\mathcal{Z}$ will be able to tell whether it is interacting

with $\pi^{\rho/\phi}$ and $\mathcal{A}$ or with $\pi$ and $\mathcal{A}_\pi$. For this purpose, we are given an adversary (a *simulator*) $\mathcal{S}_\rho$ that works for a single execution of protocol $\rho$. Essentially, $\mathcal{A}_\pi$ will run a simulated instance of $\mathcal{A}$ and will follow the instructions of $\mathcal{A}$. The interaction of $\mathcal{A}$ with the various instances of $\rho$ will be simulated using multiple instances of $\mathcal{S}_\rho$. For this purpose, $\mathcal{A}_\pi$ will play the environment for the instances of $\mathcal{S}_\rho$. The ability of $\mathcal{A}_\pi$ to obtain timely information from the multiple instances of $\mathcal{S}_\rho$, by playing the environment for them, is at the crux of the proof.

The validity of the simulation is demonstrated via reduction to the validity of $\mathcal{S}_\rho$. Dealing with many instances of $\mathcal{S}_\rho$ running concurrently is done using a *hybrid argument,* which defines many hybrid executions, where in each hybrid execution a different number of instances of $\phi$ are replaced with instances of $\rho$. This hybrid argument is made possible by the fact that $\mathcal{S}_\rho$ must be defined independently of the environment (this is guaranteed by the order of quantifiers), thus it remains unchanged under the various "hybrid environments".

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., PPT environment and adversaries), the composition theorem does not hold in general for protocols $\phi$ which are not PPT. In particular, the theorem does not necessarily hold when $\phi$ is an ideal protocol for an ideal functionality which is not PPT. (See example within.)

# 3 First steps

This section defines some basic concepts that underlie our treatment. These include the underlying model of computation, multi-party protocols, and polynomially bounded interactive computation. While rather technical, these definitions provide the essential foundations for making the treatment rigorous. Also, while some of these definitions are standard, others are new and may be of interest independently of the rest of this work (see outline in Section 2.1).

We note that some seemingly small definitional choices taken here have non-trivial effects throughout this work. In particular, some of the choices made here are quite different than in previous versions of this work. We point these choices out as we go along.

## 3.1 Probability ensembles and indistinguishability

We review the definitions of probability ensembles and indistinguishability, restricted to the case of binary ensembles. A distribution ensemble $X = \{X(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ is an infinite set of probability distributions, where a distribution $X(k,a)$ is associated with each $k \in \mathbf{N}$ and $a \in \{0,1\}^*$. The ensembles considered in this work describe outputs of computations where the parameter $a$ represents input, and the parameter $k$ is taken to be the security parameter. Furthermore, the ensembles in this work are binary, i.e. they only contain distributions over $\{0,1\}$.

**Definition 1** *Two* binary *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 $a \in \cup_{\kappa\le k^d}\{0,1\}^\kappa$ we have:*

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

We remark that Definition 1 is somewhat more relaxed than the corresponding definition in [C00] and in previous versions of this work: Here we only consider the distributions $X(k,a)$ and $Y(k,a)$ when the length of $a$ is polynomial in $k$; in contrast, previous formulations considered all lengths of $a$. This relaxation is necessary, since here we model polynomial-time Turing machines

somewhat differently than previously. Specifically, here the running time is measured as a function also of the input length, rather than as a function of the security parameter alone (as was done before). More details on, and motivation for, this change in convention appear later in this section.

## 3.2 The basic model of computation

We define the underlying model of computation. See additional motivational discussion in Section 3.4.

**Interactive Turing machines (ITMs).** We adhere to the standard formalization of algorithms (or, rather, computer programs) in a communication network as interactive Turing machines [GMRa89]. A detailed exposition of interactive Turing machines, geared towards formalizing *pairs* of interacting machines, appears in [G01, Vol I, Ch. 4.2.1]. The definition here is based on the definition there, adding some syntax that facilitates the modeling of multi-party, multi-execution settings where multiple (and potentially unboundedly many) interacting machines run in the same system. As usual, the definition contains details that are somewhat arbitrary, and do not add much insight into the nature of the problem. Nonetheless, fixing these details is essential for clarity and unambiguity of the treatment.

**Definition 2** *An* interactive Turing machine (ITM) *$M$ is a Turing machine with the following additional features. A tape of $M$ is called* externally writable (EW) *if it cannot be written to by $M$. (As seen below, it may be written to by other ITMs.) We assume that all externally writable tapes are* write-once, *in the sense that the writing head can only move in a single direction. This prevents ambiguities in the case where multiple ITMs write to the same tape. Then $M$ has the following tapes:*

- *An EW* identity *tape.*
- *An EW* security parameter *tape.*
- *An EW* input *tape and*
- *An EW* incoming communication *tape.*
- *An EW* subroutine output *tape.*
- *An* output *tape.*
- *A* random *tape.*
- *A read and write one-bit* activation *tape.*
- *A read and write work tape.*

*The contents of the identity tape of an ITM $M$ is called the* identity *of $M$. The identity of $M$ is interpreted, using some standard encoding, as two strings: the* session identity (SID) *of $M$ and the*

party identity (PID) *of $M$. The* extended identity *of $M$ is defined to be the identity of $M$, together with a description of the code (i.e., the transition function) of $M$.*[3]

*The contents of the incoming communication tape models information coming from the network. It is interpreted (using some standard encoding) to consist of a sequence of values called* messages, *where each message has two fields: the* sender *field, which is interpreted as the identity of some ITM, followed by an arbitrary* contents *field.*

*The contents of the subroutine output tape models the outputs of the subroutines of $M$. It is interpreted, using some standard encoding, to consist of a sequence of values called* subroutine outputs, *where each subroutine output has two fields: the* subroutine id *field, which is interpreted as the extended ID of some ITM, followed by an arbitrary* contents *field.*

*The code of an ITM $M$ may include instructions to write to a tape of another ITM. Such an* external-write *instruction specifies the following parameters: the extended identities of the present ITM and of a target ITM, the target tape to be written to, and the data to be written. The target tape may be either the input tape, or the incoming communication tape, or the subroutine output tape. The effect of an external-write instruction is defined below.*

*We say that $M$ is* active *if its activation tape is set to 1. Otherwise it is* inactive.

**Systems of ITMs.** We specify the basic mechanics of running multiple ITMs within a single system, and coin the relevant terminology. These definitions will be instrumental for defining the model of executing cryptographic protocols, in Section 4. They may also be of independent interest.

A system of ITMs $S = (I, C)$ is defined via an initial ITM $I$ and a control function $C : \{0,1\}^* \rightarrow \{allow, disallow\}$ that determines the effect of the external-write instructions of the ITMs in the system.[4] An execution of a system $S = (I, C)$, given security parameter $k$, input $x$, and random input $r$ to the initial ITM $I$, consists of a sequence of activations of instances (i.e., copies) of ITMs. In each activation the relevant instance $M$ becomes active, and proceeds according to its code until it enters one of two states: either a waiting state, or a halt state. If $M$ enters the waiting state then we say that the activation is complete and that $M$ is waiting for the next activation. If $M$ enters the halt state then we say that it has halted; in this case, it does nothing in all future activations. In either case $M$ becomes inactive, and another ITM is activated.

The first ITM to be activated is the initial ITM $I$, which starts with the input value $x$ written on its input tape, $1^k$ written on its security parameter tape, and with SID=PID=0. (We say that $k$ is the security parameter.) The execution ends when $I$ halts. The output of the execution is the contents of the output tape of $I$. To complete the definition of an execution, it remains to describe: (a) the effect of an external-write instruction, and (b) How to determine the next ITM to be activated, once an activation is completed. These are described next.

***Writing to a tape of another ITM and invoking ITMs.*** The effect of an external-write instruction, by an ITM $M$ with identity $id$ and code $c$ to an ITM $M'$ with identity $id'$ and code $c'$, is as follows.

[3] Jumping ahead, the SID will hold information that identifies the protocol instance that the present ITM is part of. the PID will identify the specific ITM within the protocol instance. It will often be convenient to have multiple ITMs that model programs running on the same physical computer to have related PIDs.

We use some standard encoding for representing the code of an ITM as string. Often the text does not distinguish between codes and the strings representing them. In fact, we can envision that the code of an ITM is written on its identity tape, next to the SID and PID.

[4] As seen below, $C$ can be viewed as a special-purpose ITM that implements the allowed operations defined in $C$.

1. If the control function $C$, applied to the sequence of external-write requests so far, does not allow $M$ to write to the specified tape of $M'$ (i.e., it returns a *disallow* value) then the instruction is ignored.

2. If $C$ allows the operation, and an ITM with identity $id$ (and any code) currently exists in the system, then the instruction is carried out. That is, the specified data is written on the specified tape of the target ITM, along with the identity of $M$. If the target tape is a subroutine output tape, then the *extended* identity of $M$ is written on that tape.[5]

3. If $C$ allows the operation, and no ITM with identity $id$ exists in the system, then a new ITM with code $c'$ and identity $id'$ is invoked. That is, a new ITM with the specified code is added to the system, the string $id'$ is written on its identity tape, the contents of the security parameter tape of $M$ is copied to that of the new ITM, and a infinitely long random string is written on its random input tape. Once the new ITM is invoked, the external-write instruction is carried out as above. In this case, we say that $M$ invoked $M'$.[6]

***Determining the order of activations.*** The order of activations is simple: In each activation, We allow an ITM to execute at most a single external-write instruction. The ITM whose tape was written to in an activation is activated next (i.e., its activation tape is set to 1). If no external-write operation was performed then the initial ITM $I$ is activated next.[7]

***Additional terminology.*** When $M$ writes a message $m$ to the incoming communication tape of $M'$, we say that $M$ sends $m$ to $M'$. When $M$ writes a value $x$ onto the input tape of $M'$, we say that $M$ passes input $x$ to $M'$. When $M'$ writes $x$ to the subroutine-output tape of $M$, we say that $M'$ passes output $x$ to $M$. We say that $M'$ is a subroutine of $M$ if $M$ has passed input to $M'$. (Note that $M'$ may be a subroutine of $M$ even when $M'$ was invoked by a different ITM.) $M'$ is a subsidiary of $M$ if $M'$ is a subroutine of $M$ or of another subsidiary of $M$.

***States and transcripts.*** A state of a system of ITMs represents a complete description of a certain instant in an execution of the system. Specifically, it consists of the local states of all ITMs in the system, plus the history of all the external-write requests (i.e., the inputs to $C$) in the execution so far. A transcript of an execution of a system is the sequence of states along the execution.

---

[5] This last provision represents the fact that the model allows an ITM to know the code run by its "subroutine machines". In contrast, an ITM does not necessarily know the code of the ITM that generated an incoming message. See more discussion in Section 3.4.

[6] Three remarks are in order here: First, the above invocation rules for ITMs, together with the fact that the execution starts with a single ITM, guarantee that each ITM in the system has unique identity. Second, there are no restrictions on the contents of the SID and PID. In particular, they can contain cryptographic keys or other identification information. Third, the code of the invoked ITM should be specified in the external-write instruction that caused the invocation. This means that the writing ITM determines this code, and this code exists as part of the state of the writing ITM prior to the invocation.

[7] Other orders of activation are of course possible (e.g., one can postulate that the ITMs are activated in "round robin" according to some pre-defined order). We fix the above ordering since it is simple and natural. In particular, it allows breaking down a distributed computation to a sequence of local events where at each point in time there is only a single ITM whose local state has changed since its last activation. While this simple ordering does not by itself preserve fairness or liveness, it allows us to represent these properties, among others, by defining special types of systems of ITMs (see subsequent sections).

**Extended systems.** An extended system is a system where the control function can also *modify* the external-write requests of ITMs. More precisely, 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 $C$. In this work we use this extra freedom only to modify the *codes* of ITMs. See more details in Section 4.

**Outputs of executions.** We use the following notation. Let $\text{OUT}_{I,C}(k, x)$ denote the random variable describing the output of the execution of the (possibly extended) system $(I, C)$ of ITMs when $I$'s input is $x$, and $I$'s security parameter is $k$. (Recall that the output of the system is the output of the initial ITM $I$) Here the probability is taken over the random choices of all the ITMs in the system. Let $\text{OUT}_{I,C}$ denote the ensemble $\text{OUT}_{I,C}(k, x)\}_{k\in\mathbf{N},x\in\{0,1\}^*}$.

**Multi-party protocols.** A multi-party protocol is defined as a (single) ITM as in Definition 2, representing the code to be run by each participant. Given a state of a system of ITMs, the instance of a multi-party protocol $\pi$ with SID *sid* is the set of ITMs in the system, whose code is $\pi$, and whose SID is *sid*. (Consequently, the PIDs of the ITMs in a protocol instance are necessarily distinct.) We assume that $\pi$ ignores all incoming messages where the sender SID is different than the local SID. Each ITM in an instance is called a party. A sub-party is a subroutine either of a party or of another sub-party. The extended instance of $\pi$ includes all the parties and sub-parties of this instance.

Two remarks are in order here. First, we differentiate between a *protocol,* which represents code to be run by each party, and a *protocol instance*, which represents a specific execution of a protocol. We also do not specify the number of parties in a protocol. Indeed, the model allows protocols where the number of participants is variable, dynamically changing, and even a-priori unbounded.[8]

Second, the convention of associating an SID with each protocol instance, where the SID is known to all parties, will prove very useful in our model where multiple protocol instances run concurrently in the same system. In addition, it seems to faithfully capture the practice of implementing protocols in actual computer systems (see further discussion in Section 3.4). We note that there exist other naming mechanisms for protocol instances, that do not require all parties to have exactly the same SID; still, the present convention is simple and natural, and the extra generality does not seem essential for our treatment.

We use several alternative naming methods for parties in a protocol instance. We let $\pi_i$ denote the $i$th party running protocol $\pi$, according to some arbitrary order (say, the order of invocation of the parties). The party need not know $i$ (i.e., $i$ is not explicitly written on any of $\pi_i$'s tapes). We let $\text{ID}(\pi_i)$ (resp., $\text{PID}(\pi_i)$, $\text{SID}(\pi_i)$) denote the identity (resp., PID, SID) of the ITM $\pi_i$. We also use $\pi_{(id)}$ to denote the party running $\pi$ whose extended identity is $(id, c)$. That is, $\text{ID}(\pi_{(id)}) = id$. When the protocol $\pi$ is understood from the context or is not specified we sometimes use the generic notation $P_i$ and $P_{(id)}$ instead of $\pi_i$ and $\pi_{(id)}$, respectively. Furthermore, when there is no danger of confusion we often write $P_i$ and mean $\text{ID}(P_i)$.

[8] Typically, different parties in a protocol play different roles and run different programs. This can be captured within the above single-ITM formalism by specifying in $\pi$ the programs for all participants. The reason for using this single-instance formalism is that it is natural in the case of protocols where the number of participants is unknown and potentially unbounded.

**Comparison to prior versions of this work.** The present definitions of ITMs, running a system of ITMs, and multi-party protocols are considerably more detailed, and somewhat different, than the corresponding definition in prior versions of this work. Let us highlight the main differences: (a) The partition of the ID to SID and PID did not explicitly exist before. The distinction between the identity of a protocol and the identities of parties within a protocol instance is very natural in the description and execution of protocols in a multi-party, multi-instance concurrent setting, and making it explicit seems helpful. (b) Before there was no formal distinction between a protocol and a protocol instance. In particular, the number of parties in an instance and their identities were assumed to be fixed. In addition, there was no formal distinction between $P_i$ and $\text{ID}(P_i)$, and between a party and a sub-party. (Protocols with a fixed number of parties and fixed identities can be obtained as a special case of the present formalism.) (c) No general simple rule regarding the order of activation of ITMs in a system was previously given. Instead, the ordering was more complex and model-specific. (d) Before, there was no clear distinction between transferring information via sending messages in a network, versus transferring information via local subroutine calls. In particular, the subroutine output tape, which models outputs from local subroutines, did not exist. (e) Before, the invocation of one ITM by another was not treated as explicitly as here. In particular, the need to specify the code of the invoked ITM was not explicitly addressed. (f) The requirement that a protocol ignores all incoming messages with an SID that is different than the local one was not explicitly made before (although it was used implicitly in a number of places).

### 3.3 Probabilistic polynomial time ITMs and systems

Throughout this work we concentrate on ITMs that operate in polynomial time. Rigorously defining resource-bounded computation in an interactive setting requires some care. This is especially so in our dynamic setting, where ITMs may be generated as the system evolves. To facilitate readability, we present the definition with only minimal discussion. Additional discussion is postponed to Section 3.4.3.

For concreteness and clarity, the definitions below are formulated specifically for probabilistic polynomial time (PPT) ITMs. We note, however, that they can be generalized in a straightforward way to handle any function family as a bound on the running time, instead of the family of polynomials. (See however the discussion prior to Proposition 16 on page 57.)

Our goal is to formulate a definition of polynomial time that matches our intuition in the most liberal way, and at the same time does not encumber the technical treatment with unnecessary details. In particular, we wish to adhere as much as possible to the standard notion that "a local computation is polynomial time if it takes a number of computational steps that is polynomial in the length of its input", and at the same time guarantee that the *overall* number of computational steps in an execution of a system is "polynomial".

Our starting point is the definition in [G01, Vol I, Ch. 4.2.1], which essentially says that an ITM $M$ is PPT if its total running time, in all its activations, is polynomial in the length of its input (i.e., in the number of bits written on its input tape). We extend this notion in a number of ways to fit our setting. First, we wish to bound not only the running time of $M$, but rather the running time of $M$ *together with all its subsidiaries.* To do that, we require that the number of bits written by $M$ onto the input tapes of other ITMs, plus the number of ITM instances whose tapes are written to by $M$, is at most the number of bits written on $M$'s input tape. (In a sense, this provision regards writing to the input tape of another ITM as giving some of the local "running time allowance" to that ITM. Adding the number of ITM instances whose tapes are written to bounds the overall number of new ITMs invoked by $M$. It also prevents unbounded sequences of

subroutine calls that preserve the input length.) Second, to account for the fact that an ITM may receive multiple inputs during its execution, we require that the above conditions hold at *any point* during the execution of $M$. This provision essentially allows an ITM to run indefinitely, as long as new values are written on its input tape. Third, we explicitly let the running time depend on the value of the security parameter, rather than assuming that the security parameter is part of the input. Finally, we require that all the subsidiaries of $M$ adhere to the same rules, and are bounded by a polynomial that is no larger than the polynomial bounding $M$.

**Definition 3 (locally PPT, PPT)** *Let $c \in \mathbf{N}$. An ITM $M$ is* locally PPT with exponent $c$ *if the following conditions hold for any prefix of any run of $M$:*

1. *The overall number of computational steps taken by $M$ so far is at most $n^c$, where $n$ is the security parameter plus the overall number of bits written so far on $M$'s input tape.*
2. *The number of bits written by $M$ so far to input tapes of ITM instances, plus the number of different ITM instances whose tapes are written to by $M$, is at most the overall number of bits written so far on $M$'s input tape.*

*If $M$ is locally PPT with exponent $c$, and in addition each subsidiary of $M$ is locally PPT with exponent $c' \leq c$, then we say that $M$ is* PPT with exponent $c$. *$M$ is* PPT *if there exists $c$ such that $M$ is PPT with exponent $c$.*

*A multiparty protocol is PPT if it is PPT as an ITM.*

For the purpose of modeling adversaries we will need another, more liberal variant of polynomial time, that allows an ITM to be polynomial in the overall number of bits written on *all* of its externally writable tapes. That is:

**Definition 4 (A-PPT)** *An ITM $M$ is* Adversary-PPT (A-PPT) *if there exists an exponent $c \in \mathbf{N}$ such that the overall number of computational steps taken by $M$ in any prefix of any run is at most $n^c$, where $n$ is the overall number of bits written so far on all of $M$'s externally writable tapes.*

It can be readily seen that the overall number of computational steps taken in an execution of a system where all ITMS are PPT is polynomial in the initial input of the system. Furthermore, this holds even if the system contains a single ITM instance that is only A-PPT, as long as this instance does not write to inputs tapes of other ITMs. In fact:

**Proposition 5** *If all the ITM instances in a system $(I, C)$ are PPT, perhaps with the exception of a single ITM instance that is A-PPT and does not write to input tapes of other instances, and in addition the control function $C$ is polytime computable, then an execution of the system can be simulated on a single PPT (non-interactive) Turing machine, which takes for input the initial input of the system.*

### 3.4 Discussion

Some aspects of the definition of ITMs and systems of ITMs were discussed in Section 2.1. Here we discuss some other aspect and motivate our definitional choices.

### 3.4.1 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 $\pi$-calculus of Milner, Parrow and Walker [MPW92], the *spi*-calculus of Abadi and Gordon [AG97] (that is based on $\pi$-calculus), the framework of Lincoln et. al. [LMMS98] (that is based on the functional representation of probabilistic polynomial time in [MMS98]), the I/O automata of Lynch [L96], the probabilistic I/O automata of Segala and Lynch [SL95] and more. Several reasons motivate the decision to use ITMs as a basis for our 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 at the heart of cryptographic protocols. In particular, ITMs allow the asymptotic treatment of security as a function of the security parameter. Furthermore, ITMs seem to best mesh with standard models used in complexity theory (such as standard Turing machines, oracle machines, and circuit families). Also, ITMs seem to faithfully represent the way in which existing computers operate in a network. Examples include the separation between communication and local inputs/outputs, the identities of parties, and the use of a small number of physical communication channels to interact with a large (and potentially unbounded) number of other parties (see also next remark). Finally, ITMs allow us to represent in a natural way a specific code to be executed, as opposed to merely functional specification of the result.

This last property may be regarded also as a disadvantage. Indeed, ITMs provide only 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].

That said, we note that the ITM model is in no way the only valid instantiation of the definitional approach presented here. Any other "reasonable" model that allows representing resource-bounded computation together with adversarially controlled communication would do. (See e.g. [DKMR05] as an example for such an alternative model). Still, care should be taken not to use overly abstract or restricted models that do not allow expressing realistic concerns.

***ITMs as circuit families and concrete security treatment.*** ITMs can be equivalently represented via circuit families. Here an evaluation of the circuit corresponds to a *single activation* of the corresponding ITM. That is, the input lines to the circuit representing an ITM $M$ consist (in some predefined way) of the internal state of $M$ at the beginning of an activation, plus the contents of the incoming communication tape and the subroutine output tape. The output lines of the circuit represent the internal state of $M$ at the end of the activation, plus the contents of the output tape, the outgoing communication tape, and the information written on the input tapes of the ITMs invoked by $M$. We usually think of ITMs as uniform-complexity ones, thus restricting attention to uniformly generated circuit families. However, non-uniform circuit families are possible as well (and are used to model adversarial entities).

We also note that, although the present treatment of complexity is asymptotic for sake of simplicity, a concrete-security treatment with precise parameterization can be derived from the present treatment in a straightforward way.

***Adversaries and Ideal Functionalities as ITMs.*** Although the main goal in defining ITMs is to represent programs run by the actual computers in a network, it will be convenient to use ITMs also to model more abstract entities such as adversaries, environments, and ideal functionalities. However, as will be seen, the model essentially allows the environment and adversary to be *non-uniform* PPT (by allowing the environment to have arbitrary input. See more details in Section 4.

### 3.4.2 On modeling systems of ITMs

Let us highlight the following points, in addition to the discussion in Section 2.1.

***Implicit invocation of new ITMs.*** Recall that the invocation of a new ITM instance is implicit, and occurs only when an existing ITM writes to a tape of a non-existing ITM instance. We adopt this convention since 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 ITM instance is invoked before it is first activated.

***On the global uniqueness of ITM identities.*** Recall that identities of ITMs are guaranteed to be *globally unique*. This is guaranteed via a simple mechanism: a new ITM instance with a given identity is invoked only if there is no other instance in the system with that identity.

Other methods of determining the identities of ITM instances are of course possible. For instance, an identity can be forced to be a pair (invoker ID, new ID). Here global uniqueness is guaranteed by the hierarchical encoding, so there is no need in the conditional generation method of new ITM instances. In addition, multiple ITM instances can have the same "local ID", while being distinguished via the IDs of their "ancestors" in the "ID tree". Still, we prefer the above non-hierarchical method since it is more general; in particular, the hierarchical method can be regarded as a special case of the method used here. (In particular, programs may not be aware of, or depend on, their "full ID", and may use only their "local IDs".)

***On the need to agree on the SIDs.*** Recall that the SIDs of ITMs in the same protocol instance must be identical. Since these SIDs are determined by the invoking ITM, it follows that there must be some coordination between the ITMs that invoke the various parties in a protocol instance, before the instance is invoked. This coordination may be provided by some prior agreement on these SIDs, or alternatively it may be obtained via some simple distributed protocol that provides agreement on the joint 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, plus some information that is locally unique to each participant.

Alternative conventions may only require that the SIDs of the parties in a protocol instance are related in some other way, rather than being equal. Such conventions may be able to avoid the need in a-priori coordination between the calling ITMs. Also, SIDs may be allowed to change during the course of the execution. We chose the present convention since it considerably simplifies the treatment throughout this work. Furthermore, the need in coordination among parties prior to engaging in a protocol execution is real, and appears (often implicitly) in realistic settings.

***Deleting ITM instances.*** The definition of a system of ITMs does not provide any means to "delete" an ITM instance from the system. That is, once an ITM instance 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 ITM is activated, it performs no operation and the initial ITM is activated next. The main reason for this convention is to avoid ambiguities in addressing of messages to ITM instances.

***On the distinction between input, communication, and subroutine tapes.*** In the case of ITMs defined for the purpose of interactive proof-systems [GMRa89, G01], an ITM has an input tape, output tape, an incoming communication tape and an outgoing communication tape. This distinction is very useful, since it allows separating the input/output functionality of a program from its communication with other parties; indeed, the communication with other parties is regarded as part of the workings of the protocol, rather than part of its functionality. In addition, since the communication tapes keep being overwritten throughout the computation, the distinction between input/output tapes and communication tapes facilitates modeling interactive algorithms that maintain state across messages.

We extend this approach by specifying two different methods of inter-ITM communication: communication using the usual communication tapes, and communication using the *subroutine output* tapes. Technically, the difference between the two is that, when an ITM instance writes into the subroutine output tape of another ITM instance, the *code* of the writing instance is ideally attached to the written value. The incoming communication tape provides no such guarantee. In addition, the model of computation, described in the next section, will allow the adversary to see and delay messages sent on the communication tapes, whereas the communication via the subroutine output tapes will remain secret and without delay.

This convention facilitates modeling multi-instance, multi-party computing environments. For instance, while the basic unit in a system of ITMs is a single ITM instance (which models a single execution of some program on some computing device), this convention allows delineating the trust boundaries around "physical computers" or other "trusted environments" that contain multiple ITM instances. That is, communication via the subroutine output tapes represents "trusted" communication that takes place, e.g., between a program and a subroutine of the program that runs on the same device. There is no question regarding how the received value was computed. Communication via the communication tapes represents information that "comes from the network" and is not trusted. In addition, this convention will facilitate the modeling of ideal protocols and ideal functionalities in the next section. Indeed, ideal functionalities will provide outputs to the parties via the subroutine output tapes of the latters.

Said differently, this convention allows abstracting out "details" such as the workings of the operating system or the communication stack running on a computer; at the same time, with a different set-up (e.g., a different control function), it allows explicit modeling of those "details."

Another, more technical difference between the modeling here and the one in [GMRa89, G01] is that here the parties don't have *outgoing* communication tapes. Instead, they write (if permitted by the control function) directly to the incoming communication tape of the recipient. Similarly, the parties make no use of their output tapes, and write outputs directly to the subroutine output tape of the recipient. (The one exception is the initial ITM, whose output is written on its own output tape.) This convention seems easier to work with in a multi-instance setting where the recipient identity is not fixed in advance.

***"Pseudo Concurrency" vs. "True Concurrency".*** The definition of systems of ITMs postulates a sequential execution of a system, in the sense that only a single ITM is active at any point during the execution. This stands in contrast to the physical nature of distributed systems where computations take place in multiple, physically separate places at the "same time". Furthermore, there exist mathematical models of distributed computation that directly model such "true concurrency" (see e.g. [L96, LMMS98]), via non-deterministic scheduling.

The main advantage of the present model is that it is relatively easy to argue about. In particular, it involves no non-determinism and can be simulated efficiently on a standard Turing machine. It is also readily amenable to inductive arguments, and allows modeling computationally bounded adversarial scheduling. Furthermore, we claim that, in spite of its inherent sequentiality, this "pseudo concurrent" execution model can provide arbitrarily close approximation to "true concurrency". Indeed, while no two ITMs can be active at the same time, the size of an "atomic sequential unit" can be made arbitrarily small. For instance, in the extreme case ITMs may enter the waiting state and pass control after each single invocation of the transition function.

***On the control function.*** The control function (i.e., the part of the system that sets the "rules of communication") is a powerful abstraction. As seen in the next section, it plays a central role in the security model and definition. That is, we specify the model by specifying an appropriate control function.

The fact that the control function is a separate entity than any other ITM instance provides a considerable amount of flexibility in capturing different security models. For instance, different control functions can be used to capture communication models providing different levels of liveness and fairness. They can also be used to capture models where the order of activations is different than here, such as [BPW04, DKMR05].

We remark that an alternative and equivalent description of a system of ITMs defines the control function via a special-purpose ITM $C$ that controls the flow of information between ITMs. Here the external input to the system is written to the input tape of $C$, and the security parameter is written to the security parameter tape of $C$. Once activated for the first time, $C$ writes its input to the input tape of $I$. From now on, all ITMs are allowed to write only to the incoming communication tape of $C$, and $C$ writes to externally writable tapes of all other ITMs. 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 change the recipient identity or code, according to the instruction in $C$'s code.

### 3.4.3 On defining PPT ITMs and systems

***Letting the runtime depend on the input length.*** Other notions of PPT ITMs, including those in [C00, C01, PW00, BPW04, BPW04a], restrict both the number of steps in each activation of a PPT ITM, and the overall number of activations, to be polynomial in the security parameter only, regardless of the length of the input. The present formalization is more permissive, thus it allows considering a larger class of protocols and adversaries. It is also compatible with standard modeling of cryptographic primitives such as encryption, signatures, or pseudorandom functions, where either the size or the number of inputs is determined by the adversary and is not bounded by any specific polynomial in the security parameter.

Another advantage of letting the runtime depend on the input length is that it allows proving the equivalence of some basic formulations of the main definition of security of protocols (see Section 4.4). In contrast, this equivalence does not hold with respect to the notions in [C00] and

in other works (see e.g. [HU05]).

Jumping ahead, we note that allowing the adversary to be A-PPT avoids some technical difficulties that occur when the adversary's input is short relative to the messages sent by the parties (see e.g. [DKMR05]).

***Using the control function to bound the running time.*** Another alternative approach to defining PPT systems of ITMs is to avoid making local restrictions on the running time of individual ITMs, and instead impose an overall bound on the runtime of the system. For instance, one can potentially use the control function to bound the runtime of a system. (Indeed, previous versions of this work used such a mechanism.) This approach is attractive since it is more general and considerably simpler. In particular, it is not sensitive to encodings of ITMs. We note however that this approach has the drawback that 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 ITM to be activated. This provides an "artificial" way for the last ITM to be activated (e.g., the initial ITM) to obtain global information on the execution. In particular, this notion of PPT systems can cause the notions of security defined in the rest of this work to be artificially restrictive.

***Recognizing PPT ITMs.*** One concern regarding notions of PPT Turing machines in general is that it may be impossible to decide whether a given ITM is PPT. The standard way of getting around this problem is to specify a set of rules on encodings of ITMs such that: (a) it is easy to verify whether the rules are obeyed by a given string (representing an encoding of an ITM), (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 3 is efficiently recognizable. Specifically, an encoding $\sigma$ of a *locally* PPT ITM will first specify an exponent $c$. It is then understood that the ITM encoded in $\sigma$ halts as soon as the overall number of steps taken by the ITM encoded in $\sigma$, or the number of bits written to input tapes of other ITMs, exceed their allowed values. An encoding of a PPT ITM will guarantee in addition that each subsidiary of the ITM encoded in $\sigma$ abides by the same encoding rules, with exponent $c' \leq c$. Note that these are syntactic conditions that are straightforward to verify.

***Thanks.*** We would like to thank Oded Goldreich, Dennis Hofheinz, Ralf Küsters, Yehuda Lindell, Jörn Mueller-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 a definition of protocols that securely realize a given ideal functionality, as outlined in Section 2. First we present (in Section 4.1) the basic computational model for executing distributed protocols. The model is defined in terms of a system of ITMs (see Section 3.2). It essentially captures a completely asynchronous network with unauthenticated and unreliable communication. Ideal functionalities and the ideal protocol for carrying out a given functionality are

presented in Section 4.2, followed by the general notion of protocol emulation and the definition of securely realizing an ideal functionality, in Section 4.3. Section 4.4 presents several alternative formalizations of the definition and demonstrates their equivalence to the main one. Finally, Section 4.5 defines hybrid protocols.

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

Given protocol $\pi$, adversary $\mathcal{A}$, and environment $\mathcal{Z}$, run an extended system of ITMs as specified in Section 3.2, with initial ITM $\mathcal{Z}$, and a control function as described below. $\mathcal{Z}$ starts with security parameter $k$ and input $z$. Next:

1. The first ITM invoked by $\mathcal{Z}$ is set to be the adversary, $\mathcal{A}$. All other ITMs invoked by $\mathcal{Z}$ are set to be ITMs that run $\pi$; furthermore, they are all required to have the same SID.
2. Once the adversary $\mathcal{A}$ is activated, it can pass output to $\mathcal{Z}$, and in addition it can perform one of the following activities:
   (a) $\mathcal{A}$ can deliver a message $m$ to a party or sub-party with identity $id$. There are no restrictions on the contents and sender identities of delivered messages (except for corruption messages, see next item).
   (b) $\mathcal{A}$ can corrupt an ITM $M$ with identity $id$. This operation is modeled as a special `(corrupt)` message delivered to $M$. This operation is allowed only once $\mathcal{A}$ receives an input `(corrupt` $id$`)` from $\mathcal{Z}$. The message may include additional parameters set by $\mathcal{A}$.
3. Once a party running $\pi$, or a sub-party thereof, is activated (either due to a new incoming message which was delivered by $\mathcal{A}$, or due to a new input from $\mathcal{Z}$, or due to a value written by a sub-party on the subroutine output tape), it can send messages to $\mathcal{A}$, it can pass outputs to the ITM that invoked it, and it can pass inputs to any subroutine ITM.

   The response to a `(corrupt)` message may vary according to the specific corruption model and the parameter values. In the Byzantine corruption model, the party reports its current state to the adversary. Furthermore, in all future activations by parties other than $\mathcal{A}$, the party sends to $\mathcal{A}$ its current local state. In all future activations the party follows the instructions of $\mathcal{A}$ regarding delivering values to other parties.

Figure 5: A summary of protocol execution

## 4.1 The model of protocol execution

The model for protocol execution is parameterized by three ITMs: the protocol $\pi$ to be executed, the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$. That is, given $\pi, \mathcal{Z}, \mathcal{A}$, the real-life model for executing $\pi$ is the PPT extended system of ITMs $(\mathcal{Z}, C^{\pi,\mathcal{A}}_{\text{EXEC}})$ (as defined in Section 3.2), where the initial ITM in the system is the environment $\mathcal{Z}$, and the control function $C^{\pi,\mathcal{A}}_{\text{EXEC}}$ is defined in the following paragraphs.

Recall that $\mathcal{Z}$ initially receives some input. This input 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 ITM to be invoked by $\mathcal{Z}$ is set by the control function to be $\mathcal{A}$. In addition, as the computation proceeds, $\mathcal{Z}$ may invoke as subroutines an unlimited number of ITMs, pass inputs to them, and obtain outputs from them, subject to the restriction that all the invoked ITMs have the same SID (which is chosen by $\mathcal{Z}$). The code of these ITMs is set by the control function to be the code of $\pi$. Consequently, all the

ITMs invoked by $\mathcal{Z}$, except for $\mathcal{A}$, are parties in a single instance of $\pi$.

The control function allows the parties and sub-parties of $\pi$ to invoke subroutines, to pass inputs to those subroutines, and to pass outputs to their invoking ITMs. In addition, they can send messages to the adversary $\mathcal{A}$, i.e. write messages on $\mathcal{A}$'s incoming communication tape. These messages may specify an identity of a party or sub-party of $\pi$ as the final destination of the message.

In addition, the control function allows the adversary $\mathcal{A}$ to send messages to any ITM in the system. 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 a party or sub-party of $\pi$.) There need not be any correspondence between the messages sent by the parties and the messages delivered by the adversary. $\mathcal{A}$ may not invoke any subroutines. Furthermore, all ITMs invoked by $\mathcal{A}$ (say, by delivering messages to them) must be PPT.

The adversary $\mathcal{A}$ may also *corrupt* parties or sub-parties of $\pi$. Formally, corruption of a party or a sub-party with identity *id* is modeled via a special `(corrupt)` message (potentially with additional parameters) delivered by $\mathcal{A}$ to that ITM. The control function allows delivery of that message only if $\mathcal{A}$ previously received a special `(corrupt` *id*`)` 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.) 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 6.6.

Still, for concreteness let us specify here one possible 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, $M$ follows $\mathcal{A}$'s instructions regarding external-writes to tapes of other ITMs.

We remark that the model allows $\mathcal{A}$ to invoke new ITMs by delivering messages to them. These ITMs may be either a party of the instance of $\pi$ with SID *sid*, or not. Allowing $\mathcal{A}$ to invoke parties of $\pi$ with SID *sid* is important since it allows modeling protocol instances that are invoked be a message coming from the network. One could potentially restrict $\mathcal{A}$ to invoking *only* parties of the current instance of $\pi$; but this would be cumbersome and not change the notion of security.

We restrict attention to environments that output a single bit. As discussed in Section 4.3, no generality is lost by this restriction. A summary of an execution of a multi-party protocol is described in Figure 5. See Figure 2 on page 19 for a graphical depiction[9]

We use the following notation. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ denote the random variable $\text{OUT}_{\mathcal{Z},C^{\pi,\mathcal{A}}_{\text{EXEC}}}(k,z)$. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

[9]There are four main differences between the model of computation here and in previous versions of this work: (1) Here the environment invokes parties and chooses their identities, whereas previously the set of participants and their identities were fixed in advance. (2) Here, once a party generates an outgoing message, $\mathcal{A}$ is the next to be activated. Previously, $\mathcal{Z}$ was activated next. This change does not affect the strength of the security definition, since $\mathcal{A}$ and $\mathcal{Z}$ can communicate freely throughout the computation. Its purpose is to clarify and simplify the presentation of the model. (3) Here, upon corruption, $\mathcal{A}$ learns only the *current* state of the corrupted ITM. In prior versions, $\mathcal{A}$ learned the entire sequence of prior states of the corrupted ITM. The reason for that stipulation in the prior versions was to reflect the concern that data erasures are not always effective. Still, the present formulation is more general, since it allows modeling both protocols where the security depends on effective data erasures, and also protocols that do not erase data at all. (4) Here the granularity of corruption is finer: $\mathcal{A}$ can corrupt also individual ITMs (i.e., parties or sub-parties). This finer granularity is natural in the present model; it also makes the model more expressive in terms of security requirements, and strengthens the universal composition theorem. (Of course, the traditional operation of corrupting a party and all its subsidiaries in "one shot" can be captured in the present formalization as a sequence of corruptions of the individual ITMs.)

## 4.2 Ideal functionalities and ideal protocols

**Ideal functionalities.** 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 2, with the following additional conventions. First, its input tape can be written to by a number of ITMs, and it can write to the subroutine output tapes multiple ITMs. This represents the fact that an ideal functionality behaves like a subroutine machine for a number of different ITMs (which are thought of as parties in a multi-party protocol). Next, the PID of an ideal functionality is set to $\perp$. (This fact is used to distinguish ideal functionalities from other ITMs.) In addition, an ideal functionality $\mathcal{F}$ expects all inputs to be written by ITMs whose SID is identical to the local SID of $\mathcal{F}$. Other inputs are ignored. The communication tape of $\mathcal{F}$ is used to communicate with the adversary. Typically, useful ideal functionalities will have some additional structure, such as 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 subsequent sections (see Section 6.1).

**Ideal protocols.** Let $\mathcal{F}$ be an ideal functionality. The ideal protocol for $\mathcal{F}$, denoted $\text{IDEAL}_{\mathcal{F}}$, is defined as follows. Whenever a party with identity $(sid, pid)$ is activated with input $v$, it writes $v$ onto the input tape of $\mathcal{F}_{(sid,\perp)}$, i.e. the instance of $\mathcal{F}$ whose SID is $sid$. (Recall that, according to the definition of a system of ITMs, $\mathcal{F}_{(sid,\perp)}$ is created at the first call to it.) Whenever the 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.[10] We sometime use the term dummy party for $\mathcal{F}$ to denote a party of an an ideal protocol for $\mathcal{F}$. See Figure 3 on page 21 for a graphic depiction of the ideal protocol.[11]

We use the following notation. Let $\text{IDEAL}_{\mathcal{F},\mathcal{A},\mathcal{Z}}(k, z, \vec{r})$ denote the random variable

[10] In the ideal protocol we assume that corruption messages are sent by the adversary directly to the ideal functionality. We then let the ideal functionality 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 when more than a certain number of parties have been corrupted, etc. See more discussion in Section 6.1.

[11] There are three main differences between the ideal protocol here and the "ideal process" in previous versions of this work. (I). Here the "ideal process" is presented as a specific protocol within the "real-life" model of computation, whereas previously it was presented as a separate process altogether. This difference is presentational only; indeed, while it does not change the definition of security, it simplifies the presentation considerably. (II). Here, $\mathcal{F}$ writes outputs directly on tapes of the recipient dummy party, and the recipient party is activated immediately, whereas previously it was up to the adversary to deliver messages from $\mathcal{F}$ to the dummy parties. This (quite radical) change has several advantages: (a) It better reflects the intuition that ideal functionalities are an idealization of local subroutine calls and should thus provide outputs directly to the parties via the subroutine output tapes, without intervention or knowledge of the adversary. (b) It simplifies the order of events in the ideal process and allows it to be captured as a special case of the main model of computation. (c) Most importantly, it allows capturing a number of different forms of communication (including asynchronous, synchronous, with or without guaranteed delivery, and local subroutine computation) within a single simple model. See more details in Section 6. Finally, we note that no generality is lost by this change, since an ideal functionality can always allow the adversary to delay messages, if it asks for the adversary's approval before actual delivery. (III). Previously, $\mathcal{F}$ was not notified upon corrupting parties or sub-parties, and the information learned by $\mathcal{S}$ was specified to be all the inputs and outputs obtained by the corrupted party so far. Here $\mathcal{F}$ is explicitly notified upon party or sub-party corruption, and the information obtained by the adversary upon party corruption is determined by the functionality. The present formulation allows more generality. For instance, it allows specifying requirements that depend on the identities of corrupted parties, as well as "forward security" style requirements where the adversary should not learn some internal data of a party even upon corrupting the party.

$\text{EXEC}_{\text{IDEAL}_{\mathcal{F}},\mathcal{A},\mathcal{Z}}(k,z,\vec{r})$. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

### 4.3 Definition of security

We first formalize the general notion of emulating one protocol via another protocol. Next, we use this general notion to define protocols that realize an ideal functionality. See discussion in Section 2.2. Essentially, protocol $\pi$ emulates protocol $\phi$ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that no environment can tell whether it is interacting with $\pi$ and $\mathcal{A}$ or with $\phi$ and $\mathcal{S}$:

**Definition 6** *Let $\pi$ and $\phi$ be PPT multi-party protocols. We say that $\pi$* UC-emulates *$\phi$ if for any PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any PPT environment $\mathcal{Z}$ we have:*

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

Protocols that realize an ideal functionality are defined as protocols that emulate the ideal protocol for this ideal functionality:[12]

**Definition 7** *Let $\mathcal{F}$ be an ideal functionality and let $\pi$ be an multi-party protocol. We say that $\pi$* UC realizes *$\mathcal{F}$ if $\pi$ emulates the ideal protocol for $\mathcal{F}$.*

***On statistical and perfect emulation.*** Definitions 6 and 7 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 $\pi$ statistically UC-emulates $\phi$. If in addition the two sides of (1) are required to be identical then we say that $\pi$ perfectly UC-emulates $\phi$. Another variant allows $\mathcal{S}$ 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 uniform-complexity inputs.*** Definitions 6 and 7 consider environments that take arbitrary input (of some polynomial length). This essentially makes the environment non-uniform from a complexity-theoretic point of view, since it is getting "advice"

[12] The present formulation of Definitions 6 and 7 is different from the formulation in prior versions of this work in a number of respects. First, as discussed earlier, the changes made to the model of computation and to the ideal protocol (or process) provide additional power to $\mathcal{F}$ and thus greater expressiveness in formulating ideal functionalities. It should be stressed however that these changes do not modify the basic notion of security, in the sense that the same security measures can still be captured (albeit with slightly different formulations of $\mathcal{F}$).

Second, here we separately and explicitly define protocol emulation, whereas previously this notion was implicit in the definition of security. This is a presentational change with no technical implications.

Third, the changes in the notions of indistinguishable ensembles and PPT ITMs (Definitions 1 and 3) make the present formulation somewhat more relaxed than before. That is, if a protocol securely realizes a functionality according to the previous formulation of the definition then it securely realizes the functionality also according to the current formulation of the definition. The other direction seems unlikely to hold. While the difference seems inconsequential in terms of "real world security" of protocols, it is nice to have a seemingly more relaxed formalization that still allows proving the composition theorem.

A fourth difference is that prior formulations restricted the real-life adversary to be part of a certain "class" of adversaries, where a class was interpreted as a limitation on the sets of parties that can be corrupted. Here we do not make this restriction; Indeed, since in the ideal process $\mathcal{F}$ learns the identities of the corrupted parties, the fact that security is guaranteed only with respect a certain corruption structure can be expressed within the specification of $\mathcal{F}$ rather than as a variation to the model.

that is not necessarily generated in polynomial-time. Alternatively, one may choose to consider only inputs that are in themselves the result of some uniform, polynomial time process. This weaker notion of security, which is often dubbed as "uniform-complexity security", can be captures by considering only environments that take no external input, and choose the inputs of the parties based on some internal stochastic process. (More formally, the external input to such environment contains no information other than its length, e.g. it is $1^n$ for some $n$.)

***On the transitivity of emulation.*** It is easy to see that if for some constant $n \in \mathbf{N}$ we have protocols $\pi_1, ..., \pi_n$ such that $\pi_i$ UC-emulates protocol $\pi_{i+1}$ for all $1 \leq i < n$, then $\pi_1$ UC-emulates $\pi_n$. In addition, it can be seen that transitivity extends to any polynomially long sequence of protocols, provided that there is a polynomial bound on the complexity of simulators for these protocols. (This is best seen using the equivalent notion of security via black-box simulation, see Section 4.4.) We remark though that the case of a non-constant number of different protocols seems to be of limited practical interest. 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.

## 4.4 Alternative formulations of the main definitions

We discuss some alternative formalizations of the definition of protocol emulation (Definition 6) and show that they are all equivalent to the main formalization.

**On environments with non-binary outputs.** Definition 6 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 6. We leave the proof as an exercise.

**On deterministic environments.** Since we allow the environment to receive an arbitrary external input, it suffices to consider only deterministic environments. That is, the definition that quantifies only over deterministic environments is equivalent to Definition 6. 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 $1^n$.

**Doing without the real-life adversary.** Definition 6 can be simplified as follows. Consider the following "dummy adversary", $\tilde{\mathcal{A}}$. When activated with an incoming message $m$ on its incoming communication tape, adversary $\tilde{\mathcal{A}}$ passes $m$ as output to $\mathcal{Z}$. 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, $\tilde{\mathcal{A}}$ delivers the message $m$ to the party whose identity is $id$. (Recall that the code $c$ is used in case that no party with identity $id$ exists; in this case a new party with code $c$ and identity $id$ is invoked as a result of this message delivery.) This in particular means that $\tilde{\mathcal{A}}$ corrupts parties when instructed by $\mathcal{Z}$, and passes all gathered information to $\mathcal{Z}$.

We show that, instead of quantifying over all possible adversaries $\mathcal{A}$, it suffices to require that the ideal-protocol adversary $\mathcal{S}$ be able to handle the dummy adversary $\tilde{\mathcal{A}}$ (and any environment machine $\mathcal{Z}$).

Before stating and proving this claim, we make an additional simplifying step in the definition of security against dummy adversaries. In this step we get rid of the dummy adversary altogether, and instead let $\mathcal{Z}$ interact directly with the parties. That is, in this simplified model parties and ideal functionalities send messages directly to $\mathcal{Z}$ rather than to $\tilde{\mathcal{A}}$. (Say, these messages are written to the incoming communication tape of $\mathcal{Z}$.) Similarly, $\mathcal{Z}$ writes directly to the incoming communication tapes of the parties and ideal functionalities. We use the shorthand $\text{EXEC}_{\pi,\mathcal{Z}}$ to denote $\text{EXEC}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}}$. We say that protocol $\pi$ UC-emulates protocol $\phi$ with respect to dummy adversaries if there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have $\text{IDEAL}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{Z}}$. We show:

**Claim 8** *Let $\pi, \phi$ be multiparty protocols. Then $\pi$ UC-emulates $\phi$ according to Definition 6 if and only if it UC-emulates $\phi$ with respect to dummy adversaries.*

**Proof:** Clearly if $\pi$ UC-emulates $\phi$ according to Definition 6 then it UC-emulates $\phi$ 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 simulate any adversary by itself. Thus quantifying over all environments essentially implies quantification also over all real-life adversaries. More precisely, let $\pi, \phi$ be protocols and let $\tilde{\mathcal{S}}$ be the adversary guaranteed by the definition of emulations with respect to dummy adversaries (that is, $\tilde{\mathcal{S}}$ satisfies $\text{IDEAL}_{\phi,\tilde{\mathcal{S}},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{Z}}$ for all $\mathcal{Z}$.) We show that $\pi$ UC-emulates $\phi$ according to Definition 6. 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 $\tilde{\mathcal{S}}$. In addition:

1. $\mathcal{S}$ forwards any input from the environment to the simulated $\mathcal{A}$, and copies any output of $\mathcal{A}$ to its own output tape (where it will be read by the environment).

2. Whenever the simulated $\tilde{\mathcal{S}}$ generates an output value $v$, $\mathcal{S}$ activates the simulated $\mathcal{A}$ with $v$ as incoming message. (Here $v$ is either a message sent by some party, or the internal state of a party upon corruption.) When the simulated $\mathcal{A}$ delivers a message $m$ to $P_{(id)}$, i.e. to the party with identity $id$, then $\mathcal{S}$ activates $\tilde{\mathcal{S}}$ with input $(m, id)$. (If $\mathcal{A}$ specifies the code $c$ of the recipient party, then so does $\mathcal{S}$.)

3. $\mathcal{S}$ forwards any message coming from parties running $\phi$ to the simulated $\tilde{\mathcal{S}}$, and forwards to these parties any message that $\tilde{\mathcal{S}}$ delivers.

Finally, to guarantee that $\mathcal{S}$ is A-PPT, we set a polynomial bound for the running time of $\mathcal{S}$. Let $p_\rho()$ be the polynomial bounding the overall number of communication bits sent by the instance of $\rho$ as a function of the overall length of inputs to parties of $\rho$. Then the polynomial $p()$ bounding the running time of $\mathcal{S}$ is $p_\rho()$ times the product of the polynomials bounding the running times of $\tilde{\mathcal{S}}$ and $\mathcal{A}$. (This bound is not tight; it is set for simplicity.) That is, $\mathcal{S}$ completes an activation as soon as the overall number of steps taken exceeds $p(n)$, where $n$ is the number of bits written so far on all of $\mathcal{S}$'s externally writable tapes.[13] A graphical depiction of the operation of $\mathcal{S}$ appears in Figure 6.

**Analysis of $\mathcal{S}$.** Say that an activation of $\mathcal{S}$ *ended prematurely* if it ended due to the fact that $\mathcal{S}$ exceeded its resources, before the simulation of the activation of either $\tilde{\mathcal{S}}$ or $\mathcal{A}$ completed. We

[13] We note that explicitly bounding the running time of $\mathcal{S}$ is necessary, since it simulates two interacting ITMs that are only A-PPT, rather than fully PPT. Thus, a-priori the interaction of the two ITMs may take a super-polynomial number of steps.

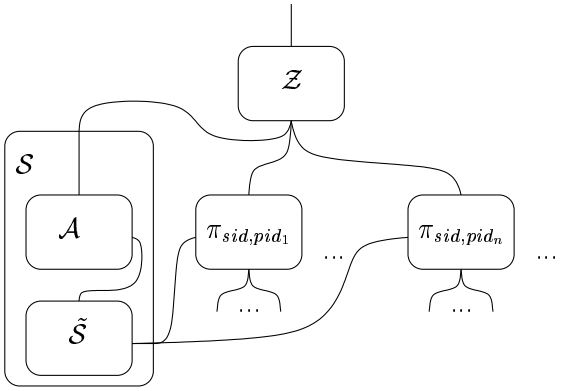


Figure 6: The operation of simulator $\mathcal{S}$ in the proof of Claim 8: Both $\mathcal{A}$ and $\tilde{\mathcal{S}}$ are simulated internally by $\mathcal{S}$. The same structure represents also the operation of the shell adversary in the definition of black-box simulation.

first analyze $\mathcal{S}$ assuming that no activation is ended prematurely. Next, we show that an activation ends prematurely with negligible probability.

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 $\tilde{\mathcal{Z}}$ such that $\text{EXEC}_{\pi,\tilde{\mathcal{S}},\tilde{\mathcal{Z}}} \not\approx \text{EXEC}_{\pi,\tilde{\mathcal{Z}}}$. Environment $\tilde{\mathcal{Z}}$ runs an interaction between simulated instances of $\mathcal{Z}$ and $\mathcal{A}$. Whenever $\tilde{\mathcal{Z}}$ is activated with a subroutine output value $v$ from its actual adversary, it passes $v$ to the simulated $\mathcal{A}$. Similarly, whenever the simulated $\mathcal{A}$ delivers a message $m$ to party $P_{(id)}$, $\tilde{\mathcal{Z}}$ delivers this message to $P_{(id)}$. In addition, $\tilde{\mathcal{Z}}$ relays all the communication from $\mathcal{Z}$ to $\mathcal{A}$ from $\mathcal{A}$ to $\mathcal{Z}$. It also relays all inputs from $\mathcal{Z}$ to the parties running $\pi$, and all the outputs from these parties to $\mathcal{Z}$. Finally, $\tilde{\mathcal{Z}}$ outputs whatever the simulated $\mathcal{Z}$ outputs. It can be readily verified that the ensembles $\text{EXEC}_{\phi,\tilde{\mathcal{S}},\tilde{\mathcal{Z}}}$ and $\text{EXEC}_{\phi,\mathcal{S},\tilde{\mathcal{Z}}}$ are identical. Similarly, ensembles $\text{EXEC}_{\pi,\tilde{\mathcal{Z}}}$ and $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ are identical.

It remains to bound the probability that an activation of $\mathcal{S}$ ends prematurely. First we note that, as long as the overall number of bits sent from the simulated $\tilde{\mathcal{S}}$ to the simulated $\mathcal{A}$ within $\mathcal{S}$ are less than $p_\rho(n)$, where $n$ is the overall number of bits that $\mathcal{Z}$ sends to the parties of $\phi$ in the execution so far, then no activation of $\mathcal{S}$ ends prematurely. To see this, recall that each activation of $\mathcal{S}$ consists of either a single activation of $\tilde{\mathcal{S}}$ followed by a single activation of $\mathcal{A}$ on data generated by $\tilde{\mathcal{S}}$, or vice versa. In either case, the activation takes less than the running time allotted to $\mathcal{S}$. Next, we argue that the overall number of bits sent from the simulated $\tilde{\mathcal{S}}$ to the simulated $\mathcal{A}$ within $\mathcal{S}$ exceeds $p_\rho(n)$ only with negligible probability. To see that, recall that the communication from the simulated $\tilde{\mathcal{S}}$ to $\mathcal{A}$ is distributed identically to the communication from $\tilde{\mathcal{S}}$ to $\tilde{\mathcal{Z}}$ in the interaction of $\tilde{\mathcal{Z}}$ with $\tilde{\mathcal{S}}$ and $\phi$. Consequently, this communication is indistinguishable from the communication generated by the parties running $\rho$ when interacting with $\tilde{\mathcal{Z}}$ in the dummy adversary model. Since the latter communication never exceeds $p_\rho(n)$ bits, it follows that that the communication from the simulated $\tilde{\mathcal{S}}$ to $\mathcal{A}$ exceeds $p_\rho(n)$ bits only with negligible probability. Also, the values of $n$ in the two executions are the same except for negligible probability. □

We remark that security with respect to dummy adversaries seems somewhat less intuitively appealing than Definition 7. In other words, we find it harder to get convinced that this definition captures the security requirements of a given task. In particular, its formulation is farther away from the formulation of the basic notion of security in, say, [C00]. Also, it is less obvious that this notion has essential properties such as transitivity. Therefore we did not present this definition as

the main one. Nonetheless, being considerably less complicated (it uses one less adversarial entity and one less quantifier), it is very useful. Indeed, we use it in most of the proofs in this work, including the proof of the UC theorem.

**On black box simulation.**[14] Another alternative formulation of Definition 6 imposes the following technical restriction on the simulator $\mathcal{S}$: Instead of allowing a different simulator for any adversary $\mathcal{A}$, we 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 specific 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.

The present formulation of black box emulation keeps the overall model of computation 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 A-PPT ITM $\tilde{\mathcal{S}}$ (called a black-box simulator) and a A-PPT adversary $\mathcal{A}$. $\mathcal{S}$ first internally invokes an instance of $\mathcal{A}$ and an instance of $\tilde{\mathcal{S}}$. Next:

- Inputs from the environment are passed through $\mathcal{A}$, then $\tilde{\mathcal{S}}$, then to $\phi$. That is, 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 $\tilde{\mathcal{S}}$. Instructions of $\tilde{\mathcal{S}}$ regarding delivering messages to the parties of $\phi$ are carried out.
- Incoming messages from $\phi$ are passed through $\tilde{\mathcal{S}}$, then $\mathcal{A}$, then to the environment. That is, upon receiving an incoming message from a party of $\phi$, $\mathcal{S}$ forwards this incoming message to $\tilde{\mathcal{S}}$. Outputs of $\tilde{\mathcal{S}}$ are forwarded as incoming messages to $\mathcal{A}$, and outputs of $\mathcal{A}$ are outputted by $\mathcal{S}$ to $\mathcal{Z}$.

In addition we require that $\mathcal{S}$ is A-PPT. That is, we require that there exists a polynomial $p()$ such that $\mathcal{S}$ ends an activation as soon as the overall number of steps taken by $\mathcal{S}$ exceeds $p(n)$, where $n$ is the number of bits written on $\mathcal{S}$'s externally writable tapes so far. (Note that $\mathcal{S}$ is not a priori guaranteed to be A-PPT since it is internally running two ITMs that are only A-PPT.) See graphical depiction of the operation of a black-box simulator in Figure 6.

Let $\text{EXEC}_{\phi,\mathcal{S}^{\tilde{\mathcal{S}},\mathcal{A}},\mathcal{Z}}$ denote the output of $\mathcal{Z}$ from an interaction with protocol $\phi$ and a shell adversary $\mathcal{S}$ that runs a black-box simulator $\tilde{\mathcal{S}}$ and an adversary $\mathcal{A}$. Say that a protocol $\pi$ UC-emulates protocol $\phi$ with black-box simulation if there exists an A-PPT shell simulator $\mathcal{S}$ and an A-PPT black-box simulator $\tilde{\mathcal{S}}$ such that for any A-PPT adversary $\mathcal{A}$, and any environment $\mathcal{Z}$, we have $\text{EXEC}_{\phi,\mathcal{S}^{\tilde{\mathcal{S}},\mathcal{A}},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. We show:

**Claim 9** *Let $\pi, \phi$ be PPT multiparty protocols. Then $\pi$ UC-emulates $\phi$ according to Definition 6 iff it UC-emulates $\phi$ 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 8 can be cast as a shell adversary with black-box simulator $\tilde{\mathcal{S}}$ and adversary $\mathcal{A}$. Furthermore, $\tilde{\mathcal{S}}$ does not depend on $\mathcal{A}$. □

We remark that the present formulation of security via black-box simulation is somewhat different than that of standard cryptographic modeling, where $S$ may query $\mathcal{A}$ in arbitrary ways. Here

[14] Thanks to Ralf Küsters for useful discussions on the formulation of black-box simulation within the UC framework.

the communication between $\tilde{\mathcal{S}}$ and $\mathcal{A}$ is much more restricted. In particular, $\tilde{\mathcal{S}}$ cannot "reset" or "rewind" $\mathcal{A}$. This difference makes the present formulation considerably *stronger* than the standard cryptographic formulation. Still, it is equivalent to the standard (non black-box) notion of security.

Two other, more technical differences are the introduction of the shell adversary $\mathcal{S}$, and the fact that $\mathcal{A}$ communicates directly with the environment, without the intervention of $\tilde{\mathcal{S}}$. The first difference allows us to stay within the model of protocol execution of Section 4.1, which postulates only a single adversary. The second difference allows us to have a *single* black-box simulator $\tilde{\mathcal{S}}$ that can handle all A-PPT adversaries. (In contrast, other notions of black-box simulation allow a different black-box simulator for each polynomial that bounds the running time of the adversary.)

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

**Letting the simulator depend on the environment.** Consider a variant of Definition 6, 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 (1). Following [L03a], we call this variant security with respect to specialized simulators. We demonstrate that this variant is equivalent to the main definition (Definition 6).

**Claim 10** *A protocol $\pi$ UC-emulates protocol $\phi$ according to Definition 6 if and only if it UC-emulates $\phi$ with respect to specialized simulators.*

**Proof:** Clearly, if $\pi$ UC-emulates $\phi$ as in Definition 6 then UC-emulates $\phi$ with respect to specialized simulators. To show the other direction, assume that $\pi$ C emulates $\phi$ 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 3, 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:

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

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 10 is an extension of the equivalence argument for the case of computationally unbounded environment and adversaries, discussed in [C00]. A crucial element in the proof of this

claim is the fact that we can have a PPT environment $\mathcal{Z}_u$ that is universal with respect to all PPT environments. This feature becomes possible only 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 restricted 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 could not be shown to be equivalent (see, e.g., [L03a]).

Finally we note that Claim 10 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 $n$.

### 4.5 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. More precisely, an $\mathcal{F}$-hybrid protocol $\pi$ 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 instance of $\pi$ has the freedom to specify the SIDs and PIDs for the different instances of $\text{IDEAL}_{\mathcal{F}}$ as it wishes. This way, different parties running $\pi$ can generate instances of $\text{IDEAL}_{\mathcal{F}}$ that have the same SID, and will thus interact with the same instance of $\mathcal{F}$. Furthermore, the generated SID and PID can be related in any way to the SID and PID of the calling instance. Recall also that by convention each instance of $\mathcal{F}$ interacts only with ITM instances that carry the same SID as its own. In particular, it does not interact with other instances of $\mathcal{F}$.

The behavior upon corruption is determined by the protocol, as usual. That is, when either an ITM running $\pi$ or the ideal functionality receives a `(corrupt)` message (typically with some additional parameters), it proceeds as specified in its code. Dummy parties ignore corruption messages.

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.[15]

[15] We highlight the main differences from the definition of the hybrid model in prior versions of this work. (a) Prior versions defined a separate "hybrid model of computation", whereas here we use the same basic model of computation and only define "hybrid protocols" within that model. This is a presentational change with no technical ramifications. (b) Prior versions include some additional syntactic conventions on the use of SIDs with ideal functionalities. Here these conventions are made already at the level of systems of ITMs and multi-party protocols (Section 3). (c) Previously the parties running $\pi$ called the instances of $\mathcal{F}$ directly, without the mediation of the dummy parties. Here the dummy parties are included in the hybrid model. (d) Previously an instance of $\mathcal{F}$ could send a message to the adversary, and request that the adversary sends a response message to that instance of $\mathcal{F}$ immediately (i.e., in the following activation). Here no such provision exists. This change has two simplifying effects on the model. First, it simplifies the algorithm for determining the order of activations, and makes it uniform over all models and types of protocols. Second, it allows extending Claim 8 to the case of hybrid protocols (see Section 4.4). Indeed, in previous versions of this work, Claim 8 did not extend to hybrid protocols. Furthermore, we note that no generality is lost by this change. Let us elaborate: The goal of allowing $\mathcal{F}$ to require immediate response by $\mathcal{A}$ was to allow $\mathcal{F}$ to obtain some information from the adversary, while still providing an output to some party in an immediate way (i.e., without allowing the adversary to activate the environment in the process.) In the present formulation, the same effect can be obtained by modifying $\mathcal{F}$ as follows. Consider a functionality $\mathcal{F}$ that sends messages to $\mathcal{A}$, and expects to obtain immediate response. Given $\mathcal{F}$, construct the functionality $\mathcal{F}'$ that is identical to $\mathcal{F}$ with the exception that $\mathcal{F}'$ obtains from the adversary the code of some ITM $\mathcal{A}'$ ($\mathcal{A}'$ is thought of as a "proxy adversary" to be run by the functionality). Furthermore, $\mathcal{F}'$ will accept messages from $\mathcal{A}$, updating the current state of $\mathcal{A}'$. Now, whenever $\mathcal{F}$

# 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. (This is in addition to the discussion in Section 2.3.)

## 5.1 The universal composition operation and theorem

While the main intended use of universal composition is for replacing the communication with an ideal functionality $\mathcal{F}$ with subroutine calls to 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 $\phi$, a protocol $\pi$ (that presumably makes subroutine calls to $\phi$), and a protocol $\rho$ (that presumably UC-emulates $\phi$), the composed protocol $\pi^{\rho/\phi} = \text{UC}(\pi, \rho, \phi)$ is identical to protocol $\pi$, with the following modifications.

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

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

When protocol $\phi$ is the ideal protocol $\text{IDEAL}_{\mathcal{F}}$ for some ideal functionality $\mathcal{F}$, we denote the composed protocol by $\pi^{\rho/\mathcal{F}}$. Also, when $\phi$ is understood from the context we use the shorthand $\pi^{\rho}$ instead. See a graphical depiction in Figure 4 on page 23.

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

Clearly, if protocols $\pi$, $\phi$, and $\rho$ are PPT then $\pi^{\rho/\phi}$ is PPT (with a exponent that is the maximum of the individual exponents).

**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 $\pi, \phi, \rho$ be three protocols, such that protocol $\rho$ UC-emulates protocol $\phi$ as in Definition 6. Then the protocol $\pi^{\rho/\phi} = \text{UC}(\pi, \rho, \phi)$ UC-emulates protocol $\pi$. 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. (As usual, protocol $\rho$ may in itself be a hybrid protocol, making ideal calls to some ideal functionality $\mathcal{E}$.)[16]

needs a value provided by $\mathcal{A}$, $\mathcal{F}'$ locally runs $\mathcal{A}'$ and treats the output as the message coming from $\mathcal{A}$. Note that $\mathcal{F}'$ needs no "immediate responses" from $\mathcal{A}$, and its effect is identical to that of $\mathcal{F}$ with immediate responses from $\mathcal{A}$.

[16] In previous versions of this work Theorem 11 was stated only for the case where protocol $\rho$ is not a hybrid protocol and $\phi$ is the ideal protocol for some ideal functionality. While the extension to the present formulation is

**Theorem 11 (Universal composition: General statement)** *Let $\pi, \rho, \phi$ be PPT multi-party protocols such that $\rho$ UC-emulates $\phi$. Then protocol $\pi^{\rho/\phi}$ UC-emulates protocol $\pi$.*

As a special case, we get:

**Corollary 12** *Let $\pi, \rho$ be PPT multi-party protocols such that $\rho$ UC-realizes a PPT ideal functionality $\mathcal{F}$. Then protocol $\pi^{\rho/\mathcal{F}}$ UC-emulates protocol $\pi$.*

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

**Corollary 13 (Universal composition: Realizing functionalities)** *Let $\mathcal{F}, \mathcal{G}$ be ideal functionalities such that $\mathcal{F}$ is PPT. Let $\pi$ be a multi-party protocol that UC-realizes $\mathcal{G}$, and let $\rho$ be a multi-party 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 11 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 $\pi$ 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 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 8). This formulation considerably simplifies the presentation of the proof. Let $\pi$, $\phi$ and $\rho$ be PPT multi-party protocols such that $\rho$ UC-emulates $\phi$, 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 $\pi^{\rho}$ (and no adversary) or with $\pi$ and $\mathcal{A}_{\pi}$. That is, for any $\mathcal{Z}$, $\mathcal{A}_{\pi}$ should satisfy

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

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

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

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).

straightforward, and the proofs are essentially identical, formally speaking the present formulation is more useful since most realistic protocols are in fact hybrid protocols with ideal access to some basic ideal functionalities (see e.g. Section 6).

Adversary $\mathcal{A}_\pi$ operates as follows. Recall that $\mathcal{Z}$ expects to interact with parties running $\pi^\rho$. 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 $\rho$ (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 $\pi$, $\mathcal{A}_\pi$ interacts directly with the actual, external parties running $\pi$. A bit more specifically, recall that $\mathcal{Z}$ receives the messages sent by the parties of $\pi$, by the parties of all instances of $\rho$, and by all their subsidiaries. In addition, $\mathcal{Z}$ delivers messages to all these entities. (These messages include also the corruption instructions and the subsequently revealed information.) $\mathcal{A}_\pi$ runs an instance of the simulator $\mathcal{S}$ for each instance of $\phi$ invoked by a party running $\pi$ in the system it interacts with. When activated with a message sent by a party or sub-party of $\pi$, $\mathcal{A}_\pi$ 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 $\rho$, 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 $\pi$ or a subroutine of $\pi$), 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 $\mathcal{A}_\pi$.

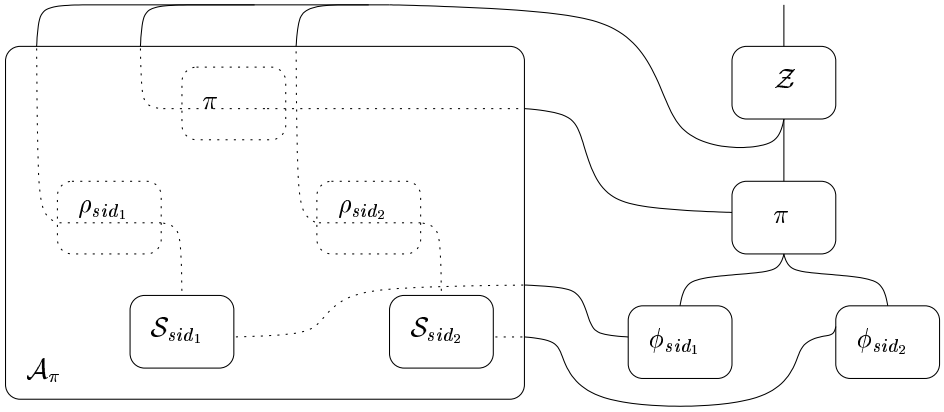


Figure 7: The operation of $\mathcal{A}_\pi$. Inputs from $\mathcal{Z}$ that represent messages of the instance of $\pi$ are forwarded to the actual instance of $\pi$. Inputs directed to an instance of $\rho$ are directed to the corresponding instance of $\mathcal{S}$. Messages from an instance of $\mathcal{S}$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

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 $\rho$ that are invoked in this interaction. Informally, for $l \leq m$ we let $\pi_l$ denote the protocol where the interaction with the first $l$ instances of $\phi$ remains unchanged, whereas the rest of the instances of $\phi$ are replaced with instances of $\rho$. In particular, protocol $\pi_m$ is essentially identical to protocol $\pi$. Similarly, protocol $\pi_0$ is essentially identical to protocol $\pi^\rho$.[17]

Now, assume that there exists an environment $\mathcal{Z}$ that distinguishes with probability $\epsilon$ between an interaction with $\mathcal{A}_\pi$ and $\pi$, and an interaction with $\pi^{\rho/\phi}$. Then there is an $0 < l \leq m$ such that $\mathcal{Z}$ distinguishes between an interaction with $\mathcal{A}_\pi$ and $\pi_l$, and an interaction with $\mathcal{A}_\pi$ and $\pi_{l-1}$. We

[17] 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) $l$th instance to be invoked. See details within.

then construct an environment $\mathcal{Z}_\rho$ that can distinguish with probability $\epsilon/m$ between an interaction with parties running a single instance of $\rho$, and an interaction with $\mathcal{S}$ and $\phi$. Essentially, $\mathcal{Z}_\rho$ runs a simulated execution of $\mathcal{Z}$, adversary $\mathcal{A}_\pi$, and parties running $\pi_l$, but with the following exception. $\mathcal{Z}_\rho$ uses its actual interaction (which is either with $\phi$ or with $\rho$) to replace the parts of the simulated execution that have to do with the interaction with the $l$th instance of $\phi$, denoted $\phi_l$. A bit more specifically, whenever some simulated party running $\pi$ passes an input $x$ to $\phi_l$, $\mathcal{Z}_\rho$ passes input $x$ to the corresponding actual party. Outputs generated by an actual party running $\rho$ are treated like outputs from $\phi_l$ to the corresponding simulated party running $\pi$. Furthermore, whenever the simulated adversary $\mathcal{A}_\pi$ passes input value $v$ to the instance of $\mathcal{S}$ that corresponds to $\phi_l$, $\mathcal{Z}_\rho$ 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 $\mathcal{Z}_\rho$.

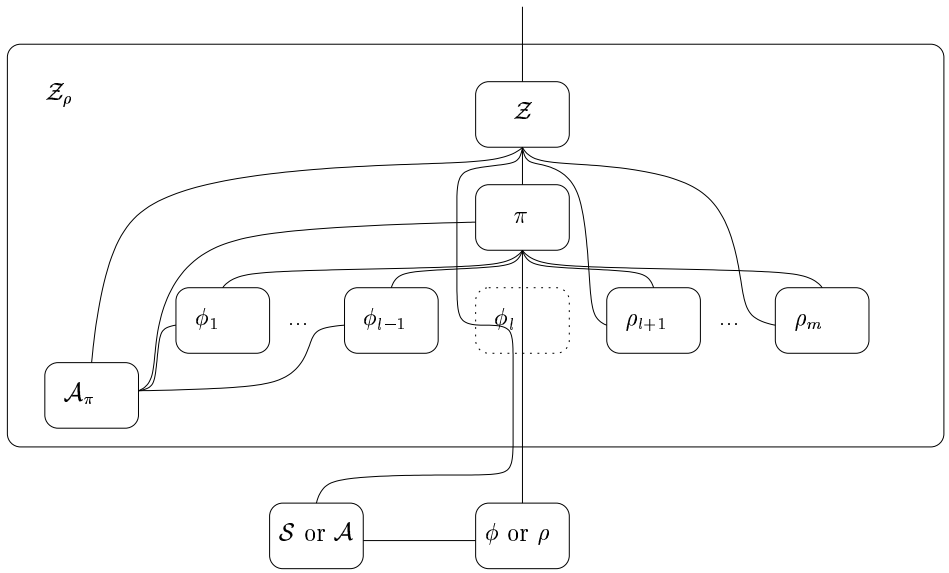


Figure 8: The operation of $\mathcal{Z}_\rho$. An interaction of $\mathcal{Z}$ with $\pi$ is simulated, so that the first $l-1$ instances of $\phi$ remain unchanged, the $l$th instance is mapped to the external execution, and the rest of the instances of $\phi$ are replaced by instances of $\rho$. For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

The proof is completed by observing that, if $\mathcal{Z}_\rho$ interacts with $\mathcal{S}$ and $\phi$, 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 $\pi_l$. Similarly, if $\mathcal{Z}_\rho$ interacts with parties running $\rho$, 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 $\pi_{l-1}$.

### 5.2.2 A detailed proof

We proceed with a detailed proof of Theorem 11, along the lines of the above outline. Most of the terminology and notations were defined in Section 3.

**Construction of $\mathcal{A}_\pi$.** Let $\pi, \phi, \rho$ be protocols, where $\rho$ emulates $\phi$, and and let $\pi^\rho = \pi^{\rho/\phi}$ be the composed protocol. The fact that $\rho$ UC-emulates $\phi$ guarantees that there exists an ideal-process

**Adversary $\mathcal{A}_\pi$**

Adversary $\mathcal{A}_\pi$ proceeds as follows, interacting with parties running protocol $\pi$ and environment $\mathcal{Z}$. Initially $\mathcal{A}_\pi$ keeps no instances of $\mathcal{S}$ running.

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 $c = \rho$ (i.e., the code $c$ is the code of protocol $\rho$), and there is no instance of $\mathcal{S}$ running internally with SID $sid$, then internally invoke a new instance of $\mathcal{S}$ with identity $(sid, \perp)$, activate this instance with input $(m, id, c)$, and follow its instructions. The new instance of $\mathcal{S}$ is denoted $\mathcal{S}_{(sid,\perp)}$.
   (b) Else, if there already exists an instance $\mathcal{S}_{(sid',\perp)}$ of $\mathcal{S}$, running internally, that handles the protocol instance associated with SID= $sid$ (that is, either $sid' = sid$ or $sid$ is the SID of a subsidiary of a party with SID=$sid'$), then activate $\mathcal{S}_{(sid',\perp)}$ with input $(m, id, c)$ and follow its instructions.
   (c) Otherwise, deliver the message $m$ to the party with identity $id$. (Using the terminology of Definition 2, this means that $\mathcal{A}_\pi$ executes an external-write request to the incoming message tape of a party with extended identity $(c, id)$.)
2. When activated with an incoming message $m$ from some party running $\pi$, do:
   (a) If $m$ was sent by some party $\phi_{(sid,pid)}$ of protocol $\phi$ then internally activate the instance $\mathcal{S}_{(sid,\perp)}$ 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 $\mathcal{S}_{(sid,\perp)}$.)
   (b) Else (i.e., if $m$ was sent by another party or sub-party $\pi_{(id)}$ of $\pi$, then pass $(m, id)$ as output to $\mathcal{Z}$.
3. When an instance $\mathcal{S}_{(id)}$ of $\mathcal{S}$ internally generates a request to deliver a message $m$ to some party running $\phi$, then deliver $m$ to this party. Similarly, when $\mathcal{S}_{(id)}$ requests to pass an output $v$ to its environment then pass $v$ to $\mathcal{Z}$.

Figure 9: The adversary for protocol $\pi$.

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 ITMs (parties and sub-parties of $\pi$ and $\rho$) 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 $\mathcal{A}_\pi$.** Clearly, $\mathcal{A}_\pi$ is A-PPT. (This is so since it runs at most a polynomial number of instances of $\mathcal{S}$, and the overall size of incoming data of each instance of $\mathcal{S}$ is polynomial in the size of incoming data of $\mathcal{A}_\pi$.) 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 $\rho$. (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

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

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 $\pi$ with adversary $\mathcal{A}_\pi$ and environment $\mathcal{Z}$. Let $t = t(k, |z|)$ be an upper bound on the number of instances of $\phi$ within $\pi$ 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 all ITMs considered are strongly polynomial.) For $0 \leq l \leq t$, Let the $l$-hybrid model for running protocol $\pi$ 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 ITMs of external-write requests.) The external-write requests to tapes of the first $l$ instances of $\phi$ to be invoked are treated as usual. The external-write requests to the tapes of all other instances of $\phi$ are directed to the corresponding instances of parties running $\rho$. That is, let $sid_i$ denote the SID of the $i$th instance of $\phi$ to be invoked in an execution. Then, given an external-write request made by an some ITM to $\phi_{(sid_i,pid)}$ (i.e., to the party running $\phi$ with identity $(sid_i, pid)$ for some $pid$, where $i > l$, the control function writes the requested value to the requested tape of $\rho_{(sid_i,pid)}$. (As usual, if no such ITM exists then one is invoked.) We let $\text{EXEC}^l_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ denote the output of this system of ITMs on input $z$ and security parameter $k$ for the environment $\mathcal{Z}$.

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. Recall that $\mathcal{A}_\pi$ expects its inputs to have the form $(m, id, c)$ (see Step 1 in Figure 9). $\hat{\mathcal{A}}_\pi$ 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 Step 1a in Figure 9. Let $\mathcal{Z}^{(l)}$ denote the environment machine that is identical to $\mathcal{Z}$, with the following exceptions. Whenever $\mathcal{Z}$ generates a message $(m, id, \rho)$ to be delivered to the party running $\rho$ 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 $\rho$, 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 $\pi$ 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 $\phi$ with which $\hat{\mathcal{A}}_\pi$ interacts.) The rest of the instances of $\mathcal{S}$ are replaced by interacting with the actual parties or sub-parties of the corresponding instances of $\rho$. Consequently, we have that the output of $\mathcal{Z}^{(t)}$ from an interaction with $\pi$ and $\hat{\mathcal{A}}_\pi$ in the $t$-hybrid model is distributed identically to the output of $\mathcal{Z}$ from an interaction with $\pi$ 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 $\pi$ and $\hat{\mathcal{A}}_\pi$ in the 0-hybrid model is distributed identically to the output of $\mathcal{Z}$ from an interaction with $\pi^\rho$ in the basic model of computation, i.e. $\text{EXEC}^0_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(0)}} = \text{EXEC}_{\pi,\mathcal{Z}}$. 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. Clearly, $\mathcal{Z}_\rho$ is PPT. This follows from the fact that the entire execution of the system is completed in polynomial number of steps. (Here we use the fact that the composed protocol, $\pi^\rho$, is PPT.)

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

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

**Environment $\mathcal{Z}_\rho$**

Environment $\mathcal{Z}_\rho$ proceeds as follows, given a value $k$ for the security parameter, input $z_\rho$, and expecting to interact with parties running a single instance of $\rho$. 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 $\pi$ 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 $\phi_l$ denote the $l$th instance of $\phi$ that is invoked in the simulated execution, and let $sid_l$ denote the SID of $\phi_l$.
   (a) Some simulated party passes input $x$ to a party with identity $(sid_l, pid)$. (That is, the recipient party participates in the $l$th instance of $\phi$.) 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 $z_\rho$ 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 $\pi$ 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 $\phi$ or a party running $\rho$.)
      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 $\rho$.

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

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

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 $\phi$ 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 $\pi$ and $\hat{\mathcal{A}}_\pi$ in the $l$-hybrid model. Similarly, if $\mathcal{Z}_\rho$ interacts with parties running $\rho$ 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 $\pi$ and $\hat{\mathcal{A}}_\pi$ in the $(l-1)$-hybrid model.

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

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

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 uniform-complexity inputs.** Recall that a uniform-complexity variant of the definition of emulation (Definition 6) 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 definition: the only difference 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 $1...m$.

**Composing multiple different protocols.** The composition theorem (Theorem 11) is stated only for the case of replacing instances of a *single* protocol $\phi$ with instances of another protocol. Yet the theorem holds, with essentially the same proof, also for the case where multiple different protocols $\phi_1, \phi_2, ...$ are replaced by protocols $\rho_1, \rho_2, ...$, respectively. Notice however that the more general formalization hardly adds any power to the model and the theorem, since one can always define a single protocol that mimics multiple different ones, say via a "universal protocol". In particular, in the special case where each $\phi_i$ is an ideal protocol for an ideal functionality $\mathcal{F}_i$, one can write a single ideal functionality that captures all the $\mathcal{F}_i's$.

**Nesting of protocol instances.** The universal composition operation can be applied repeatedly to handle multiple "nesting" of replacements of calls to sub-protocols with calls to other sub-protocols. We demonstrate that repeated applications of the composition operation maintains security. For instance, if a protocol $\rho_1$ UC-emulates protocol $\phi_1$, and protocol $\rho_2$ UC-emulates protocol $\phi_2$ using calls to $\phi_1$, then for any protocol $\pi$ that uses calls to $\phi_2$ it holds that the composed protocol $\pi^{(\rho_2^{\rho_1/\phi_1})/\phi_2} = \text{UC}(\pi, \text{UC}(\rho_2, \rho_1, \phi_1), \phi_2)$ UC-emulates $\pi$.

When the number of applications of the composition operation (i.e., the "depth of the nesting") is constant, the fact that the composed protocol UC-emulates the original one follows directly by repeated applications of the UC theorem. When the number of applications is not bounded by a constant (e.g., in the case of recursive calls to subroutines, where the "depth of the recursion" can depend on the input size or the security parameter), the implication no longer follows from the theorem statement alone, since the complexity of the simulator may not be polynomially bounded, and furthermore the distinguishing probability of the environment may not remain negligible. Still, we demonstrate that repeated applications of the UC operations preserves security as long as the number of applications (i.e., the "depth of the nesting") is polynomial in the security parameter,

and there exists a polynomial that bounds the complexity of all simulators. We note that essentially the same observation was already made in [BM04]. That is, we have:

**Corollary 14 (Universal composition: polynomial nesting)** *Let $t : \mathbf{N} \to \mathbf{N}$ be a polynomial. Let $\phi$ be a protocol, and let $\rho$ be a $\phi$-hybrid protocol that UC-emulates $\phi$. Let $\rho^{\langle t \rangle}$ be the protocol that, given security parameter $k$, is obtained by repeatedly applying the universal composition operation, starting with $\rho$, for $t(k)$ times. That is, $\rho^{\langle t \rangle}(k) = \text{UC}(\rho, ...\text{UC}(\rho, \rho, \phi)..., \phi)$. Then protocol $\rho^{\langle t \rangle}(k)$ UC-emulates $\rho$.*

For simplicity, Corollary 14 is stated for the case where the replaced protocols at all levels are the same (i.e., $\phi$), and the replacing protocols at all levels are the same (i.e., $\rho$). No generality is lost since the code of the protocol can specify different actions to different levels, or more generally may run code that is provided as part of the input. Another simplifying detail is that in protocol $\pi^{\langle t \rangle}$ the bound $t(k)$ is determined in advance, given the security parameter $k$. This avoids the case where different parties have different values for the "depth of nesting".

**Proof (sketch):** The proof is a straightforward extension of the proof of the UC theorem (Theorem 11). Here we only sketch these extensions. As there, we use security with respect to dummy adversaries (see Claim 8). That is, we construct an adversary $\mathcal{S}^*$ and show that $\text{EXEC}_{\rho,\mathcal{S}^*,\mathcal{Z}} \approx \text{EXEC}_{\rho^{\langle t \rangle},\mathcal{Z}}$ for all environments $\mathcal{Z}$.

Let $\mathcal{S}$ be the simulator, guaranteed from the fact that $\rho$ UC-emulates $\phi$, such that $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\rho,\mathcal{Z}}$ for all $\mathcal{Z}$. The instance of $\rho$ that's invoked by $\mathcal{Z}$ is said to be at depth 0. An instance of a protocol is at depth $i$ if it is invoked by an instance of a protocol at depth $i-1$. The construction of $\mathcal{S}^*$ is a natural extension of adversary $\mathcal{A}_\pi$ from the proof of Theorem 11, where the top level instance of $\rho$ plays the role of protocol $\pi$. That is, $\mathcal{S}^*$ internally runs an instance of $\mathcal{S}$ for each instance of $\phi$ that is replaced by an instance of $\rho$ in $\rho^{\langle t \rangle}$. Somewhat more precisely, it proceeds as follows.

1. Inputs from $\mathcal{Z}$ that contain messages to be sent to parties running the instance of $\rho$ of depth 0 are delivered as instructed. Similarly, messages sent by parties running this instance are forwarded to $\mathcal{Z}$.

2. Messages from $\mathcal{Z}$ to be sent to another instance of $\rho$ are forwarded to the corresponding instance of $\mathcal{S}$. Similarly, outputs generate by an instances of $\mathcal{S}$ that describe protocol messages sent by the parties running the corresponding instances of $\rho$ are forwarded to $\mathcal{Z}$.

3. Messages coming from an actual instance of $\phi$ are forwarded to the corresponding instance of $\mathcal{S}$. (These instances of $\phi$ are at depth 1.) Messages from an instance of $\mathcal{S}$ to an instance of $\phi$ at depth 1 are forwarded to that instance. (These are the instances of $\phi$ that exist in the actual run of $\rho$.)

4. Messages sent by an instance of $\mathcal{S}$ to the corresponding instance of $\phi$, where this instance is at depth greater than 1, are forwarded to the instance of $\mathcal{S}$ that corresponds to the instance of $\rho$ invoked this instance of $\phi$. (Notice that if the sending instance of $\mathcal{S}$ corresponds to an instance at depth $i$ then the recipient instance is of $\mathcal{S}$ corresponds to an instance at depth $i-1$.) Similarly, outputs of an instance of $\mathcal{S}$ (to be sent to the environment) that describe messages coming from an instance of $\phi$ are forwarded to the instance of $\mathcal{S}$ that simulates the instance of $\rho$ that replaces this instance of $\phi$. (Here, if the sending instance of $\mathcal{S}$ corresponds to an instance at depth $i$, then the recipient instance of $\mathcal{S}$ corresponds to an instance at depth $i+1$.)

Analysis of $\mathcal{S}^*$ is practically identical to the analysis of $\mathcal{A}_\pi$ at the proof of Theorem 11. First, we note that $\mathcal{S}^*$ is PPT. This is so since the overall number of instances of $\mathcal{S}$'s run by $\mathcal{S}^*$ is polynomial, each instance is PPT, and the input length of each instance is polynomial in the input length of $\mathcal{S}^*$. (Notice that there is no "nesting" in the calls to the instances of $\mathcal{S}$'s. That is, $\mathcal{S}^*$ directly calls all the instances of $\mathcal{S}$.) Now, assume that there is an environment $\mathcal{Z}$ such that $\text{EXEC}_{\rho^{\langle t \rangle},\mathcal{Z}} \not\approx \text{EXEC}_{\rho,\mathcal{S}^*,\mathcal{Z}}$. Then we define the hybrid environments $\mathcal{Z}^{(j)}$ and construct the distinguishing environment $\mathcal{Z}_\rho$ in exactly the same way as there. As there, the number of hybrids is the overall number of instances of all the instances of $\rho$. This number may be considerably larger than $t(k)$; still, it is polynomial. □

**On universal composition with joint state.** Informally speaking, the UC theorem implies that if a protocol $\phi$ UC-realizes some functionality $\mathcal{G}$ then multiple concurrent instances of $\phi$ UC realize multiple instances of $\mathcal{G}$. Consequently, instead of directly analyzing the security of the multi-instance system, it suffices to analyze the security of a single instance, and deduce the security of the multi-instance system from the UC theorem.

However this type of analysis is valid only when the uncorrupted instances of $\phi$ have mutually disjoint local states and local randomness. In contrast, in many cases we have a system where multiple concurrent instances of some protocol $\phi$ use the same instance of an underlying subroutine, $\rho$. Two common examples are (a) multiple instances of a pairwise key-exchange or a secure communication protocol, where all instances use the same instance of a long-term authentication mechanism (say, digital signatures, public-key encryption, or a pre-shared key), and (b) multiple instances of a protocol in the common reference string model (i.e., a protocol where the parties use a common source of randomness), where all instances use the same instance of the reference string. Here it is impossible to "de-compose" the system into protocol instances with disjoint local states; thus the UC theorem cannot be directly applied.

The Universal Composition with Joint State (JUC) theorem [CR03] provides a means to deduce the security of the multi-instance case from the security of a single instance, even when multiple instances use some joint state, or a joint subroutine. Informally, using this theorem we can deduce that at if we have a protocol $\gamma$ that UC-realizes functionality $\mathcal{G}$ using an ideal functionality $\mathcal{F}$, and a protocol $\rho$ that UC realizes, within a single instance, multiple instances of $\mathcal{F}$, then the protocol that consists of multiple concurrent instances of $\gamma$, where all the calls to $\mathcal{F}$ made by all instances of $\gamma$ are replaced by calls to a single instance of $\rho$, UC-realizes multiple instances of $\mathcal{G}$. See Figure 11 for a graphic depiction.

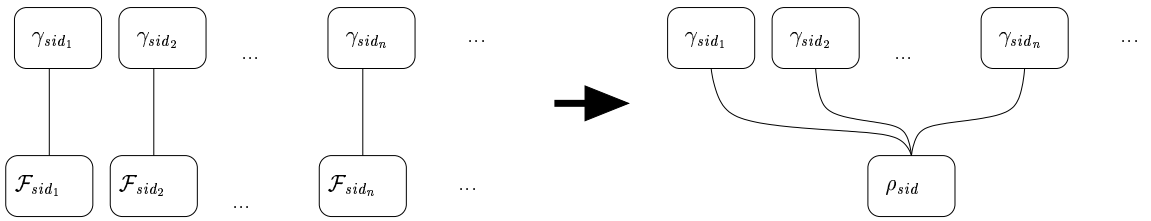


Figure 11: Universal composition with joint state. The instances of $\gamma$ (in the left figure) are analyzed assuming that each instance has access to a separate instance of $\mathcal{F}$. Later, all instances of $\mathcal{F}$ are replaced by a single instance of $\rho$ (right figure).

More rigorously, given an ideal functionality $\mathcal{F}$, let $\hat{\mathcal{F}}$, the multi-session extension of $\mathcal{F}$, be the ideal functionality that represents multiple independent instances of $\mathcal{F}$ within a single instance.

That is, $\hat{\mathcal{F}}$ expects to receive inputs of the form $(sid, ssid, v)$, where $sid$ is the SID of $\hat{\mathcal{F}}$, and $ssid$ (for "sub-session identifier") is an arbitrary string. Upon receiving such an input, $\hat{\mathcal{F}}$ internally invokes a instance of $\mathcal{F}$ whose SID is $ssid$, and forwards the input $(ssid, v)$ to that instance. (If such an instance already exists then $\hat{\mathcal{F}}$ simply forwards $(ssid, v)$ to it.) When an internal instance of $\mathcal{F}$ generates output $(ssid, v)$ to some party, $\hat{\mathcal{F}}$ generates output $(sid, ssid, v)$ to the same party.

Now, the universal composition with joint state operation, JUC, is defined as follows.[18] We start with an $\mathcal{F}$-hybrid protocol $\pi$, and a protocol $\rho$ that UC-realizes $\hat{\mathcal{F}}$. (Using the above terminology, protocol $\pi$ represents the multi-instance version of protocol $\gamma$.) Then the composed protocol $\pi^{[\rho/\mathcal{F}]} = \text{JUC}(\pi, \mathcal{F}, \rho)$ is identical to $\pi$ with the exception that, at the onset of the computation, $\pi^{[\rho/\mathcal{F}]}$ instructs each party to invoke a instance of $\rho$ with some arbitrary $sid$, say a fixed value. Then, each call $(ssid, v)$ to the instance $ssid$ of $\mathcal{F}$ is replaced with an input $(sid, ssid, v)$ to the instance $sid$ of $\rho$, and each output $(sid, ssid, v)$ of the instance $sid$ of $\rho$ is treated as a value $v$ received from instance $ssid$ of $\mathcal{F}$. Then, the JUC theorem states that protocol $\pi^{[\rho/\mathcal{F}]}$ UC-emulates protocol $\pi$. That is:

**Theorem 15 ([CR03])** *Let $\mathcal{F}$ be an ideal functionality, let $\pi$ be an $\mathcal{F}$-hybrid protocol, let $\rho$ be a protocol that UC-realizes $\hat{\mathcal{F}}$, and let $\pi^{[\rho/\mathcal{F}]} = \text{JUC}(\pi, \mathcal{F}, \rho)$ be the composed protocol. Then protocol $\pi^{[\rho/\mathcal{F}]}$ UC-emulates protocol $\pi$.*

Let us exemplify the use of the JUC theorem to analyzing a system that consists of multiple concurrent instances of a two-party key-exchange protocol, where each party uses a single instance of a signature scheme (i.e., a single signing key and a single verification key) to authenticate all the exchanges it participates in. We proceed in several steps: (a) Capture the functionality expected from a single instance of a key-exchange protocol via an ideal functionality, $\mathcal{F}_{\text{KE}}$. (b) Capture the behavior of a single instance of a signature scheme via a functionality, $\mathcal{F}_{\text{SIG}}$. (c) Construct a protocol, $\rho$, that realizes $\hat{\mathcal{F}}_{\text{SIG}}$, the multi-session extension of $\mathcal{F}_{\text{SIG}}$, using a single instance of an actual signature scheme. (d) Construct a protocol, $\gamma$, that realizes $\mathcal{F}_{\text{KE}}$ in the $\mathcal{F}_{\text{SIG}}$-hybrid model. It is stressed that each instance of protocol $\gamma$ realizes only a single exchange of a key, and is analyzed under the assumption that no other protocols exist; in particular, it is assumed that $\mathcal{F}_{\text{SIG}}$ is used by no other instance. This step is typically the main part of the analysis. (e) Let $\pi$ denote the protocol that consists of multiple independent instance of $\gamma$. Then use the JUC theorem to conclude that protocol $\pi^{[\rho/\mathcal{F}_{\text{SIG}}]}$ emulates protocol $\pi$. In particular, $\pi^{[\rho/\mathcal{F}_{\text{SIG}}]}$ exhibits the behavior of multiple independent instances of $\mathcal{F}_{\text{KE}}$, in spite of the fact that in $\pi^{[\rho/\mathcal{F}_{\text{SIG}}]}$ all instances of $\gamma$ within $\pi$ use the same joint instance of protocol $\rho$. (Said otherwise, notice that protocol $\pi$ in fact UC-realizes $\hat{\mathcal{F}}_{\text{KE}}$. It then follows from the JUC theorem that protocol $\pi^{[\rho/\mathcal{F}_{\text{SIG}}]}$ UC-realizes $\hat{\mathcal{F}}_{\text{KE}}$ as well. For more details about the application to key exchange protocols, and for other applications, see [CR03, CLOS02, C04].

**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.

That is, 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 $(I, C)$. Say

[18] For clarity, we present the JUC operation and theorem for the special case of replacing an ideal functionality with a protocol. As in the case of UC, the operation is defined for any two protocols.

that protocol $\pi$ UC-emulates protocol $\phi$ with respect to class $\mathcal{C}$ if Definition 6 holds when the class of PPT ITMs is replaced with class $\mathcal{C}$ (i.e., when $\pi$, $\mathcal{A}$, $\mathcal{S}$, 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 $\pi, \rho, \phi$ be multi-party protocols in $\mathcal{C}$ such that $\rho$ UC-emulates $\phi$ with respect to class $\mathcal{C}$. Then protocol $\pi^{\rho/\phi}$ UC-emulates protocol $\pi$ with respect to class $\mathcal{C}$.*

Furthermore, the UC theorem holds with respect to statistical and perfect emulation. That is, if $\rho$ statistically (resp., perfectly) UC-emulates $\phi$ then $\pi^{\rho/\phi}$ statistically (resp., perfectly) UC-emulates protocol $\pi$. (We remark that these statements do not follow from Proposition 16, since the notions of statistical and perfect emulation impose stricter requirements on the simulator than on the adversary.)

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 (more specifically, by an environment machine). We exemplify this point for the case where all entities in the system are bound to be PPT, except for the protocol $\phi$ which is not PPT. More specifically, we present an ideal functionality $\mathcal{F}$ that is not PPT, and a PPT protocol $\rho$ that UC-realizes $\mathcal{F}$ with respect to PPT environments. Then we present an $\mathcal{F}$-hybrid protocol $\pi$, that uses only *two* instances of the ideal protocol for $\mathcal{F}$, and such that $\pi^{\rho/\mathcal{F}}$ does not emulate $\pi$. In fact, for *any* PPT $\rho'$ we have that $\pi^{\rho'/\mathcal{F}}$ does not emulate $\pi$.

In order to define $\mathcal{F}$, we first recall the notion of pseudorandom evasive sets defined in [GK89] for a related purpose. (We use the notion of set ensembles, which is needed when dealing with non-uniform adversaries.) An ensemble $\mathcal{S} = \{S_k\}_{k\in\mathbf{N}}$ where each $S_k = \{s_{k,i}\}_{i\in[2^k]}$ 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 $i \in [2^k]$ we have that a random element $x \stackrel{R}{\leftarrow} s_{k,i}$ is computationally indistinguishable from $x \stackrel{R}{\leftarrow} \{0,1\}^k$. (b) $\mathcal{S}$ is evasive, that is for any PPT algorithm $A$ and for any $z \in \{0,1\}^*$, we have that $\text{Prob}_{i\stackrel{R}{\leftarrow}[2^k]}[A(z,i) \in s_{k,i}]$ 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. $\mathcal{F}$ interacts with one party only. Given security parameter $k$, it first chooses $i \stackrel{R}{\leftarrow} [2^k]$ and outputs $i$. Then, given an input $(x, i') \in \{0,1\}^k$, it first checks whether $x \in s_{k,i}$. If so, then it outputs `success`. Otherwise it outputs $r \stackrel{R}{\leftarrow} s_{k,i'}$.

Protocol $\rho$ for realizing $\mathcal{F}$ is simple: Given security parameter $k$ it outputs $i \stackrel{R}{\leftarrow} [2^k]$. Given an input $x \in \{0,1\}^k$, it outputs $r \stackrel{R}{\leftarrow} \{0,1\}^k$. It is easy to see that $\rho$ 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 $\rho$.

Now, consider the following $\mathcal{F}$-hybrid protocol $\pi$. $\pi$ 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 $x_1 \stackrel{R}{\leftarrow} \{0,1\}^k$, and feeds $(x_1, i_2)$ to $\mathcal{F}_1$. If $\mathcal{F}_1$ outputs `success` then $\pi$ outputs `success` and halts. Otherwise, $\pi$ feeds the value $x_2$ obtained from $\mathcal{F}_1$ to $\mathcal{F}_2$. If $\mathcal{F}_2$ outputs `success` then $\pi$ outputs `success`; otherwise it outputs `fail`. It is easy to see that $\pi$ always outputs success. However, $\pi^{\rho/\mathcal{F}}$ never outputs success. Furthermore, for *any* PPT protocol $\rho'$ that UC-realizes $\mathcal{F}$, we have that $\pi^{\rho'/\mathcal{F}}$ outputs `success` only with negligible probability.

We thank Manoj Prabhakaran and Amit Sahai for this example.

# 6 Capturing various computational models

Definition 7 captures one specific model of computation, which we call the bare model. Essentially, this model represents networks with unauthenticated, asynchronous, and unreliable communication, and an adversary that corrupts parties in an adaptive and Byzantine way throughout the computation. (See more discussion in Section 2.)

This is a very adversarial environment for designing protocols. We would of course like to be able to capture also a number of other models of computation, such as models that guarantee authenticated or secure communication, several levels of synchrony, or other abstractions. One way to do that would be to specify, for each such model or variant, an appropriately modified framework. However, re-defining the framework, and in particular re-proving the composition theorem for each variant is quite tedious, and also limits the number of options and model variants we can hope to specify.

Here we take an alternative approach, which makes strong use of the modularity of the framework: We demonstrate how several computational models of interest can be cast within the present model of computation, via making use of specific types of protocols. The main tool is to use hybrid protocols with access to ideal functionalities that capture the specific properties of the desired model. These ideal functionalities can be regarded either as set-up assumptions, or as a security goal for other "low level" protocols that are aimed at realizing these models. In addition, we specify different corruption models via different response modes to corruption requests from the adversary.

The main advantage of this approach is that it keeps the basic framework (i.e., the bare model) relatively simple and with few options, and at the same time avoids re-defining the framework and re-proving the composition theorem for each variant. In addition, this approach buys us greater flexibility in defining combinations and variants of models. For instance, we can model a system where different protocol instances use different levels of synchrony or different levels of authenticity, and we can define new variants by slight modifications to the relevant ideal functionalities.

We start with authenticated and secure communication. We then proceed to synchronous communication. Next we show how to capture within the present framework protocols that only guarantee security when run as "stand-alone", in the sense that no other protocol instance may be running concurrently to it. (This notion of security turns out to be essentially equivalent to the general security notion of [C00], which is easier to realize.) Finally, we discuss several models for the corruption of parties.

First, however, we set some writing conventions for ideal functionalities. These conventions will be used throughout the rest of this work.

## 6.1 Some 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.

**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 $P$) input from the adversary, $\mathcal{F}$ 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 $\mathcal{S}$, and receives a "modified input value" $v'$ from $\mathcal{S}$. Also, all output values intended for $P$ are sent to $\mathcal{S}$ instead. This behavior captures the standard behavior of the ideal process upon corruption of a party in existing definitional frameworks.[19]

**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 capture the fact that outputs generated by interactive protocols may be 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 $P$. (The value $v$ is not mentioned in this request. Furthermore, the request 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 request (by echoing the unique identifier), $\mathcal{F}$ outputs the value $v$ to $P$.

## 6.2 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 at the time when R receives the message.)

In the present formalization, protocols that make use of 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 12. $\mathcal{F}_{\text{AUTH}}$ first waits to receive an input $(\texttt{send}, sid, R, m)$ from a party with PID $S$ and SID $sid$. We require that the sender's PID is encoded also in the SID. (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 highlights the fact that each instance of $\mathcal{F}_{\text{AUTH}}$ is associated with a single, known sender.) $\mathcal{F}_{\text{AUTH}}$ then records $(S, R, m)$ and forwards this value to the adversary. Only when the adversary responds, with some input $(\texttt{send}, sid, R', m')$, does $\mathcal{F}_{\text{AUTH}}$ generate output: If the sender, $(sid, S)$, is uncorrupted *at this time* then $\mathcal{F}_{\text{AUTH}}$ delivers $(S, R', m')$ to $R'$ only if $R' = R$ and $m' = m$. Otherwise, $\mathcal{F}_{\text{AUTH}}$ puts no restrictions on $R'$ or $m'$.

Let us highlight several points regarding the formalization 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 us to concentrate on the simpler task of designing and analyzing protocols for authenticating only a single message, rather than dealing directly with a multi-message, multi-party protocol. Similarly, it allows higher-level protocols that use authenticated communication to be defined and analyzed for a single instance. Second, $\mathcal{F}_{\text{AUTH}}$

[19] 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.

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

1. Upon receiving an input $(\texttt{send}, sid, R, m)$ from party $S$, do: If $sid = (S, sid')$ then record $(R, m)$ and send $(\texttt{sent}, sid, R, m)$ to the adversary. Else, ignore this input.
2. Upon receiving $(\texttt{deliver}, sid, R', m')$ from the adversary, where $sid = (S, sid')$, do: If $(R', m')$ is recorded, or $S$ is corrupted, then output $(\texttt{sent}, sid, m')$ to party $R'$ and halt. Else halt.

Figure 12: The Message Authentication functionality, $\mathcal{F}_{\text{AUTH}}$

reveals the contents of the message to the adversary. This captures the fact that secrecy is not provided. Third, $\mathcal{F}_{\text{AUTH}}$ 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. Finally, $\mathcal{F}_{\text{AUTH}}$ 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.* This holds even if the sender was uncorrupted at the point when it sent the message.[20]

**On realizing $\mathcal{F}_{\text{AUTH}}$.** While it is impossible to UC-realize $\mathcal{F}_{\text{AUTH}}$ in the bare model (see proof in [C04]), there are a number of ways to realize $\mathcal{F}_{\text{AUTH}}$ given some standard set-up assumptions. Examples include ideally authenticated communication between parties at a preliminary stage, or alternatively trusted "registration services", where parties can register public values (e.g., keys) that can be authentically obtained by other parties upon request. A rigorous treatment of one particular way of realizing $\mathcal{F}_{\text{AUTH}}$ is given in [C04]. Specifically, it is shown there how to realize $\mathcal{F}_{\text{AUTH}}$ given any signature scheme that is secure against chosen message attacks (see [GMRa89, G01]), if an ideal "registration service" as described above is available. That is, we have:

**Claim 17** [C04] *If there exist signature schemes secure against chosen message attacks then $\mathcal{F}_{\text{AUTH}}$ can be realized by $\mathcal{F}_{\text{REG}}$-hybrid protocols.*

Here $\mathcal{F}_{\text{REG}}$ formalizes the registration service described above. We remark that protocols that UC-realize $\mathcal{F}_{\text{AUTH}}$ are closely related to the authenticators of [BCK98]. (A similar notion was used in [CHH00] in a different setting.) These are general "compilers" that transform any protocol that assumes ideally authenticated links into a protocol with essentially the same functionality in a model where the communication is unauthenticated. Specifically, applying an authenticator to a protocol $\pi$ (that assumes authenticated communication) corresponds to composing $\pi$ with a protocol $\rho$ that UC-realizes $\mathcal{F}_{\text{AUTH}}$. In fact, the protocol used in the proof of Claim 17 is essentially the signature-based authenticator of [BCK98], which has the sender sign the message together with a fresh "nonce" provided by the receiver. Here, however, since a different instance of the protocol is used per message, a new instance of a signature scheme is used to authenticate each message.

Finally, we mention two methods for realizing multiple instances of $\mathcal{F}_{\text{AUTH}}$ more efficiently than running multiple independent copies of a protocol that realizes a single instance of $\mathcal{F}_{\text{AUTH}}$. First, as demonstrated in [C04], it is possible to realize multiple instances of $\mathcal{F}_{\text{AUTH}}$ using a single instance

[20] Previous formulations of $\mathcal{F}_{\text{AUTH}}$ (e.g., [C01]) failed to let the adversary change the delivered message and recipient if the sender gets corrupted between sending and delivery. This resulted in an unnecessarily strong guarantee, that is in fact unrealizable by reasonable protocols. This oversight in the previous formulations was pointed out in several places, including [HMS03, AF04].

of a signature scheme per party. This method uses universal composition with joint state [CR03] in an essential way. Furthermore, when two parties engage in a "communication session" where they exchange multiple messages between them, they can use session authentication protocols, which employ symmetric message authentication mechanisms that are based on a secret shared key between the parties. See more details on modeling and analysis of such protocols in Section 7.1 and in [CK01, CK02].

### 6.3 Secure Communication

The abstraction of secure message transmission usually means that the transmission is ideally authenticated, and in addition that 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 some information on the domain from which the message is drawn). 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 13. 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 $m$, 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 $(\texttt{send}, sid, R, m)$ from party $S$, do: If $sid = (S, sid')$ then record $(R, m)$ and send $(\texttt{sent}, sid, R, l(m))$ to the adversary. Else, ignore this input.
2. Upon receiving $(\texttt{deliver}, sid, R', m')$ from the adversary, where $sid = (S, sid')$, do: If $R' = \perp$ and a record $(R, m)$ exists, then output $(\texttt{sent}, sid, m)$ to party $R$. Else if $S$ is corrupted then output $(\texttt{sent}, sid, m')$ to $R'$. In any case, halt.

Figure 13: 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.

**On preventing 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.) Schemes that hide the fact that a message was sent, (and are thus robust to traffic analysis) can be modeled by appropriate modifications to $\mathcal{F}_{\text{SMT}}$.

**On realizing $\mathcal{F}_{\text{SMT}}$.** Protocols that UC-realize $\mathcal{F}_{\text{SMT}}$ can be constructed, based on semantically secure public-key encryption schemes, 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 $\pi_E$. When invoked within $S$ with input $(\texttt{send}, sid, R, m)$ with $m \in D$, $\pi_E$ 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 $(sid, c)$, $\pi_E$ within $R$ outputs $dec(sk, c)$ and returns.

Choosing new keys for each message to be transmitted is of course highly inefficient and does not capture common practice for achieving secure communication. Nonetheless, it is easy to see that:

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

*Furthermore, if $E$ is non-committing (as in* [CFGN96]*) then $\pi_E$ UC-realizes $\mathcal{F}_{\text{SMT}}^D$ 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.*

Finally, 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 by using 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 e.g. [N02, CHK05].

In addition, to encrypt multiple messages exchanged between a pair of parties, have the parties generate a shared secret key and then use an appropriate symmetric encryption function (see e.g. [CK01, CK02]). Also here, security-preserving composition of such a protocol for encrypting multiple messages, with higher level protocols that assume multiple independent instances of $\mathcal{F}_{\text{SMT}}$, can be done using universal composition with joint state.

## 6.4 Synchronous Communication

A popular 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 specific and popular variant. Essentially, this variant of the synchronous communication model 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 $r - 1$.

Note that this guarantee implies in essence two other guarantees:

**Guaranteed delivery.** Each message sent by an uncorrupted party is guaranteed to arrive at the destination.

**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.*

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 14. 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 0. It then 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. If a party 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 this party at this round. In any case, once the round number advances, the set of messages sent at a 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 it at the previous round, together with the round number.[21] [22]

In order 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 $\pi$. Initially (e.g., when activated for the first time), each party of $\pi$ locally initializes a round counter to 0, and inputs to $\mathcal{F}_{\text{SYN}}$ a list $M$ of messages to be sent to the other parties of $\pi$. 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 $\pi$). 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 $\pi$ halts.

It can be seen that the message delivery pattern for such a protocol $\pi$ 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, a round 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

[21] 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.

[22] 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.

**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 $r \leftarrow 0$.
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 $r$ by all parties in $\mathcal{P}$, both corrupted and uncorrupted.
   (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 $L_P^{-1} = \perp$.)

Figure 14: The synchronous communication functionality, $\mathcal{F}_{\text{SYN}}$.

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 a round while the uncorrupted parties may have not received these messages. 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.

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 fast or arbitrarily slow. 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.

Finally 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.

**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. Alternative formulations of $\mathcal{F}_{\text{SYN}}$, which, say, allow the adversary to stop delivering messages or to modify messages sent by corrupted parties also in "mid-round", are of course possible.

Another potential relaxation of $\mathcal{F}_{\text{SYN}}$ is to reduce the "timeliness" guarantee. Specifically, it may only be guaranteed that messages are delivered within a given number, $\delta$, of rounds from the time they are generated. The bound $\delta$ 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 $\delta$ 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 such protocols, since this operation does not allow the composed protocols to have any joint subroutines. We note, however, that protocol copies that use the same instance of $\mathcal{F}_{\text{SYN}}$ can be composed using universal composition with joint state (see Section 5.3). That is, it is easy to see that multiple instances of $\mathcal{F}_{\text{SYN}}$ can be easily realized using a single "larger" instance of $\mathcal{F}_{\text{SYN}}$, whose set of participants is the union of the sets of participants of the realized instances. In other words, there exists a simple $\mathcal{F}_{\text{SYN}}$-hybrid protocol that UC-realizes $\hat{\mathcal{F}}_{\text{SYN}}$, the multi-session extension of $\mathcal{F}_{\text{SYN}}$, using a single instance of $\mathcal{F}_{\text{SYN}}$. (Intuitively, the point here is that having access to a globally synchronizing functionality provides a stronger guarantee than having access to multiple more local synchronizing functionalities.)

### 6.5 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 specify such "non-concurrent" security properties. Specifically, we present an ideal functionality, $\mathcal{F}_{\text{NC}}$, that guarantees that no other protocol instances are running concurrently to the calling instance. Thus, an $\mathcal{F}_{\text{NC}}$-hybrid protocol is essentially guaranteed to be running as "stand-alone" in the system. Functionality $\mathcal{F}_{\text{NC}}$ is presented in Figure 15. 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. (Figure 15 presents a variant of $\mathcal{F}_{\text{NC}}$ that emulates the bare, unauthenticated communication model. To capture non-concurrent security with, say, authenticated communication, $\mathcal{F}_{\text{NC}}$ needs to be modified somewhat so that only authentic communication can be delivered, mimicking the guarantees provided by $\mathcal{F}_{\text{AUTH}}$.)

One use for $\mathcal{F}_{\text{NC}}$ is to analyze systems where some of the components cannot run concurrently with other protocols, whereas the rest of the system components may be running concurrently with each other. Here the "non-composable" components can be written as $\mathcal{F}_{\text{NC}}$-hybrid protocols. For some concrete examples see Section 7.3.

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

**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, consider the natural generalization of the [C00] notion (which is stated for secure function evaluation in a synchronous communication networks) to the case of reactive functionalities and asynchronous networks. Recall that this generalized [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. That is:

**Definition 19** *Let $\pi$ and $\phi$ be PPT multi-party protocols. We say that $\pi$* NC-emulates *$\phi$ if for any PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any PPT environment $\mathcal{Z}$, that interacts with $\mathcal{A}$ in a restricted way as described above, we have $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.*

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 [G01]) 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.

We claim that having access to $\mathcal{F}_{\text{NC}}$ is essentially equivalent to running in the non-concurrent security model described above. More specifically, we claim that a protocol that sends all its messages via $\mathcal{F}_{\text{NC}}$ UC-realizes some ideal functionality $\mathcal{F}$ if and only if the protocol realizes $\mathcal{F}$ in the above non-concurrent security model. More generally:

**Proposition 20** *Let $\pi$ be an $\mathcal{F}_{\text{NC}}$-hybrid protocol that sends all its messages via $\mathcal{F}_{\text{NC}}$. Then $\pi$ UC-emulates protocol $\phi$ if and only if $\pi$ NC-emulates $\phi$.*

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

## 6.6 Various corruption models

The basic model of protocol execution does not fully specify the behavior of parties upon corruption. This highlights the fact that the composition operation and theorem apply to any type of behavior

upon corruption. Specifically, recall that party $P$ corruption 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 such a 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 concentrate on non-adaptive, passive (semi-honest), and proactive corruptions. While these corruption models are very different from each other in the security guarantees they provide, from a definitional point of view they are treated in similar ways, and are thus discussed together.

**Non-adaptive (static) party corruptions.** Definition 7 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 7 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.

In the present framework, 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. Both variants can be captured in a natural way via different 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. In the present framework, 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].

In the present framework, this type of attacks can be directly modeled via appropriate sets of reactions of parties to corruption requests. Security against such attacks is then defined as usual (i.e., by realizing an ideal functionality), with the exception that 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 an appropriate message to the adversary.

# 7 UC formulations of some primitives

This section demonstrates the general applicability of the framework by using it to provide universally composable definitions of a number of known cryptographic tasks. To do that, we formulate ideal functionalities that capture the security requirements of these tasks; a protocol is said to securely carry out a task if it securely realizes the corresponding ideal functionality as in Definition 7.

In most cases, the ideal functionalities here are formulated somewhat differently than the ones in the previous version of this work. This is due to two main factors. First, substantial progress was made on understanding and modeling some of these primitives since the first version of this work. In particular, a number of works have pointed out mistakes in the original formulations, as well as additional interesting variants of the corresponding primitives. Second, some changes are necessitated by the changes in the details of the overall framework. In all, this section can be regarded as a brief survey of current knowledge regarding UC formulations of the relevant primitives. It is stressed, though, that these formulations are not "set in stone"; they can (and should) be modified to fit different settings.

> The rest of this section is still under construction. It should be available shortly. Please check http://eprint.iacr.org/2000/067.

# 8 Future directions

The present work puts forth a general framework for defining and analyzing security or protocols. This may be regarded as a step towards putting the art of cryptographic protocol design on firm grounds. Let us briefly mention several current and future directions for furthering this goal and related ones. Some of these directions are the focus of current research reviewed in Section 1.4. Others are yet to be explored.

**Capturing and realizing cryptographic tasks and concerns.** In principle, the UC framework allows capturing practically any cryptographic task and concern. Some basic examples have

been given in the previous sections. Works that show how to capture other tasks, primitives, and concerns are reviewed in Section 1.4. Other tasks and concerns remain to be captured. A very incomplete list includes concerns such as anonymity; deniability and accountability; (in)coercibility; fairness; and tasks such as distributed ("threshold") versions of known primitives such as signature, encryption, zero-knowledge, etc.; agreement primitives such as broadcast and multicast with various levels of security.

**Finding better notions of security.** Another goal is to try to relax the requirements from protocols that realize a certain task, while maintaining "reasonable" security as well as strong composability properties. Indeed, the approach of defining security as the ability to "emulate" an ideal process has traditionally resulted in definitions that are more stringent than definitions based on other methods for defining security. Examples include Zero-Knowledge versus Witness-Indistinguishability of Interactive Proof-systems [GMRa89, FS90], and Key Exchange and Secure Session protocols [CK01, CK02]. The present framework is even more restrictive in that it gives more power to the adversarial environment. Indeed, the inability to demonstrate security of a given protocol within the present framework does not necessarily mean that the protocol is "bad" for its task. Formulating more relaxed notions of security is thus a natural and important goal.

Let us point to two alternative ways in which this can be done. A first direction may be to modify the framework itself (i.e., to relax Definition 7) in a way that maintains its main security and composability features, but is easier to realize. (Such an attempt was made in e.g. [PS04].) A second direction is to try to formulate ideal functionalities in a way that relaxes the requirements as much as possible. Examples of how this can be done include the treatment of digital signatures in [C04], Key-Exchange in [CK02], and commitments in [PS05].

A related question is to find characterizations of the functionalities that can be realized in certain settings. Some initial work was done in [CKL03], which concentrate on the task of two-party function evaluation given authenticated communication. Still, the question remains open in many other interesting settings.

**Formalizing and automating the analysis.** As discussed in Section 1.4, casting cryptographic analysis of protocols within a formal framework has many benefits, including eventual automation of parts of the analytical process. First steps towards this goal were described there. This is a promising research direction, with potentially far-reaching influence both on theoretical cryptography and on the security of actual computer systems. In particular, it may eventually enable cryptographic analysis of large-scale systems and networks.

**Exploring connections with game theory and mechanism design.** Similarly to cryptography, the disciplines of Game Theory and Mechanism Design study the interactions between mutually distrustful parties, which have potentially conflicting interests, but still wish to perform some joint computation. Some interesting connections between the theory of cryptographic protocols and game theory were explored in [DHR00, LMPS04]. The present framework may provide additional insight into these connections. In particular, it may help understand the compositional aspects of games and mechanisms.

**Extending to quantum computation and communication.** Designing cryptographic protocols and analyzing their security in a setting where all or some of the computing agents and

communication links exploit the effects of Quantum Physics is a non-trivial task that is quite different than its "classical" counterpart. Issues include delineating the borders between quantum and classical components, and compositionality of quantum adversaries. In particular, being able to define security against quantum adversaries that is preserved under protocol composition is a desirable goal. Current work towards extending the present framework to the quantum setting was described in Section 1.4. Future work includes extending the definitions of cryptographic tasks (such as commitment, coin-tossing, oblivious transfer, or zero-knowledge) to the quantum setting, and finding protocols for realizing them.

### 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, 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.

[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+04] B. Barak, R. Canetti, Y. Lindell, R. Pass, T. Rabin and A. Rosen. Secure Computation Without Authentication. Manuscript. 2004.

[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.

[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.

[BR93a] 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.

[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.

[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. (Earlier version of the present work.) Available at http://eccc.uni-trier.de/eccc-reports/2001/TR01-016/revision01.ps. Extended abstract in *42nd FOCS*, 2001.

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

[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.

[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.

[CGGM00] R. Canetti, O. Goldreich, S. Goldwasser, S. Micali. Resettable zero-knowledge. *32nd STOC,* pp. 235-244, 2000.

[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.

[CHKLM04] R. Canetti, S. Halevi, J. Katz, Y. Lindell, P. Mackenzie. Universally Composable Password-Based Key Exchange. Manuscript. 2004.

[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. *EUROCRYPT 2003,* pp. 68–86, 2003. Extended version at the eprint archive, eprint.iacr.org/2004/116.

[CKOR00] R. Canetti, E. Kushilevitz, R. Ostrovsky and A. Rosen. Randomness vs. Fault-Tolerance. *Journal of 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.

[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.

[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 Crecenzo, 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.

[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.

[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.

[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. *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) and Vol 2 (2004).

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

[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.

[HKN04] S. Halevi, P. Karger and D. Naor. A Cryptographic Model For Access Control. Manuscript. 2004.

[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),* 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

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

[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.

[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.

[L96] 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.

[L96] N. Lynch. *Distributed Algorithms.* Morgan Kaufman, San Francisco, 1996.

[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.

[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.

[MPW92] R. Milner, J. Parrow and D. Walker. A calculus of mobile processes, parts I and II. *Information and computation,* 1992. pp. 1-40 and 41-77.

[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.

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

[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.

[R+00] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe and B. Roscoe. *The Modelling 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.

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

[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 The main changes from the previous versions

**Changes from the version of October 2001.** The changes from this version (see [C01]) are detailed throughout the text. Here we provide a brief, high-level outline of the main changes:

**Non-technical changes:**

1. A more complete survey of related work (prior, concurrent, and subsequent) is added in Section 1.4.
2. The section on realizing general ideal functionalities (Section 7 in the October '01 version) is not included. It will be completed to contain a full proof and published separately.
3. Motivational discussion is added throughout.

**Technical changes:**

1. Extended and more detailed definitions for a "system of interacting ITMs" are added, handling dynamic generation and addressing of ITM instances in a multi-party, multi-protocol, multi-instance environment.
2. New notions of probabilistic polynomial time ITMs and systems are used. The new notions provide greater expressibility and generality. They also allow proving equivalence of several natural variants of the basic notion of security.
3. The model for protocol execution is restated in terms of a system of interacting ITMs. In particular, the order of activations is simplified.
4. The ideal process and the hybrid model are simplified and made more expressive. In particular, they are presented as special types of protocols within the general model of protocol execution.
5. The composition theorem is stated and proven in more general terms, considering the general case of replacing one subroutine protocol with another.
6. Various models of computation, including authenticated channels, secure channels, and synchronous communication, are captured as hybrid protocols that use appropriate ideal functionalities within the basic model of computation. This avoids defining extensions to the model to handle these cases, and in particular avoids the need to re-prove the UC theorem for these extended models. Various corruption models are also captured as special protocol instructions within the same model of execution.

**Changes from the version of January 6, 2005.** The main change from this version is in the definition of PPT ITMs and systems. Instead of globally bounding the running time of the system by a fixed polynomial, we provide a more "locally enforceable" characterization of PPT ITMs that guarantees that an execution of the system completes in polynomial time overall. See discussion in Section 3.4.3. These changes required updating the notion of black-box simulation and the proof of Claim 8, both in Section 4.4.