Electronic Colloquium on Computational Complexity, Revision 3 of Report No. 16 (2001)

Universally Composable Security: A New Paradigm for Cryptographic Protocols*

Ran Canetti†

December 14, 2005

Abstract

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

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

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

*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 January 2005 and 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. An extended abstract of this work appears in the proceedings of the 42nd Foundations of Computer Science conference, 2001.

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

ISSN 1433-8092

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. Furthermore, 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 the security of cryptographic protocols. Within this framework, we formulate a general methodology for expressing the security requirements of cryptographic tasks. Furthermore, we define (extending prior work) a very general method for composing protocols, and show that notions of security expressed within this framework preserve security under this composition operation. We call this composition operation universal composition and say that definitions of security in this framework (and the protocols that satisfy them) are universally composable (UC). Consequently, we dub this framework the UC security framework.[1]

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*

[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.

the task if running the protocol "emulates" the ideal process for the task, in the sense that any 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 standard model of protocol execution, captured 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 the adversary and all parties is called the *global output*. In the *ideal process* for evaluating some function f all parties ideally hand their inputs to an incorruptible *trusted party,* who computes the function values and hands them to the parties as specified. Here the adversary is limited to interacting with the trusted party in the name of the corrupted parties. Protocol π securely evaluates a function f if for any adversary $\mathcal{A}$ (that interacts with the protocol) there exists an ideal-process adversary $\mathcal{S}$ such that, for any set of inputs to the parties, the global output of running π with $\mathcal{A}$ is indistinguishable from the global output of the ideal process for f with adversary $\mathcal{S}$.

This definition suffices for capturing the security of protocols in a "stand-alone" setting where only a single protocol instance runs in isolation. Indeed, if π securely evaluates f then the parties running π are guaranteed to generate outputs that are indistinguishable from the values of f on the same inputs. Furthermore, any information gathered by an adversary that interacts with π is generatable by an adversary that only gets the inputs and outputs of the corrupted parties. (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 π securely evaluates a function f if for any adversary $\mathcal{A}$ there exists an "ideal adversary" $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with π and $\mathcal{A}$ or with $\mathcal{S}$ and the ideal process for f. (In fact, a similar notion of environment is already used in [c00] to capture non-concurrent composability for adaptive adversaries.)

The main difference between the UC framework and the basic framework of [c00] is in the way the environment interacts with the adversary. Specifically, in the UC framework the environment and the adversary are allowed to interact freely throughout the course of the computation. In particular, they can exchange information after each message or output generated by a party running the protocol. If protocol π securely realizes function f with respect to this type of "interactive environment" then we say that π UC-realizes f.

This seemingly small difference in the formulation of the computational models is in fact very significant. From a conceptual point of view, it represents the fact that "information flow" between the protocol instance under consideration and the rest of the network may happen at any time during the run of the protocol, rather than only at input or output events. Furthermore, at each point the information flow may be directed both "from the outside in" and "from the inside out". Modeling such information flow is essential for capturing the threats of a multi-instance concurrent 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 are generated throughout the computation, and may depend on previously generated values. This is obtained by replacing the "trusted party" in the ideal process for secure function evaluation with a general algorithmic entity called an ideal functionality. The ideal functionality, which is modeled as another ITM, repeatedly receives inputs from the parties and provides them with appropriate output values, while maintaining local state in between. This modeling guarantees that the outputs of the parties in the ideal process have the expected properties with respect to the inputs, even when new inputs are chosen adaptively based on previous outputs. We 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 weaker variant where the real-life adversary $\mathcal{A}$ is restricted to simply serve as a channel for relaying information between the environment and the protocol, or alternatively the variant where $\mathcal{S}$ may depend on the environment. It is also equivalent to the seemingly stronger variant where the ideal adversary $\mathcal{S}$ is restricted to black-box access to the adversary $\mathcal{A}$. (We remark that in other frameworks, including previous versions of this framework, these variants result in different formal requirements. See e.g. [hu05])

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 co-exist. 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 and may be determined dynamically depending on the execution. 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. (This composition operation can be regarded as a generalization of the "subroutine substitution" operation for sequential algorithms to the case of distributed protocols.) Let ρ be a protocol that UC-realizes some ideal functionality $\mathcal{F}$, according to the above definition. In addition, let π be some arbitrary protocol (we think of π as a "high level protocol") where, in addition to interacting in the usual way, the parties make ideal calls to multiple instances of $\mathcal{F}$. That is, protocol π 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 π is an $\mathcal{F}$-hybrid protocol. Here 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 π.

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

The universal composition theorem states that running protocol π^ρ, with no access to $\mathcal{F}$, has essentially the same effect as running the $\mathcal{F}$-hybrid protocol π. More precisely, it guarantees that for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{A}_\mathcal{F}$ such that no environment machine can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running π^ρ, or with $\mathcal{A}_\mathcal{F}$ and parties running π. In particular, if π securely realizes some ideal functionality $\mathcal{G}$ then π^ρ 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. 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 application is indeed very useful. Here, however, we propose a different interpretation 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 useful security guarantee for protocols that run in complex and unpredictable environments, such as modern communication networks.

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 towards defining various models of computation: We keep the basic framework simple and unchanged. Different communication models are captured by letting the parties access 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, as well as the Common Reference String model and several initial key registration models. (Essentially, an ideal functionality $\mathcal{F}$ captures a computational 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 formulate an ideal functionality that captures, within the present framework, security properties that are not necessarily preserved under concurrent 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.

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 and analyses. Security in a multi-party, multi-instance setting is guaranteed via the universal composition theorem.

Some of the more basic cryptographic tasks are best cast as realizing (the ideal functionality that represents) a communication model out of the ones mentioned in the previous section. Indeed, the present formalism does not distinguish between "realizing a model of computation" and "realizing a cryptographic task". Both goals are cast as "realizing an ideal functionality".

A first example of such a basic task is guaranteeing *message authentication,* which is formalized as realizing the ideal functionality that captures the authenticated communication model mentioned in the previous section. This functionality, denoted $\mathcal{F}_{\text{AUTH}}$, proceeds roughly as follows. It first expects to be invoked with a request by some party, S, to transmit a message m to another party, R. Then, it forwards (S, R, m) to the adversary, and waits for a response. Once the adversary returns an "ok" message, $\mathcal{F}_{\text{AUTH}}$ forwards (S, m) to R and halts. (If S 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 S then S 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 authenticating a *single message* between two parties. Protocols that use authenticated communication channels can then be modeled as $\mathcal{F}_{\text{AUTH}}$-hybrid protocols that use a new instance of $\mathcal{F}_{\text{AUTH}}$ for each message sent in an authenticated way.

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 communication sessions* and *key exchange.* The secure communication sessions functionality, $\mathcal{F}_{\text{SCS}}$, 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 $\mathcal{F}_{\text{SCS}}$ is that it allows for more efficient realizations, via key-exchange combined with symmetric cryptography using the generated keys. Indeed, $\mathcal{F}_{\text{SCS}}$ captures the basic security properties required from the many standardized and commercially available secure communication protocols. The key-exchange part of a secure communication session protocol is captured by the key exchange functionality, $\mathcal{F}_{\text{KE}}$, which essentially provides pairs of parties with ideally chosen random values.

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 Sections 1.4.3 and 1.4.5 for discussion on the realizability of these primitives. Finally, we formulate ideal functionalities that are aimed at capturing two basic multi-party tasks, namely verifiable secret sharing and secure function evaluation.

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. I apologize for any omissions and mis-representations. If you notice any, please let me 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. Work on the realizability of UC definitions of security and the minimality of the security requirements of the UC framework is reviewed next, 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.

1.4.1 Other simulation-based security frameworks

Two works that essentially "laid out the field" of general security definitions for cryptographic protocols are the work of Yao [Y82], which 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, the [MR91, B91] formalisms only address security in restricted settings; in particular, they do not deal with computational issues.) [GL90, MR91, B91] are surveyed in [C00] in more detail.

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 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] (called reactive simulatability) is similar to the one here. In particular, the role of their "honest user" can be roughly mapped to the role of the environment as defined here. However, there are several differences. First, they use a finite-state machine model of computation that builds on the I/O automata model of [Ly96] 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. They also postulate fixed *identities* of parties and protocol instances (sessions). Other technical differences include the notion of polynomial time computation, the order of activations, the scheduling of messages, and the generation of outputs. See [DKMR05] for a comparison of some 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 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 formulate a general security framework, it points out some important security concerns that arise when several cryptographic protocols run concurrently within a larger system. Making sure that the concerns pointed out in [DDN00] are addressed plays a central role in the present framework.

1.4.2 Connections with formal and automated protocol analysis

There are several well-known frameworks for formally analyzing properties of distributed systems and protocols. Examples include the CSP model of Hoare [H85], the π-calculus of Milner [M89, M99] and the I/O automata of Lynch [Ly96]. These frameworks and others are very attractive: they provide systematic methodologies of analyzing distributed systems, they are specifically geared towards analyzing the communication and concurrency aspects of protocols, they provide tools for modular design and analysis of protocols, and, most importantly, they naturally lend to mechanization and automation of the analysis. However, these frameworks 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.

Several efforts have been made towards extending these frameworks to capture randomized protocols and computationally bounded adversaries. Mitchell, Mitchell and Schedrov [MMS98] and Impagliazzo and Kapron [IK03] study ways for formally capturing polynomially bounded adversarial computation. Lincoln, Mitchell, Mitchell and Scedrov [LMMS98, LMMS99] introduce a variant of the π-calculus that incorporates random choices and computationally limitations on adversaries. In that setting, their definitional approach has a number of similarities to the simulation-based approach taken here: They define a computational 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.

Mateus, Mitchell and Scedrov [MMS03] and Datta, Küsters, Mitchell, and Ranamanathan [DKMR05] (see also [D05]) extend the [LMMS98, LMMS99] framework to express simulatability as defined here, cast in a polytime probabilistic process calculus, and demonstrate that the 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. Tight correspondence between the [MMS03] notion of security and the one defined here is demonstrated in Almansa [A04].

In a different thread of results, Lynch, Segala and Vaandrager [SL95, LSV03] extend I/O automata to handle probabilistic protocols. In Canetti et al. [CCKL+05] this model is further extended to capture resource bounded computation. In particular, they propose a definition of security similar to the one here, cast within their extension of the I/O automata model.

Another approach to addressing computational issues in formal security analysis of protocols was proposed somewhat implicitly in Needham and Schroeder [NS78] and more explicitly in Dolev and Yao [DY83]: Represent the cryptographic primitives in use as symbolic operations that provide absolute secrecy and authenticity guarantees regarding the transmitted data. Such modeling results in "idealized protocols" that use no randomness and whose security can be analyzed without having to model computationally bounded adversaries or computational assumptions. This line of modeling and analysis (which is often called the "Dolev-Yao modeling") was carried out in numerous works, including [BAN90, M94a, KMM94, LO96, AG97, AFG98, FHG98, R$^+$00, SO99, MP04]. It is attractive in that it allows for relatively simple analysis. More importantly, it is readily amenable to automation. Indeed, several tools that automate this style of analysis exist, e.g. [P88, SO99]. In all, Dolev-Yao style analysis was instrumental in finding flaws in many protocols, including widely used and standardized ones.

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

Abadi and Rogaway [AR00] proposes a methodology for bridging this analytical gap, by demonstrating Dolev-Yao style symbolic analysis that is also cryptographically sound. Specifically, they devise a formalism for arguing about "symbolic indistinguishability" of expressions based on symbolic encryption in an abstract and unconditional way. Furthermore, the following guarantee is given: If two symbolic expressions are 'symbolically indistinguishable' then the corresponding distribution ensembles, obtained by replacing the symbolic encryption by a semantically secure symmetric encryption scheme, are computationally indistinguishable in the standard sense. The Abadi-Rogaway 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 symbolic analysis is to interpret the Dolev-Yao symbolic operations as ideal functionalities in a composable security framework. When appropriately formulated, these ideal functionalities would allow the protocols that use them to be analyzable without having to address computational issues, and sometimes even deterministically. Cryptographic soundness would then follow from an appropriate composability property (such as universal composition). This approach was proposed in [PW00, C01]. In a sequence of works, including [BPW03, BPW03a, BP04], Backes, Pfitzmann and Waidner formulate, within their reactive simulatability framework, such an ideal functionality (which they call a "composable cryptographic library"), and show how certain security properties of protocols can be formally asserted using their ideal functionality. Still, in this approach, even the abstract, "symbolic" analysis has to be done within a full-fledged cryptographic framework. More discussion of the [BPW03] cryptographic library appears in [C04].

Canetti and Herzog [CH04] propose a different approach for carrying out computationally sound Dolev-Yao style symbolic analysis within a composable security framework. (This approach can be regarded as a "hybrid" between the above two approaches.) Specifically, they show how one can use symbolic and automated analysis to assert whether a protocol securely realizes an ideal functionality within the UC framework. More specifically, they concentrate on protocols for mutual authentication and key exchange, where the protocol has a specific format and the only cryptographic primitive is public-key encryption. They show how to translate such protocols into symbolic protocols in symbolic model (which is a variant of existing models, e.g. [AG97, FHG98]). Then, they write a symbolic mutual authentication criterion and a symbolic key exchange criterion, and show that the symbolic protocol satisfies the symbolic mutual authentication (resp., key exchange) criterion if and only if the original protocol UC-realizes a mutual authentication (resp., key exchange) ideal functionality. Verifying the symbolic criterion is considerably simpler than directly asserting UC security. The analysis is further simplified since only a single instance of the protocol needs to be analyzed; as usual, security of the multi-instance case is guaranteed by the UC theorem. Furthermore, they show how the symbolic criteria can be automatically verified using an existing tool, namely the tool of Blanchet [B04]. This allows to automatically assert whether a protocol of the appropriate format UC-realizes the mutual authentication or key exchange functionalities. Patil [P05] extends this work to handle protocols that use digital signatures in addition to public-key encryption.

1.4.3 On the realizability and minimality of UC definitions of security

Definitions of security in the proposed framework are often 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 composability.

We first note that realizing some ideal functionalities, such as functionalities for public-key encryption and signature, ends up being equivalent to traditional notions of security for these primitives. Other functionalities (e.g. key-exchange) are realizable by protocols that are commonly used in practice. See more details in Section 7.1. Furthermore, some known protocols for general secure function evaluation are, in fact, universally composable. For instance, the [BGW88] protocol (both with and 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, and authenticated and synchronous communication is available. Using [RB89], any corrupted minority is tolerable. Asynchronous communication 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 honest majority of the parties is not 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 unrealizability by plain protocols of almost all "non-trivial" deterministic two-party functions and many probabilistic two-party functions, and in Datta et al. [DDMRS06] to demonstrate impossibility of realizing *any* "ideal commitment functionality", namely any functionality that satisfies the basic correctness, binding and secrecy properties of commitment in a perfect way.

The unrealizability results imply, in particular, that none of the known two-party protocols for any of the above tasks is UC secure. One main reason for failure of the standard security analysis is 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. The reason for that is that in the present framework the ideal adversary has to interact with the environment which cannot be "rewound". (Indeed, it can be argued that the meaningfulness of black-box simulation with rewinding in a concurrent execution setting is questionable.)

Still, all is not lost: the commitment and zero-knowledge functionalities can be realized via protocols that use ideal access to the coin-tossing ideal functionality and authenticated communication [CF01, DN02, DG03]. (The ideal coin-tossing functionality lets all protocol participants output the value of a common random bit.) Interestingly, having access to the coin-tossing functionality turns out to be essentially a re-formulation 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 do without the common random string 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; the [DN00, DG03] construction is more efficient, but uses 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 a single trusted source of 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 that setting, dubbed the key set-up model, no single authority or string has to be completely trusted by all parties. This model can be viewed as a natural strengthening of the standard "public-key infrastructure" trust model, which is the basis for any large-scale authenticated communication.

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

1.4.4 Extensions and relaxations of the UC framework

The above results naturally bring up the question of whether one can relax the UC framework so that it will be possible to naturally express security requirements of general tasks in a way that is realizable by plain protocols (or easier to realize in general), but still guarantees reasonable composability and security.[2] Here, Lindell [L03a] 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.) In [L04] these negative results are shown to hold even for the case of self composition, where it is guaranteed that all protocol executions in the network are instances of the same protocol. These results stand in contrast with the general realizability results in [PR03, P04, PR05a, PR05b] for the cases of self composition when there is an a-priori known bound on the number of protocol instances running concurrently, or alternatively some "synchronizing event" that limits the concurrency.

Still, some meaningful relaxations of the UC framework exist. Prabhakaran and Sahai [PS04] propose a family of extensions of the UC framework; furthermore, they demonstrate a specific extended framework where security is preserved under universal composition, and where any functionality is realizable *in the plain model.* In accordance with Lindell's results, the [PS04] notion does not always imply even the basic notion of [C00, G01]. Nonetheless, [PS04] provides evidence that this notion is strong enough in a number of interesting cases.

Essentially, the idea in the [PS04] extension is to enhance the ideal-process adversary (namely, the simulator) by giving it access to super-polynomial computational resources (called "angels"),

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

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

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

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

Another meaningful relaxation of UC security is proposed in Lindell, Prabhakaran and Tauman [LPT04], by restricting the amount by which messages are delayed, along the lines of the "timing model" of [DNS98, G02]. That is, parties are assumed to have clocks with bounded drift, and message delays 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 other set-up assumptions, other than authenticated communication, are needed.

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 bounds in [L03, L04] indicate, this is in a sense the "basic definition" to use whenever one needs meaningful security that is preserved under universal composition.

Dodis et al. [DPW05] extend the UC framework to capture situations where the only available set-up is public and usable also by arbitrary other protocols, that were potentially designed in a poor or malicious way. This allows capturing concerns such as "deniability" of actions and "transferability" of proofs in the presence of a common public set-up. They also extend the general feasibility results of [CLOS02, BCNP04] to hold in their extended notion.

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

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

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

1.4.5 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. Some of these primitives are discussed in more detail in Section 7.

First we mention a useful tool for defining tasks and analyzing protocols, namely 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 protocol instances to be composed have some joint state and randomness. This enables modular analysis in many cases where we would otherwise be forced to analyze an entire complex system as a single atomic unit. Some examples appear below. (See more details in Section 5.3.)

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. [HMS03] point out some flaws in the [CK02] formulation of the ideal key exchange functionality, and demonstrate how these flaws can be fixed so that the security analyses of the protocols in [CK02] remain valid. The key exchange functionality in Section 7.1.1 takes these points into account.

In [CHKLM05] an ideal functionality capturing password-based key exchange is formulated and shown to be UC-realized by a variant of the [GL03] protocol. We remark that formulating an ideal functionality that adequately captures this task turns out to be non-trivial; in fact, 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 shown 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 in [C04] keeps to the approach of the original formulation and the equivalence with the [GMRi88] notion. It is also shown there 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. See more details in Section 7.2.1.

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

Nielsen [N02] demonstrates that realizing $\mathcal{F}_{\text{PKE}}$ in the presence of adaptive party corruptions is *impossible.* This holds even in the case where parties 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 during a given time window, and show how to realize this relaxed functionality under certain number-theoretic assumptions. A more detailed discussion appears in Section 7.2.2.

Two-party protocols. UC commitment protocols in the common reference string (CRS) model were constructed in [CF01, CLOS02, DN00, DG03]. The protocol of [CF01] is based on general assumptions (specifically, existence of claw-free trapdoor permutations) and is non-interactive, but is inefficient in that the size of the commitment string is roughly the security parameter times the length of the committed string. The scheme of [CLOS02] uses only standard trapdoor permutations. These schemes can be made secure against adaptive adversaries, assuming existence of enhanced trapdoor permutations, and can be made to use a uniformly distributed reference string assuming dense cryptosystems. The scheme of Damgaard and Nielsen [DN00] is interactive, uses reference string whose length is linear in the number of parties, but the size of the commitment is roughly the same as the size of the committed string. It is based on the Paillier assumption. Damgaard and Groth show how to make do with a reference string that is independent of the number of parties, while retaining the other efficiency parameters. Barak et. al. [BCNP04] show how to make the above schemes work in the key set-up model rather than the CRS model with the same hardness assumptions, in the presence of non-adaptive corruptions. Dodis et.al. [DPW05] extend these results to adaptive corruptions. Hofheinz et. al. construct a surprisingly simple UC commitment protocol in the random oracle model [HM04]. See more discussion in Section 7.3.1.

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

UC definitions of threshold key generation for the purpose of threshold signature and encryption schemes are formulated and realized in Abe and Fehr [AF04] and independently in Wikström [W04a].

1.5 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. Section 2.2, sketches the definition of protocol execution, the "ideal process" for realizing tasks, and the notion of security. Finally, Section 2.3 sketches the composition theorem and its proof. (These sections roughly correspond to the material in Sections 3, 4 and 5, respectively.) Throughout this section, 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

Before defining security of protocols, we should first formulate a reasonable model for representing distributed systems and protocols within them. Pinning down a model that is both technically workable and at the same time general enough to capture most realistic situations and concerns turns out to be non-trivial. Here we provide a high-level view of the devised model.

Our model extends the interactive Turing machine (ITM) model of [GMRa89, G01]. That is, the programs run by parties in a communication network are represented as Turing machines with some "shared tapes" which can be written into by one machine and read by another. 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 3.4.1.

Due to the relative complexity of modeling multi-party, multi-protocol, multi-instance distributed systems, we chose to define the underlying "mechanics" of inter-ITM communication separately from the security model and definition. For this purpose, we coin 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. Indeed, this notion may be of interest even regardless of the notions of security which are the focus of the rest of this work.

Before describing the operation of a system of ITMs, we formalize the notion of an instance of an ITM. In contrast with an ITM, which corresponds to a "static object", namely an algorithm or a program, an ITM instance (dubbed an ITI) is a "run-time object", namely an instance of a program running on some specific data. In particular, the same program (ITM) may have multiple instances (ITIs) in an execution of a system. (Conceptually, an ITI is closely related to a *process* in a process calculus. We refrain from using this term to avoid confusion with other formalisms.)

A system (I, C) of ITMs consists of an initial ITM M and a control function C. An execution of a system starts by invoking an instance of I (i.e., an ITI running the program I) on some external input. This ITI is called the initial ITI. In addition to its local computation, this ITI can *invoke* other ITIs and write information on some of their tapes. Once an ITI is invoked, it executes its code, and potentially invokes other ITIs or writes to tapes of other ITIs. Overall, an execution of a system consists of a sequence of activations, where in each activation a single ITI is active (i.e., running). In each activation, the active instance may write to the tapes of only one other ITI. Once this ITI enters a special "waiting" state, the ITI whose tapes were written to becomes active. If the tape of no other ITI is written to, then the initial ITI is activated. The control function C determines which tapes of which ITIs can be written to by each ITI. Figuratively, C can be viewed as a central "routing ITM" which forwards information from one ITI to another. An execution ends when the initial ITI halts. The output of an execution is the output of the initial ITI.

Note that the above extremely simple order of activations is in fact quite powerful, in that it allows us to express practically any security, liveness, and fairness requirement. At the same time it considerably simplifies reasoning about systems of ITMs since there is no inherent non-determinism in the scheduling of activations of ITIs.)

Another point to notice is that the above modeling allows ITIs to generate other ITIs "on the fly", in adaptive 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 ITI specifies which ITI it wishes to address in an "external write" instruction. For this purpose we let each ITI have an identity that is determined at invocation time (by the invoking instance) and is unchangeable. The fact that identities are determined by the code of the invoking ITIs, and are readable by the ITIs themselves, allows capturing within the model attacks that use adversarially chosen identities of parties. Furthermore, it provides a powerful tool for protocol design and analysis. In addition, the criteria for invoking ITIs guarantee that identities are *globally unique* within the system, thus preventing addressing ambiguities. (While giving the code access to identities that are guaranteed to be globally unique simplifies the model, it is indeed an idealization of reality. Yet, no expressibility is lost here since it is always possible to consider only protocols that ignore the identities. See Section 3.4.2 for further discussion on the role of identities in the present framework.)

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 ITIs 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 ITIs. Specifically, we let each identity consist of two separate fields: the session id (SID) and the party id (PID). A set of ITIs 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 ITIs within a protocol instance; they can also be used to associate ITIs 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-ITI 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 *input* and *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 ITI appears in Figure 1.

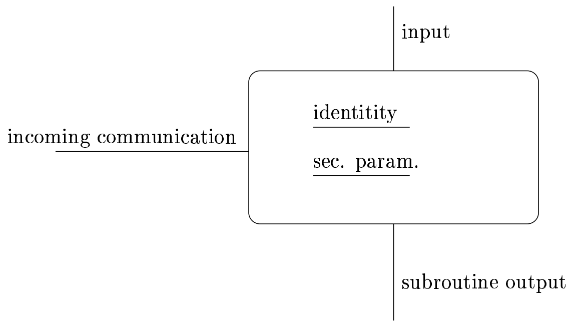

Figure 1: An instance of an interactive Turing machine. The tapes writable by other instances (ITIs) are the input, incoming communication, and subroutine output tapes. The identity and security parameter tapes are written to only at invocation time. For graphical clarity, we draw input tapes as lines coming from above, communication tapes as lines coming from either side, and subroutine output tapes as lines coming from below.

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 (program) M is locally probabilistic polynomial time (PPT) if, at any point during the execution of any ITI μ with code M, the 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 ITIs, plus the number of other ITIs invoked by μ, is less than the the length of μ'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 ITMs 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. An additional technical advantage of this definition is that 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. Finally, we note that the same definitional structure holds when the class of polynomially bounded ITMs is replaced by any other reasonable class of resource bounded computation.

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 an 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 made on the bare model, such as the composition theorem.

The model of protocol execution is parameterized by three ITMs: π, 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 at 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 the following system of ITMs.

Recall that an execution consists of a sequence of activations, where in each activation a single ITM instance (i.e., ITI) is active and may write to the tapes of one other ITI, which becomes the next instance to be activated. The environment $\mathcal{Z}$ is the initial ITM and is activated first. Next, the control function is defined as follows. The first ITI to be invoked by $\mathcal{Z}$ is the adversary $\mathcal{A}$. In all other activations, $\mathcal{Z}$ may either pass information to $\mathcal{A}$ (this information can be thought of as "instructions" or "questions"), or provide inputs to parties in an instance of π. That is, all ITIs invoked by $\mathcal{Z}$ throughout the computation (except for $\mathcal{A}$) are required to have the program π; 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. Party corruption is modeled as a 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. (To guarantee that the environment knows who is corrupted, we postulate that the adversary corrupts a party only after being instructed to do so by the environment.)

Once a party of protocol π 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 π may invoke other ITIS as subroutines, provide inputs to them, and obtain outputs from them. Once a subroutine of a party of π (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.

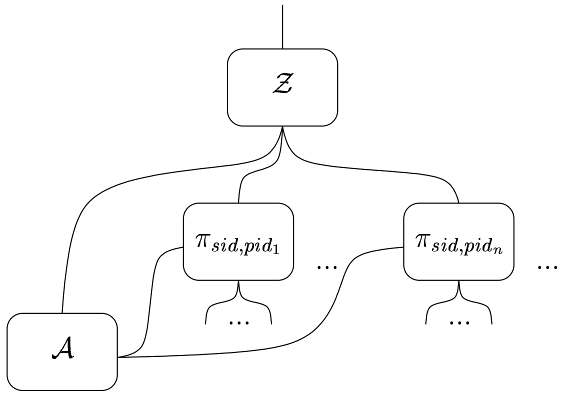

Figure 2: The model of protocol execution. The environment $\mathcal{Z}$ writes the inputs and reads the outputs of the parties running the protocol, while the adversary $\mathcal{A}$ controls the communication. In addition, $\mathcal{Z}$ and $\mathcal{A}$ interact freely via $\mathcal{A}$'s input tape 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 of π. It does not have direct access to the communication among the parties, nor to the inputs and outputs of the subroutines of π. The adversary $\mathcal{A}$ has access only to the communication among the parties and has no access to their inputs and outputs. This is in keeping with the intuition that $\mathcal{Z}$ represents, among other things, the protocols that provides inputs to and obtains outputs from the present instance of π, while $\mathcal{A}$ represents an adversary that attacks the protocol via the communication links, without having access to the local (and potentially secret) inputs and outputs.

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 ITIS, 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 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 ITIS.

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.

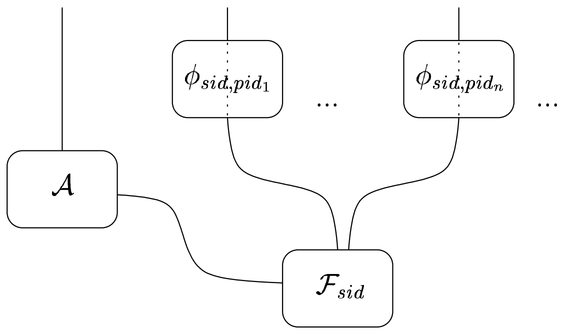

Figure 3: The ideal protocol ϕ for an ideal functionality $\mathcal{F}$. The parties of ϕ are "dummy parties": they relay inputs to the instance of $\mathcal{F}$ with the same SID, and relay outputs of $\mathcal{F}$ to their outputs. The adversary $\mathcal{A}$ communicates only with $\mathcal{F}$.

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

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

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

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 best match the intuitive notion of "emulating an ideal process".

The above notion of emulation 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, Sec. 4.2] 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 π* UC-realizes *an ideal functionality $\mathcal{F}$ if π emulates* 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 π UC-realizes $\mathcal{F}$ then the parties running π 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 π is obtainable by an adversary that only interacts with $\mathcal{F}$. (See [C00] for more motivational discussion.) In addition, the definition here guarantees that security is preserved under a very general composition operation, described below.

On quantitative notions of emulation. The notion of protocol emulation as defined above only guarantees that for any feasible (i.e. PPT) attack against π there is another feasible attack against the emulated protocol ϕ. We also provide more fine-tuned definitions that explicitly quantify parameters such as the difference between the complexities of the given adversary for π and the constructed adversary for phi, and the probability of distinguishing between the interaction with π and the interaction with ϕ. These definitions allow for more quantitative comparison between protocols, as well as bounding how the "quality of security" deteriorates by operations such as iterative replacement of a protocol by a protocol that emulates it, and protocol composition as described above.

Furthermore, this more quantitative study reveals the following interesting property of UC-emulation: It turns out that if a protocol π UC-emulates protocol ϕ as informally defined above, then there exists a specific polynomial $g(\cdot)$ (that depends only on π and ϕ) such that the complexity of the constructed adversary $\mathcal{S}$ is bounded by the complexity of the given adversary $\mathcal{A}$ plus $g(c(\cdot))$, where $c(\cdot)$ is essentially the overall communication of the protocol. Consequently, in typical cases where the volume of communication is bounded independently of the complexity of the adversary, the *difference* between the running times of $\mathcal{S}$ and $\mathcal{A}$ is bounded by a polynomial that depends only on π and ϕ. This can be interpreted as saying that "there exists a fixed polynomial $g(\cdot)$ such that, for any value of the security parameter k, any successful attack against ϕ can be turned into a successful attack against π with only $g(k)$ more resources".

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 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 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 case of ideal functionalities and ideal protocols follows as a special case. Let π be a protocol that uses subroutine calls to some protocol ϕ, and let ρ be a protocol that UC-emulates ϕ. The composed protocol, denoted $\pi^{\rho/\phi}$, is the protocol in which each invocation of ϕ is replaced by an invocation of ρ. That is, protocol π is modified so that each instruction to provide an input to some instance of ϕ is is replaced with an instruction to give the same input to an instance of ρ with the same identity, and each output received from an instance of ρ is treated as an output received from an instance of ϕ with the same identity. It is stressed that an execution of π may involve an unbounded number of concurrent instances of ϕ. Similarly, an execution of $\pi^{\rho/\phi}$ may involve an unbounded number of concurrent instances of ρ. When the replaced protocol ρ 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.

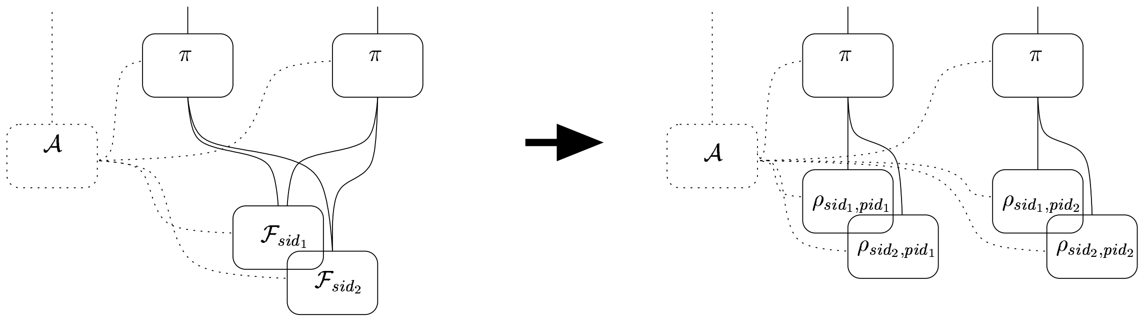

Figure 4: The universal composition operation, for the case where the replaced protocol is an ideal protocol for $\mathcal{F}$. Each instance of $\mathcal{F}$ (left figure) is replaced by an instance of ρ (right figure). The solid lines represent inputs and outputs. The dotted lines represent communication.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

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

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

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

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 ρ 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 ρ.) 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 ρ securely realizes functionality $\mathcal{F}$, and consider the following protocol π 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 π means that protocol ρ 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 ϕ be a protocol, let π be a protocol that makes subroutine calls to ϕ, let ρ be a protocol that emulates ϕ and let $\pi^{\rho/\phi}$ be the composed protocol. Let $\mathcal{A}$ be an 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 π 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 ρ. 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 ρ 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 ϕ are replaced with instances of ρ. 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 ϕ which are not PPT. More discussion and an example appear in Section 5.3.3.

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 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. Also, 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 $a \in \cup_{\kappa \leq k^d}\{0,1\}^\kappa$ 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 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 extends that definition by adding some syntax that facilitates the modeling of multi-party, multi-execution settings where multiple interacting machines run in the same system, and the number of interacting machines may grow depending on the computation with no a-priori bound. 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 *A tape of a Turing machine is called* externally writable (EW) *if 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. An* interactive Turing machine (ITM) *M is a Turing machine with the following tapes:*

- *An EW* identity *tape.*
- *An EW* security parameter *tape.*
- *An EW* input *tape.*
- *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 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 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 an 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 an identity of some ITM along with the code (i.e., transition function) of some ITM, and an arbitrary* contents *field.*[3]

The code of an ITM M may include instructions to write to a tape of another ITM. The syntax and effect of these instructions are described below.

Systems of ITMs. We specify the basic mechanics of running multiple ITMs within a single system, and coin the necessary 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 $C : \{0,1\}^* \to \{allow, disallow\}$ that determines the effect of the external-write instructions of the ITMs in the system.[4]

Executions of systems of ITMs. A configuration of an ITM M consists of the description of the code, the control state, the contents of all tapes and the head positions. A configuration is active if the activation tape is set to 1, else it is inactive. An instance $\mu = (M, id)$ of an ITM consists of the code (transition function) M, plus an identity string $id \in \{0,1\}^*$. (Somewhat abusing notation, we often use the same notation for an ITM and its code.) We say that a configuration is a configuration of instance μ if the code in the configuration is M and the contents of the identity tape in the configuration is id. We often use the acronym ITI to denote an ITM instance. (Intuitively, an ITI represents an instantiation of an ITM, namely a process that runs the code of the ITM on some specific input.)

An activation of an ITI μ is a sequence of configurations that correspond to a computation starting from some active configuration of μ, until an inactive configuration is reached. In this case we say that the activation is completed and that μ is waiting for the next activation. If a special halt state is reached then we say that μ has halted; in this case, it does nothing in all future activations.

An execution of a system $S = (I, C)$, given security parameter k and input x, consists of a sequence of activations of ITIs. The first activation starts from the configuration with the code of I, the input x written on the input tape, 1^k written on the security parameter tape, a sufficiently

[3]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, an equivalent formulation restricts all ITMs to have the transition function of the universal Turing machine, and have the actual code of an ITM to be written on a special externally writable tape.

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

long random string r written on the random tape, and identity 0. In accordance, the ITI $(I, 0)$ is called the initial ITI.

The execution ends when the initial ITI halts (that is, when a halting configuration of the initial ITI is reached). The output of the execution is the contents of the output tape of the initial ITI in its halting configuration.

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 ITI to be activated, once an activation is completed. These are described next.

Writing to a tape of another ITI and invoking ITIs. An external-write instruction of an ITM specifies the following parameters: the codes and identities of the present ITI and a target ITI, 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, by an ITI $\mu = (M, id)$ to an ITI $\mu' = (M', id')$ is as follows.

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

2. If C allows the operation, and an ITI $\mu'' = (M'', id'')$ with identity $id'' = id'$ currently exists in the system (namely, one of the past configurations in the execution so far has identity id'), then:

   (a) If the target tape is the incoming communication tape of μ', then the specified data is written on the incoming communication tape of μ'', together with the identity id of the writing ITI. (That is, a new configuration of μ'' is generated. This configuration is the last configuration of μ'' in this execution, with the new information written on the incoming communication tape.) This is done regardless of whether the code M'' of μ'' equals the code M' specified in the external write request.

   This rule has the effect that an ITI does not necessarily know the code of the ITI it sends messages to or receives messages from over the communication tapes.

   (b) If the target tape is the subroutine output tape of μ', then the specified data is written on the subroutine output tape of μ'' together with the code and identity of μ. Also here, this is done regardless of whether $M'' = M'$.

   This rule has the effect that an ITI knows the code of the ITIs that write to its subroutine output tape. However, an ITI does not necessarily know the code of the ITI to whom it generates output.

   (c) If the target tape is the input tape of μ' and $M'' = M'$, then the specified data is written on the subroutine output tape of μ'' together with id, the identity of μ. If $M' \neq M''$ then μ transitions to a special error state.

   This rule has the effect that an ITI can verify the code of the ITI to whom it provides input. However, an ITI does not necessarily know the code of the ITI from which it receives input.

3. If C allows the operation, and no ITI with identity id' exists in the system, then a new ITI with code M' and identity id' is invoked. That is, a new configuration is generated, with code M', the value id' written on the identity tape, 1^k written on the security parameter tape, and

a sufficiently long random string is written on the random input tape. Once the new ITI is invoked, the external-write instruction is carried out as above. In this case, we say that μ invoked μ'.

Discussion. Two remarks are in order here: First, the above invocation rules for ITIs, together with the fact that the execution starts with a single ITI, guarantee that each ITI in the system has unique identity. That it, no two ITIs have the same identity, regardless of their codes. Second, since the code and identity of the target ITI should be specified in the external-write instruction, they must be computed by the writing ITI prior to the invocation. In particular, there are no restrictions on the contents of the SID and PID. In particular, they can contain cryptographic keys or other identification information. More discussion appears in Section 3.4.

Determining the order of activations. The order of activations is simple: In each activation, We allow an ITI to execute at most a single external-write instruction. The ITI whose tape was written to in an activation is activated next (i.e., the activation tape in the newly generated configuration of this ITI is set to 1). If no external-write operation was performed then the initial ITI is activated next.

Discussion. Other orders of activation are of course possible (e.g., one can postulate that the ITIs 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 ITI 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).

Additional terminology. When an ITI μ writes a message m to the incoming communication tape of ITI μ', we say that μ sends m to μ'. When μ writes a value x onto the input tape of μ', we say that μ passes input x to μ'. When μ' writes x to the subroutine-output tape of μ, we say that μ' passes output x (or simply outputs x) to μ. We say that μ' is a subroutine of μ if μ has passed input to μ' or μ' has passed output to μ. μ' is a subsidiary of μ if μ' is a subroutine of μ or of another subsidiary of μ.

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 sequence of all the configurations in all the activations of the execution of the system so far. A transcript of an execution of a system is the final state in the execution, namely the state where the last configuration is a terminating configuration of the initial ITI.

Extended systems. An extended system is a system where the control function can also *modify* the external-write requests. 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 newly generated ITIs. See more details in Section 4.

Outputs of executions. We use the following notation. Let $\textsc{out}_{I,C}(k,x)$ 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. Here the probability is taken over the random choices of all the ITMs in the system. Let $\textsc{out}_{I,C}$ denote the ensemble $\textsc{out}_{I,C}(k,x)\}_{k\in\mathbf{N},x\in\{0,1\}^*}$.

Protocols. A 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 π with SID *sid* is the set of ITIs in the system, whose code is π, and whose SID is *sid*. (Consequently, the PIDs of the ITIs in a protocol instance are necessarily distinct.) We assume that π ignores all incoming messages where the sender SID is different than the local SID. Each ITI in a protocol instance is called a party. A sub-party is a subroutine either of a party or of another sub-party. The extended instance of π 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.[5]

Second, the convention of associating an SID with each protocol instance, where the SID is known to all parties, is convenient 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 this extra generality does not seem essential for our treatment.

We use several alternative naming methods for parties (i.e., ITIs) in a protocol instance. We let π_i denote the ith party running protocol π, 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 π_i's tapes). We let $\textsc{id}(\pi_i)$ (resp., $\textsc{pid}(\pi_i)$, $\textsc{sid}(\pi_i)$) denote the identity (resp., PID, SID) of the ITI π_i. We also use $\pi_{(id)}$ to denote the party (π, id). That is, $\textsc{id}(\pi_{(id)}) = id$. When the protocol π is understood from the context or is not specified we sometimes use the generic notation P_i and $P_{(id)}$ instead of π_i and $\pi_{(id)}$, respectively. Furthermore, when there is no danger of confusion we write P_i and mean $\textsc{id}(P_i)$.

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 $\textsc{id}(P_i)$, and between a party and a sub-party. (Protocols with a fixed number of parties and fixed

[5]Different parties in a protocol often play different roles and run different programs. This can be captured within the above single-ITM formalism by specifying in π 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.

identities can be obtained as a special case of the present formalism.) (c) No general simple rule regarding the order of activation of ITIs 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 ITI by another was not treated as explicitly as here. In particular, the need to specify the code of the invoked ITI was not explicitly addressed, and uniqueness of ITIs was guaranteed based on identities rather than *extended* identities. (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 resource bounded ITMs, and more specifically 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 ITIs 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.

Our goal is to formulate a definition of polynomial time that matches our intuition in a 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). However, That formulation is geared towards capturing pairs of ITMs whose inputs are fixed throughout the computation. We extend it in a number of ways to fit our more general setting, where ITM instances can write to arbitrary tapes of other instances, and where new instances can be generated throughout the execution. Specifically, we need to be able to bound the overall running time of a system of ITMs. In particular, both the overall runtime of each ITM instance, and the overall number of instances should be bounded.

One way to guarantee this "overall bound" is to explicitly specify a bound on the runtime of the system, and require that the execution of the system halts once this bound is reached. However, as argued in Section 3.4.3, having a known exact bound on the runtime of a system causes definitions of security to be more cumbersome. We thus choose a different, more flexible method for guaranteeing overall polynomiality. Specifically, in addition to requiring that the overall runtime of a PPT ITM M is bounded by a polynomial function of the length of the input plus the security parameter, we require that the number of bits written by M onto the input tapes of other ITMs, plus the number of different ITM instances that M writes to, is less than the number of bits written on M's input tape. We do that by requiring that the number of computational steps taken by M is bounded by a polynomial in n, where n is calculated as follows: To the security parameter k, add the overall number of bits written so far on M's input tape, and then subtract the number of bits written by M so far to input tapes of ITM instances. Finally, subtract k times the number of different ITM instances whose tapes are written to by M. (Subtracting the number of bits written on the input tapes of other instances has the effect that writing to the tapes of existing instances does not increase the overall runtime available to a system. Subtracting k times the number of different instances written to has the effect that invoking new instances does not increase the overall runtime available to the system, either.[6]) In order 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. Furthermore, we require that all the subsidiaries of M adhere to the same rules, and are all bounded by a polynomial that is no larger than the polynomial bounding M.[7]

Definition 3 (p-bounded, PPT) *Let $p : \mathbf{N} \to \mathbf{N}$. An ITM M is* locally p-bounded *if, at any point during an execution of M (namely, in any configuration of M), the overall number of computational steps taken by M so far is at most $p(n)$, where*

$$n = k + n_I - n_O - k \cdot n_N,$$

k is the security parameter, n_I is the overall number of bits written so far on M's input tape, n_O is the number of bits written by M so far to input tapes of ITM instances, and n_N is the number of different ITM instances whose tapes are written to by M.

If M is locally p-bounded, and in addition either M does not make external write requests, or each external write request specifies a recipient ITM which is p-bounded, then we say that M is p-bounded. *M is* PPT *if there exists a polynomial p such that M is p-bounded.*

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

It can be readily seen that the overall number of computational steps taken in an execution of a system, where the initial ITM is p-bounded, is bounded by $p(n + k)$, where n is the initial input to the system and k is the security parameter. In particular, the overall number of ITM instances invoked in an execution is bounded by $p(n + k)$. Furthermore, if the control function C is computable in time $q(t)$, where t is the length of input to C, then an execution of a system of ITMs can be simulated by a sequential Turing machine given the initial input x and the security parameter k, in $O(q(p(|x| + k)))$ steps. That is:

Proposition 4 *If the initial ITM in a system (I, C) is p-bounded, and in addition the control function C is computable in time $q(\cdot)$, then an execution of the system can be simulated on a single (non-interactive) Turing machine M, which takes for input the initial input x and the security parameter k, and runs in time $O(q(p(k + |x|)))$. In particular, if both I and C are PPT then so it M. The same holds also for extended systems of ITMs, as long as all the ITMs invoked are p-bounded.*

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

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.

[6]It would suffice of course to subtract only the number of ITM instances that were actually invoked by the instance in question. However, this number cannot be determined exclusively by the code of the ITM in question; rather, it depends on the entire system.

[7]Previous versions of this work considered also an additional, more relaxed notion of polytime bounded computation, called A-PPT. In the present formulation this notion becomes unnecessary, and is thus omitted. See more discussion in Section 4.3. We note that a similar notion is used in [K05].

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 π-calculus of Milner [M89, M99], the *spi*-calculus of Abadi and Gordon [AG97] (that is based on π-calculus), the framework of Lincoln et. al. [LMMS98] (that uses the functional representation of probabilistic polynomial time in [MMS98]), the I/O automata of Lynch [Ly96], the probabilistic I/O automata of Lynch, Segala and Vaandrager [SL95, LSV03] 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 naturally 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). Furthermore, ITMs allow us to represent in a natural way a specific code to be executed, as opposed to merely functional specification of the desired behavior.

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, C+05].

Finally, 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 (e.g., to model adversarial entities).

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. As will be seen, the model 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 ITIs. Recall that the invocation of a new ITM instance (i.e., ITI) is implicit, and occurs only when an existing ITI writes to a tape of a non-existing ITI. 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 ITI is invoked before it is first activated. (In this case it would be necessary to explicitly guarantee uniqueness of identities, as discussed below.)

Making the identities available to the code. The present formalism allows ITMs to read and use their identities. Indeed, incorporating the identities (or parts thereof) in the code of the protocol is an important and powerful tool in cryptographic protocol design. In fact, in some cases making use of the identities is *essential.* This fact is exemplified in [LLR02] for the basic tasks of broadcast and Byzantine agreement. Still, it is of course possible to study within the present framework protocols that do not use identities (by explicitly considering only protocols that ignore the identities). We note that other formulations of systems of ITMs [K05] do not let the code access to system-generated identities, thus requiring the protocol to generate its own addressing mechanism. Still, the expressibility seems remain the same in all models.

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

While generating globally unique identities is quite simple in practice (e.g., by choosing an identity at random from a large enough domain), providing parties with identities that are guaranteed to always be globally unique is indeed an idealization of reality. We adopt this mechanism since it simplifies the model, and avoids duplicate indexing and naming mechanisms.

Still, we note that other methods of determining the identities of ITIs 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 ITIs. In addition, multiple ITIs can have the same "local ID", while being distinguished via the IDs of their "ancestors" in the "ID tree". We prefer the above non-hierarchical method since it is more general; indeed, the hierarchical method can be cast within the present modeling.

On determining the session identifiers. Recall that the SIDs of all ITIs in the same protocol instance must be identical. In situations where different ITIs in a protocol instance are invoked by different ITIs (which, presumably, represent different entities in a distributed network), the invoking entities must somehow agree on a joint SID. There are multiple ways for such agreement to take place. Let us mention three such methods. A first method, which is applicable in cases where there is some prior coordination between the entities that call a given protocol instance, is to determine the SID of the instance in advance (potentially as a function of the SID of the calling protocol and other run-time parameters). This method is natural, say, when the protocol instance in question is a subroutine in a larger protocol.

A second alternative is to design the protocol in such a way that the agreement on the SID is done by the protocol instance itself. For instance, make sure that all ITIs in a protocol instance (except for the first one) are invoked via incoming messages from other ITIs in that instance, rather than by ITIs from other protocol instances. This way, all ITIs in a protocol instance can have the same SID as the first ITI in the protocol instance, *without any prior coordination and without running an agreement protocol.* Indeed, in the rest of this work we try to design protocols and protocol specification in a way that allows this procedure to take place, thus eliminating the need for prior agreement on the SID.

A third alternative, which is viable in situations where there is no prior coordination among the entities that invoke the various ITIs in a protocol instance, and the parties wish to jointly determine the SID, is to run some simple agreement protocol among the parties in order to determine a 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.

Finally we remark that it is possible to formulate alternative conventions that allow the SIDs of the parties in a protocol instance to be related in some other way, rather than being equal. Such a more general convention may allow more loose coordination between the ITIs in a protocol instance. Also, SIDs may be allowed to change during the course of the execution. We stick to the present convention since it considerably simplifies the treatment throughout this work.

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

On the distinction between input, communication, and subroutine output 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, 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 ITI writes into the subroutine output tape of another ITI, the *code* of the writing instance is ideally attached to the written value. Furthermore, when an ITI writes to the input tape of another ITI, it can specify the code of the target ITI; if the specified code does not match the actual code then an error message is generated. 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.

These conventions facilitate modeling multi-instance, multi-party computing environments. For instance, while the basic unit in a system of ITMs is a single ITI (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 ITIs. 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. The recipient knows the program that computed the received value. 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 ITI, 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 ITI is active at any point during the execution. This stands in contrast with the physical nature of distributed systems where computations take place in multiple, physically separate places at the "same time". Furthermore, there exist mathematical models of distributed computation that directly model such "true concurrency" (see e.g. [Ly96, 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 ITIs 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 ITIs 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 ITI 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 ITIs. 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 ITIs are allowed to write only to the incoming communication tape of C, and C writes to externally writable tapes of all other ITIs. 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 an instance 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 4.3). In contrast, this equivalence does not hold with respect to the notions in [c00] and in other works (see e.g. [hu05]).

Using the control function to bound the running time. An alternative approach to defining PPT systems of ITMs is to avoid making local restrictions on the running time of individual ITIs, and instead impose an overall bound on the runtime of the system. For instance, one can potentially bound the overall runtime of the control function (or control ITM) by some fixed polynomial in the input length, and postulate that the execution halts as soon as this bound is reached. (Indeed, previous versions of this work used such a mechanism.) This approach is attractive since it is considerably simpler. However this approach 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 last ITI to be activated. This provides an "artificial" way for the last ITI to be activated (i.e., the initial ITI) to obtain global information on the execution via the timing in which the execution halts. Consequently, this notion of PPT systems would cause the notions of security defined in the rest of this work to be artificially restrictive. (Jumping ahead, the effect of bounding the runtime this way is that the environment will be able to distinguish between two executions by simply comparing the overall number of computational steps taken in the two executions. However, we would like to consider two systems as equivalent from the point of view of the environment even in cases where the overall number of computational steps and communicated bits in the systems might differ by a polynomial amount.)

Recognizing PPT ITMs. One general 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 σ of a *locally* PPT ITM will first specify an exponent c. It is then understood that the ITM encoded in σ halts as soon as the overall number of steps taken by the ITM encoded in σ, 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 σ 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 Müller-Quade, Rainer Steinwandt and Dominic Unruh for very useful discussions on modeling PPT ITMs and systems, and for pointing out to us shortcomings of the definition of PPT ITMs in earlier versions of this work and of some other definitional attempts. Discussions with Dennis were particularly instructive.

4 Defining security of protocols

This section presents a definition of protocols that securely realize a given ideal functionality as outlined in Section 2.2. The order of presentation here is slightly different than there: 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. The general notion of protocol emulation is presented in Section 4.2. Section 4.3 presents several alternative formalizations of the definition and demonstrates their equivalence to the main one. Ideal functionalities and the ideal protocol for carrying out a given functionality are presented in Section 4.4, followed by the definitions of securely realizing an ideal functionality. Finally, Section 4.5 defines hybrid protocols.

4.1 The model of protocol execution

The model for protocol execution is parameterized by three ITMs: the protocol π to be executed, the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$. That is, given $\pi, \mathcal{Z}, \mathcal{A}$, the model for executing π is the extended system of PPT ITMs $(\mathcal{Z}, C_{\text{EXEC}}^{\pi,\mathcal{A}})$ (as defined in Section 3.2), where the initial ITM in the system is the environment $\mathcal{Z}$, and the control function $C_{\text{EXEC}}^{\pi,\mathcal{A}}$ 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 ITI to be invoked by $\mathcal{Z}$ is set by the control function to be $\mathcal{A}$. In addition, as the computation proceeds, $\mathcal{Z}$ may invoke as subroutines an unlimited number of ITIs, pass inputs to them, and obtain outputs from them, subject to the restriction that all the invoked ITIs have the same SID (which is chosen by $\mathcal{Z}$). The code of these ITIs is set by the control function to be the code of π, regardless of the code specified by $\mathcal{Z}$. Consequently, all the ITIs invoked by $\mathcal{Z}$, except for $\mathcal{A}$, are parties in a single instance of π. Other than that, $\mathcal{Z}$ may not communicate with any other ITIs. That is, it can not pass inputs to any other ITI, nor can any other ITI pass output to $\mathcal{Z}$. In particular, $\mathcal{Z}$ cannot interact with

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

Given protocol π, 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 ITI invoked by $\mathcal{Z}$ is set to be the adversary, $\mathcal{A}$. The code of all other ITIs invoked by $\mathcal{Z}$ is set to π; furthermore, they are all these ITIs are 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 ITI with identity id. This operation is modeled as a special `(corrupt)` message delivered to that ITI. This operation is allowed only once $\mathcal{A}$ receives an input `(corrupt` μ`)` from $\mathcal{Z}$. The message may include additional parameters set by $\mathcal{A}$.
3. Once a party of π, 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 $\mathcal{A}$, and it can pass inputs and outputs to other parties and sub-parties of this instance of π. In addition, the parties of π can pass pass outputs to $\mathcal{Z}$.

   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

subroutines of parties of the single instance of π.[8]

The control function allows the parties and sub-parties of π to invoke ITIs and to pass inputs and outputs to any other ITI other than the adversary and the environment. In addition, they can write messages on $\mathcal{A}$'s incoming communication tape. These messages may specify an identity of a party or sub-party of π as the final destination of the message.

In addition, the control function allows the adversary $\mathcal{A}$ to send messages to any ITI 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 the recipient.) There need not be any correspondence between the messages sent by the parties and the messages delivered by the adversary. $\mathcal{A}$ may not invoke any subroutines.

$\mathcal{A}$ may also *corrupt* parties or sub-parties of π. Formally, corruption of a party or a sub-party with identity id is modeled via a special `(corrupt)` message (potentially with additional

[8]This restriction of the environment simplifies the model considerably, since it restricts attention to a single protocol execution. An alternative formulation would allow $\mathcal{Z}$ to invoke arbitrary ITIs with multiple different SIDs. If none of these ITIs sends input to or receives output from a subroutine of the π-instance under consideration, this more general formulation ends up being equivalent to the present one. Otherwise, this more general formulation captures a meaningful extension of the model, addressing security in the presence of a pre-existing setup. See more details in [DPW05].

parameters) delivered by $\mathcal{A}$ to that ITI. 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 6.7. 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 ITIs.

We remark that the model allows $\mathcal{A}$ to invoke new ITIs by delivering messages to them. These ITIs may, but need not be parties of the instance of π with SID sid. Allowing $\mathcal{A}$ to invoke parties of π with SID sid is important since it allows modeling protocol instances that are invoked by a message coming from the network. One could potentially restrict $\mathcal{A}$ to invoking *only* parties of the current instance of Still, we require that the only parties of the instance of π with SID sid can pass outputs to $\mathcal{Z}$.

We restrict attention to environments that output a single bit. As discussed in Section 4.2, no generality is lost by this restriction. A summary of an execution of a protocol is described in Figure 5. See Figure 2 on page 21 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 $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

4.2 Protocol emulation

We formalize the general notion of emulating one protocol via another protocol. (See discussion in Section 2.2.) Essentially, protocol π emulates protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that no environment can tell whether it is interacting with π and $\mathcal{A}$ or with ϕ and $\mathcal{S}$:[10]

[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 ITI. In prior versions, $\mathcal{A}$ learned the entire sequence of prior states of the corrupted ITI. 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 ITIs (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 ITIs.)

[10]The present formulation of Definition 5 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.

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

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

On statistical and perfect emulation. Definitions 5 and 13 can be extended to the standard notions of statistical and perfect emulation (as in, say, [c00]). That is, when $\mathcal{A}$ and $\mathcal{Z}$ are allowed unbounded complexity, and the simulator $\mathcal{S}$ is allowed to be polynomial in the complexity of $\mathcal{A}$, we say that π statistically UC-emulates ϕ. If in addition the two sides of (1) are required to be identical then we say that π perfectly UC-emulates ϕ. Another variant allows $\mathcal{S}$ to have unlimited computational power, regardless of the complexity of $\mathcal{A}$; however, this variant provides a weaker security guarantee, see discussion in [c00].

On security with respect to uniform-complexity inputs. Definitions 5 and 13 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" 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 captured by considering only environments whose external input contains no information other than its length, e.g. it is 1^n for some n. Such environments would choose the inputs of the parties based on some uniform-complexity internal stochastic process.

More quantitative notions of emulation. The notion of protocol emulation as defined above only provides a "qualitative" measure of security. That is, it essentially only gives the guarantee that "any feasible attack against π can be turned into a feasible attack against ϕ". We formulate more quantitative variants of this definition. Specifically, we quantify two parameters: the emulation slack, meaning the probability in which the environment distinguishes between the interaction with π and the interaction with ϕ, and the simulation overhead, meaning the difference between the complexity of the given adversary $\mathcal{A}$ and that of the constructed adversary $\mathcal{S}$. Recall that an ITM is p-bounded if the function bounding its running time is p (see Definition 3), and that a functional is a function from functions to functions. Then:

Definition 6 *Let π and ϕ be multi-party protocols and let g, ϵ be functionals. We say that π UC-emulates ϕ with* emulation slack *ϵ and* simulation overhead *g (or, in short, π (e, g)-*UC-emulates *ϕ), if for any polynomial $p_{\mathcal{A}}(\cdot)$ and any $p_{\mathcal{A}}$-bounded adversary $\mathcal{A}$, there exists a $g(p_{\mathcal{A}})$-bounded adversary*

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 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 or adversary class.

$\mathcal{S}$, such that for any polynomial $p_{\mathcal{Z}}$, any $p_{\mathcal{Z}}$-bounded environment $\mathcal{Z}$, any large enough value $k \in \mathbf{N}$ and any input $x \in \{0,1\}^{p_{\mathcal{Z}}(k)}$ we have:

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

A more concrete variant of Definition 6 abandons the asymptotic framework and instead concentrates on a specific value of the security parameter k:

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

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

Including the security parameter k in the parameterized definition is necessary when the protocol depends on it. Naturally, when k is understood from the context it can be omitted. A more relaxed variant of Definition 7 parameterizes also the running times of the adversary and the environment:

Definition 8 *Let π and ϕ be multi-party protocols, and let $k, t_{\mathcal{A}}, t_{\mathcal{S}}, t_{\mathcal{Z}}, \epsilon \in \mathbf{N}$. We say that π $(k, t_{\mathcal{A}}, t_{\mathcal{S}}, t_{\mathcal{Z}}, \epsilon)$-UC-emulates ϕ if for any adversary $\mathcal{A}$ that runs in time $t_{\mathcal{A}}$ there exists an adversary $\mathcal{S}$ that runs in time $t_{\mathcal{S}}$ such that for any environment $\mathcal{Z}$ that runs in time $t_{\mathcal{Z}}$, and any input $x \in \{0,1\}^{t_{\mathcal{Z}}}$ we have:*

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

We note however that Definition 8 is considerably weaker than definition 7, since it guarantees security only for adversaries and environments that are bounded by specific run-times. Furthermore, both the protocols and the simulator can depend on these run-times. In contrast, Definition 7 bounds the specified parameters for any arbitrarily complex environment and adversary. Consequently, while the UC theorem can be easily adapted to the formulations of Definitions 6 and 7, Definition 8 does not seem to guarantee universal composability.

An interesting property of the notion of UC-emulation is that the simulation overhead can be bounded by an additive factor that depends only on the overall communication of the adversary (or, equivalently, of the protocol π). Furthermore, the dependence is polynomial. That is, say that the communication of adversary $\mathcal{A}$ is bounded by a function $c_{\mathcal{A}} : \mathbf{N} \to \mathbf{N}$ if the overall number of bits written by $\mathcal{A}$ to the incoming communication tapes of parties, plus the overall number of bits written by parties to $\mathcal{A}$'s incoming communication tape, is bounded by $c_{\mathcal{A}}(n)$, where n is the overall length of input to $\mathcal{A}$ plus the security parameter. (Clearly, $c_{\mathcal{A}}(n) \leq p_{\mathcal{A}}(n)$. Often, however, it is much smaller than $p_{\mathcal{A}}(n)$.) Say that a functional ϵ is negligible if $e(p)$ is a negligible function whenever p is a polynomial. Then we have:

Claim 9 *Let π and ϕ be protocols such that π UC-emulates ϕ as in Definition 5. Then there exists a negligible functional ϵ and a polynomial α such that π UC-emulates ϕ with emulation slack ϵ and simulation overhead $g(p_{\mathcal{A}})(\cdot) = p_{\mathcal{A}}(\cdot) + \alpha(c_{\mathcal{A}}(\cdot))$, where $c_{\mathcal{A}}$ bounds the communication of adversary $\mathcal{A}$.*

Said otherwise, if π UC-emulates ϕ then it is guaranteed that the running time of the adversary grows by an additive polynomial factor that depends only on the communication generated by $\mathcal{A}$, rather than on the "internal computation time" of $\mathcal{A}$. In fact, in typical cases where the "effective communication complexity" of a protocol depends only on the protocol's input rather than on the adversary the simulation overhead is effectively $g(p_{\mathcal{A}})(\cdot) = p_{\mathcal{A}}(\cdot) + \alpha(k)$, where $\alpha(k)$ is a quantity that does not depend on the adversary. This means that the overhead of running $\mathcal{S}$ rather than $\mathcal{A}$ is strictly an additive value that is polynomial in the security parameter and independent of $\mathcal{A}$. (We remark that most interesting protocols have that property. In particular, since the communication complexity of each party in a protocol is bounded by a polynomial in the input length of that party, we have that any protocol where the number of parties does not depend in the adversary satisfies this condition.)

The proof of Claim 9, while not immediate from the definition of the simulation overhead, follows as an easy corollary from the proof of Claim 10 below. We thus postpone the proof of Claim 9 till after the proof of Claim 10.

On the transitivity of emulation. It is easy to see that if protocol π_1 UC-emulates protocol π_2, and π_2 UC-emulates π_3, then π_1 UC-emulates π_3. Moreover, if π_1 (e_1, g_1)-UC-emulates π_2, and π_2 (e_2, g_2)-UC-emulates π_3, then π_1 $(e_1+e_2, g_2 \circ g_1)$-UC-emulates π_3. (Here e_1+e_2 is the functional that output the sum of the outputs of e_1 and e_2, and $\circ$ denotes composition of functionals.) Transitivity for any number of protocols $\pi_1, ..., \pi_n$ follows in the same way. Note that if the number of protocols is not bounded by a constant then the complexity of the adversary may no longer be bounded by a polynomial. Still, when there is an overall polynomial bound on the communication of all the protocols $\pi_1, ..., \pi_n$, Claim 9 implies that the simulation overhead remains polynomial as long as the number of protocols is polynomial in the security parameter. Similarly the emulation slack remains negligible as long as the number of protocols is polynomial. Finally, we stress that the question of transitivity of emulation should not be confused with the question of multiple *nesting* of protocols, which is discussed in Section 5.3.

4.3 Alternative formulations of protocol emulation

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

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

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

4.3.1 Security with respect to the dummy adversary

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

More specifically, the dummy adversary, denoted $\mathcal{D}$, proceeds as follows. When activated with an incoming message m on its incoming communication tape, adversary $\mathcal{D}$ passes m as output to $\mathcal{Z}$. (Recall that m already contains with it the identity of the sender.) When activated with an input (m, id, c) from $\mathcal{Z}$, where m is a message, id is an identity, and c is a code for a party, $\mathcal{D}$ delivers the message m to the party 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 $\mathcal{D}$ corrupts parties when instructed by $\mathcal{Z}$, and passes all gathered information to $\mathcal{Z}$.

To guarantee that $\mathcal{D}$ is a PPT ITM as in Definition 3, we need to make sure that the running time of $\mathcal{D}$ is polynomial in the length of its input. For this purpose, we say that $\mathcal{D}$ halts if the number of bits it needs to write to the incoming communication tapes of other ITIs is more than the number of bits written on its input tape. We note that such a situation might indeed happen if the number of bits received on the incoming communication tape is larger than the number of bits written on the input tape. Still, we would like to allow the environment to enable $\mathcal{D}$ to forward long messages written by parties; to do this, we slightly modify the behavior of $\mathcal{D}$: When activated with an incoming message m on its incoming communication tape, adversary $\mathcal{D}$ first passes $(\texttt{length}, |m|)$ as output to $\mathcal{Z}$, where $|m|$ is the length of m, represented in binary. Then, if it receives an input $1^{|m|}$ (i.e., the length of m in unary), then $\mathcal{D}$ passes m as output to $\mathcal{Z}$. If $1^{|m|}$ is not received then $\mathcal{D}$ halts.

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

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

Proof: Clearly if π UC-emulates ϕ according to Definition 5 then it UC-emulates ϕ with respect to dummy adversaries. The idea of the derivation in the other direction is that, given direct access to the communication sent and received by the parties, the environment can run any adversary by itself. Thus quantifying over all environments essentially implies quantification also over all adversaries. More precisely, let π, ϕ be protocols and let $\tilde{\mathcal{S}}$ be the adversary guaranteed by the definition of emulation 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 π UC-emulates ϕ according to Definition 5. For this purpose, given an adversary $\mathcal{A}$ we construct the adversary $\mathcal{S}$ as follows. $\mathcal{S}$ runs simulated instances of $\mathcal{A}$ and $\tilde{\mathcal{S}}$. 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. 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. Whenever the simulated $\tilde{\mathcal{S}}$ generates a $(\texttt{length}, l)$ output, $\mathcal{S}$ gives input 1^l to $\tilde{\mathcal{S}}$. When $\tilde{\mathcal{S}}$ output a value v (with $|v| = l$), $\mathcal{S}$ activates the simulated $\mathcal{A}$ with v as incoming message.

4. $\mathcal{S}$ follows the instructions of $\tilde{\mathcal{S}}$ regarding delivering messages to parties. That is, if $\tilde{\mathcal{S}}$ instructs to write a message m on some tape of an ITI P, then $\mathcal{S}$ writes m to that tape of P. When $\mathcal{S}$ obtains a message m on its incoming communication tape, it writes m on the incoming communication tape of $\tilde{\mathcal{S}}$.

5. If the local instance of either $\mathcal{A}$ or $\mathcal{S}$ reaches its bound on running time, as a function of its input, then $\mathcal{S}$ halts.

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

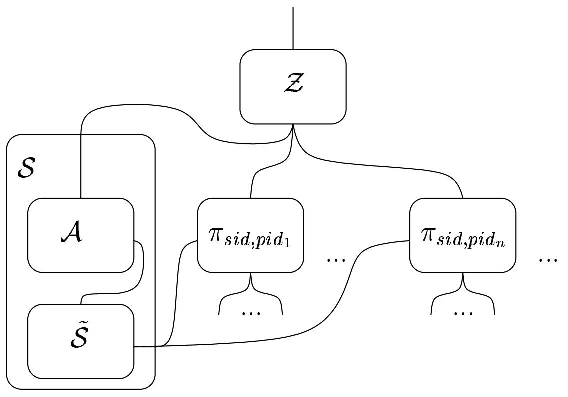

Figure 6: The operation of simulator $\mathcal{S}$ in the proof of Claim 10: Both $\mathcal{A}$ and $\tilde{\mathcal{S}}$ are simulated internally by $\mathcal{S}$. The same structure represents also the operation of the shell adversary in the definition of black-box simulation.

Analysis of $\mathcal{S}$. We first note that $\mathcal{S}$ is PPT. Indeed, the running time of $\mathcal{S}$ on input of length n is bounded by $p_{\mathcal{A}}(n) + p_{\mathcal{S}}(c_{\mathcal{A}}(n))$, where $p_{\mathcal{A}}$ is the polynomial bounding the running time of $\mathcal{A}$, $p_{\mathcal{S}}$ is the polynomial bounding the running time of $\mathcal{S}$, and $c_{\mathcal{A}}$ is the polynomial bounding the overall number of bits written by $\mathcal{A}$ on the incoming communication tapes of other ITIs, plus the number of bits received by $\mathcal{A}$ on its incoming communication tape. (In general $c_{\mathcal{A}}$ can be as high as $p_{\mathcal{A}}$; but it may often be considerably smaller.)

Next we assert the validity of $\tilde{\mathcal{S}}$. Assume for contradiction that there is an adversary $\mathcal{A}$ and environment $\mathcal{Z}$ such that $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \not\approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. We construct an environment $\tilde{\mathcal{Z}}$ such that $\text{EXEC}_{\pi,\tilde{\mathcal{S}},\tilde{\mathcal{Z}}} \not\approx \text{EXEC}_{\pi,\mathcal{D},\tilde{\mathcal{Z}}}$. Environment $\tilde{\mathcal{Z}}$ runs an interaction between simulated instances of $\mathcal{Z}$ and $\mathcal{A}$. Whenever $\tilde{\mathcal{Z}}$ is activated with value $(\texttt{length}, l)$ on its subroutine output tape, coming from the adversary, it passes input 1^l to the adversary. When it receives another output value v from its adversary, $\tilde{\mathcal{Z}}$ 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 π, 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 $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ are identical; in particular, $\mathcal{S}$ never runs out of running time before the simulated $\mathcal{A}$ within $\mathcal{S}$ does. Similarly, ensembles $\text{EXEC}_{\pi,\mathcal{D},\tilde{\mathcal{Z}}}$ and $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ are identical; in particular, $\mathcal{D}$ never runs out of running time. $\square$

From a technical point of view, emulation with respect to the dummy adversary is an easier definition to work with, since it involves one less quantifier, and furthermore it restricts the interface of the environment with the adversary to be very simple. Indeed, we almost always prefer to work with this notion. However, we chose not to present this formulation as the main notion of protocol emulation, since we feel is is somewhat less intuitively appealing than Definition 13. 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 definition has basic properties such as transitivity.

Proof of Claim 9. Claim 9 states that if a protocol π UC-emulates protocol ϕ then for any adversary $\mathcal{A}$ whose running time is bounded by the polynomial $p_{\mathcal{A}}(\cdot)$ and whose communication is bounded by $c(\cdot)$ there is a simulator whose running time is bounded by $p_{\mathcal{A}}(\cdot) + \alpha(c(\cdot))$. The claim follows directly from the proof of Claim 10. Indeed, if π UC-emulates ϕ then there exists a simulator $\tilde{\mathcal{S}}$ for the dummy adversary. Then, for any adversary $\mathcal{A}$ construct the simulator $\mathcal{S}$ from $\mathcal{A}$ and $\tilde{\mathcal{S}}$ as in the proof of Claim 10. The constructed $\mathcal{S}$ obeys the claimed complexity bounds. $\square$

Doing without the dummy adversary. Consider the following simplified variant of the notion of emulation with respect to the dummy adversary. Instead of having a dummy adversary that basically serves as a "channel" between the environment and the parties, change the model of execution (of protocol π) so that the dummy adversary is no longer present. Instead, the parties write outgoing messages directly to the incoming communication tape of the environment, and the environment writes delivered messages directly to the incoming communication tapes of the parties. Say that π UC-emulates ϕ with *direct environment* if there exists an adversary $\mathcal{S}$ such that no environment can tell with non-negligible probability whether it interacts directly with π as described here, or alternatively with $\mathcal{S}$ and ϕ in the standard model of executing a protocol (as in Figure 2).

Conceptually, the notion of emulation with respect to direct environments is very close to emulation with respect to dummy adversaries. However, technically speaking the notions are not equivalent. Indeed, while it is easy to see that a protocol π that UC-emulates protocol ϕ with respect to direct environments also UC-emulates ϕ with respect to the dummy adversary, the converse is not necessarily true. In fact, UC-emulation with respect to direct environments is not even reflexive, in the sense that a protocol does not necessarily UC-emulate itself with respect to direct environments. To see that, consider the protocol π where the parties simply send to the adversary any input that they receive. Then, clearly π UC-emulates π with the dummy adversary. However, π does *not* UC-emulate π with respect to direct environments, since the environment can distinguish between a direct execution with a π and an execution with $\mathcal{S}$ and ϕ by simply "starving" $\mathcal{S}$: Assume that the environment gives very long inputs to the parties running the protocol, expects to have these messages copied on its incoming communication tape, and at the same time does not deliver any messages. In the direct execution with π, the environment will always receive the full input that it provided to the parties. However, in the interaction with π and $\mathcal{S}$, the simulator $\mathcal{S}$ receives no input from the environment; thus it is bound to exhaust its running time, and will not be able to write the full inputs on the environment's incoming communication tapes.

We remark that the notion of UC-emulation with respect to direct environments can be relaxed in a natural way to become equivalent to UC-emulation. Specifically, restrict the environment so that at any point throughout the execution the input given to the adversary is longer than the sum of lengths of inputs given to the parties running the protocol.[11]

On the forwarder property and A-PPT adversaries. Using the terminology of Datta et al. [DKMR05], the distinction between UC emulation with respect to the dummy adversary and UC emulation with direct environments (or, *strong simulatability* as they call it) is a manifestation of the fact that the notion of PPT ITMs used here does not satisfy the forwarder property. This property essentially requires that it will be possible to insert a "dummy ITI" (i.e., an ITI which only forwards inputs and messages) between any two communicating ITIs, without changing the properties of the system.

We remark that previous versions of this work, as well as [K05], consider a different notion of polytime-bounded reactive computation for adversaries: While the definition of PPT ITMs requires that the runtime be bounded by a function of the input tape only, the relaxed notion, called A-PPT, allows the runtime to be polynomial in the overall number of bits written on all the incoming tapes, including the input, incoming communication, and subroutine output tapes. In those works, adversaries are allowed to be A-PPT, while all other ITIs (including the environment) are required to be PPT as defined here. One advantage of allowing adversaries to be A-PPT is that, this way, the forwarder property holds in the framework. However, allowing for A-PPT adversaries violates Claim 10, unless one restricts either the environment or the ideal protocol. Furthermore, it can be verified that the present definition of protocol emulation, with respect to PPT adversaries, implies the definition that uses A-PPT adversaries. A simulation based security model that allows for A-PPT adversaries appears in [K05].

4.3.2 Security with respect to black box simulation

Another alternative formulation of Definition 5 imposes the following technical restriction on the simulator $\mathcal{S}$: Instead of allowing a different simulator for any adversary $\mathcal{A}$, 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.

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

- Inputs from the environment are passed through $\mathcal{A}$, then $\hat{\mathcal{S}}$, then to ϕ. That is, upon receiving an input from the environment, $\mathcal{S}$ forwards this input to $\mathcal{A}$. Any outgoing message generated

[11]Previous versions of this work did not make a clear distinction between UC emulation with respect to the dummy adversary and UC emulation with direct environments, implicitly assuming that the two notions are equivalent. We thank Ralf Küsters for pointing out this distinction to us.

by $\mathcal{A}$ is given as input to $\hat{\mathcal{S}}$. Instructions of $\hat{\mathcal{S}}$ regarding delivering messages to the parties of ϕ are carried out.

- Incoming messages from ϕ are passed through $\hat{\mathcal{S}}$, then $\mathcal{A}$, then to the environment. That is, upon receiving an incoming message from a party of ϕ, $\mathcal{S}$ forwards this incoming message to $\hat{\mathcal{S}}$. Outputs of $\hat{\mathcal{S}}$ are forwarded as incoming messages to $\mathcal{A}$, and outputs of $\mathcal{A}$ are outputted by $\mathcal{S}$ to $\mathcal{Z}$.

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

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

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

Proof: The 'only if' direction follows from the definition. For the 'if' direction, notice that the simulator $\mathcal{S}$ in the proof of Claim 10 can be cast as a shell adversary with adversary $\mathcal{A}$ and the following black-box simulator $\hat{\mathcal{S}}$: $\hat{\mathcal{S}}$ runs $\tilde{\mathcal{S}}$, and whenever $\tilde{\mathcal{S}}$ outputs $(\texttt{length}, l)$, $\hat{\mathcal{S}}$ returns input 1^l to $\tilde{\mathcal{S}}$. Furthermore, $\hat{\mathcal{S}}$ does not depend on $\mathcal{A}$. (Indeed, $\hat{\mathcal{S}}$ is not PPT as an ITM by itself. However, as argued in the proof of Claim 10, for any PPT adversary $\mathcal{A}$, the shell simulator $\mathcal{S}$ is PPT with runtime bounded by $p_{\mathcal{A}}(\cdot) + p_{\tilde{\mathcal{S}}}(p_{\mathcal{A}}(\cdot))$.) $\square$

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

Another, more technical difference is the introduction of the shell adversary $\mathcal{S}$. This difference allows us to stay within the model of protocol execution of Section 4.1, which postulates only a single adversary.

The present formulation of black-box simulation is reminiscent of the notions of strong black-box simulation in [DKMR05] and in [PW00] (except for the introduction of the shell adversary). However, in these works this notion is not equivalent to the standard one, due to different formalizations of probabilistic polynomial time.[12]

4.3.3 Letting the simulator depend on the environment

Consider a variant of Definition 5, where the simulator $\mathcal{S}$ can depend on the code of the environment $\mathcal{Z}$. That is, for any $\mathcal{A}$ and $\mathcal{Z}$ there should exist a simulator $\mathcal{S}$ that satisfies (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 5).

[12]Thanks to Ralf Küsters for useful discussions on the formulation of black-box simulation within the UC framework, and for pointing out the inadequacy of some other formulations.

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

Proof: Clearly, if π UC-emulates ϕ as in Definition 5 then UC-emulates ϕ with respect to specialized simulators. To show the other direction, assume that π C emulates ϕ with respect to specialized simulators. That is, for any PPT adversary $\mathcal{A}$ and PPT environment $\mathcal{Z}$ there exists a PPT simulator $\mathcal{S}$ such that (1) holds. Consider the "universal environment" $\mathcal{Z}_u$ which expects its input to consist of $(\langle\mathcal{Z}\rangle, z, 1^t)$, where $\langle\mathcal{Z}\rangle$ is an encoding of an ITM $\mathcal{Z}$, z is an input to $\mathcal{Z}$, and t is a bound on the running time of $\mathcal{Z}$. Then, $\mathcal{Z}_u$ runs $\mathcal{Z}$ on input z for up to t steps, outputs whatever $\mathcal{Z}$ outputs, and halts. Clearly, machine $\mathcal{Z}_u$ is PPT. (in fact, it runs in linear time in its input length). We are thus guaranteed that there exists a simulator $\mathcal{S}$ for $\mathcal{Z}_u$ such that (1) holds. We claim that $\mathcal{S}$ satisfies (1) with respect to *any* PPT environment $\mathcal{Z}$. To see this, fix a PPT machine $\mathcal{Z}$ as in Definition 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 $\textsc{exec}_{\phi,\mathcal{S},\mathcal{Z}}(k,z)$ is identical to the distribution $\textsc{exec}_{\phi,\mathcal{S},\mathcal{Z}_u}(k,z_u)$, where $z_u = (\langle\mathcal{Z}\rangle, z, 1^{c\cdot|z|})$. Similarly, the distribution $\textsc{exec}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ is identical to the distribution $\textsc{exec}_{\pi,\mathcal{A},\mathcal{Z}_u}(k,z_u)$. Consequently, for any $d \in \mathbf{N}$ we have:

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

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, $\textsc{exec}_{\phi,\mathcal{S},\mathcal{Z}} \approx \textsc{exec}_{\pi,\mathcal{A},\mathcal{Z}}$. (Notice that if $|z_u|$ were not polynomial in k then the last derivation would not hold.) $\square$

Remark: Claim 12 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 restrict ITMs to run in time that is bound by a fixed polynomial in the security parameter, standard security and security with respect to specialized simulators end up being different notions (see, e.g., [L03a, HU05]).

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

4.4 Realizing ideal functionalities and hybrid protocols

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

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 ITIs, and it can write to the subroutine output tapes of multiple ITIs. This represents the fact that an ideal functionality behaves like a subroutine machine for a number of different ITIs (which are thought of as parties of some protocol instance). Next, the PID of an ideal functionality is set to $\perp$. (This fact is used to distinguish ideal functionalities from other ITIs.) In addition, an ideal functionality $\mathcal{F}$ expects all inputs to be written by ITIs whose SID is identical to the local SID of $\mathcal{F}$. Other inputs are ignored. The communication tape of $\mathcal{F}$ is used 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. (The intention here is that, in the ideal protocol, the adversary should send corruption messages directly to the ideal functionality. The ideal functionality will then determine the effect of corrupting a party. One typical response would be to let the adversary know some or all of the inputs and outputs the party has received so far. Other, more global responses may include a reduction in the overall security guarantees when more than a certain number of parties have been corrupted, etc. See more discussion in Section 6.1.) 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 23 for a graphic depiction of the ideal protocol.[13]

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

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

[13]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 for 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.

Definition 13 *Let $\mathcal{F}$ be an ideal functionality and let π be an multi-party protocol. We say that π* UC-realizes *$\mathcal{F}$ if π emulates the ideal protocol for $\mathcal{F}$.*

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 π is a protocol that includes subroutine calls to $\text{IDEAL}_{\mathcal{F}}$, the ideal protocol for $\mathcal{F}$.

Recall that running $\text{IDEAL}_{\mathcal{F}}$ means invoking "dummy parties" for $\mathcal{F}$, which in turn invoke an instance of $\mathcal{F}$. This somewhat "indirect" access to $\mathcal{F}$ has the advantage that the calling instance of π has the freedom to specify the SIDs and PIDs for the instances of $\text{IDEAL}_{\mathcal{F}}$ as it wishes. This way, different parties running π 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.

The behavior upon corruption is determined by the protocol, as usual. That is, when either an ITI running π 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.[14]

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

[14] We highlight the main differences between the definition of hybrid protocols and 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 the 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 π 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 10 to the case of hybrid protocols (see Section 4.3). Indeed, in previous versions of this work, Claim 10 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 $\mathcal{F}$ 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 $\mathcal{A}$.

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 $\mathcal{F}$ for 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 $\textsc{uc}()$, is defined as follows. Given a protocol ϕ, a protocol π (that presumably makes subroutine calls to ϕ), and a protocol ρ (that presumably UC-emulates ϕ), the composed protocol $\pi^{\rho/\phi} = \textsc{uc}(\pi, \rho, \phi)$ is identical to protocol π, with the following modifications.

1. Wherever π contains an instruction to pass input x to an ITI running ϕ with identity (sid, pid), then $\pi^{\rho/\phi}$ contains instead an instruction to pass input x to an ITI running ρ with identity (sid, pid).
2. Whenever $\pi^{\rho/\phi}$ receives an output passed from $\rho_{(sid,pid')}$ (i.e., from an ITI running ρ with identity (sid, pid')), it proceeds as π proceeds when it receives an output passed from $\phi_{(sid,pid')}$.

When protocol ϕ is the ideal protocol $\textsc{ideal}_{\mathcal{F}}$ for some ideal functionality $\mathcal{F}$, we denote the composed protocol by $\pi^{\rho/\mathcal{F}}$. Also, when ϕ is understood from the context we use the shorthand π^{ρ} instead for $\pi^{\rho/\phi}$. See a graphical depiction in Figure 4 on page 25.

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

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

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

Theorem statement. We are now ready to state the composition theorem. First we state a general theorem, to be followed by two corollaries. The general formulation makes the following statement: Let π, ϕ, ρ be protocols, such that protocol ρ UC-emulates protocol ϕ as in Definition 5 and both ϕ and ρ are subroutine respecting. Then the protocol $\pi^{\rho/\phi} = \textsc{uc}(\pi, \rho, \phi)$ UC-emulates protocol π. Here all protocols may be hybrid protocols, i.e. they may call the ideal protocol for some ideal functionalities, as long as these ideal functionalities are PPT. (As usual, protocol ρ may in itself be a hybrid protocol, making ideal calls to some ideal functionality $\mathcal{E}$.) A more quantitative statement of the UC theorem is discussed in Section 5.3.[15]

Theorem 14 (Universal composition: General statement) *Let π, ρ, ϕ be PPT multi-party protocols such that ρ UC-emulates ϕ and both ϕ and ρ are subroutine respecting. Then protocol $\pi^{\rho/\phi}$ UC-emulates protocol π.*

As a special case, we get:

Corollary 15 *Let π, ρ be PPT protocols such that ρ UC-realizes a PPT ideal functionality $\mathcal{F}$, and both ϕ and ρ are subroutine respecting. Then protocol $\pi^{\rho/\mathcal{F}}$ UC-emulates protocol π.*

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

Corollary 16 (Universal composition: Realizing functionalities) *Let $\mathcal{F}, \mathcal{G}$ be ideal functionalities such that $\mathcal{F}$ is PPT. Let π be a subroutine respecting protocol that UC-realizes $\mathcal{G}$, and let ρ be a subroutine respecting protocol that securely realizes $\mathcal{F}$. Then the composed protocol $\pi^{\rho/\mathcal{F}}$ securely realizes $\mathcal{G}$.*

Proof: Let $\mathcal{A}$ be an adversary that interacts with parties running $\pi^{\rho/\mathcal{F}}$. Theorem 14 guarantees that there exists an adversary $\mathcal{A}_{\mathcal{F}}$ such that $\text{EXEC}_{\pi,\mathcal{A}_{\mathcal{F}},\mathcal{Z}} \approx \text{EXEC}_{\pi^{\rho/\mathcal{F}},\mathcal{A},\mathcal{Z}}$ for any environment $\mathcal{Z}$. Since π UC-realizes $\mathcal{G}$, there exists a simulator $\mathcal{S}$ such that $\text{IDEAL}_{\mathcal{G},\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A}_{\mathcal{F}},\mathcal{Z}}$ for any $\mathcal{Z}$. Using the transitivity of indistinguishability of ensembles we obtain that $\text{IDEAL}_{\mathcal{G},\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi^{\rho/\mathcal{F}},\mathcal{A},\mathcal{Z}}$ for any environment $\mathcal{Z}$. □

5.2 Proof of the composition theorem

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

5.2.1 Proof outline

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

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

[15]In previous versions of this work Theorem 14 was stated only for the case where protocol ρ is not a hybrid protocol and ϕ is the ideal protocol for some ideal functionality. While the extension to the present formulation is 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).

Also, previous versions did not explicitly make the restriction that π and ρ are mutually subroutine respecting. Rather, it was implicitly assumed that this is the case. I thank Rafael Pass for pointing out this important issue.

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

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

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

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

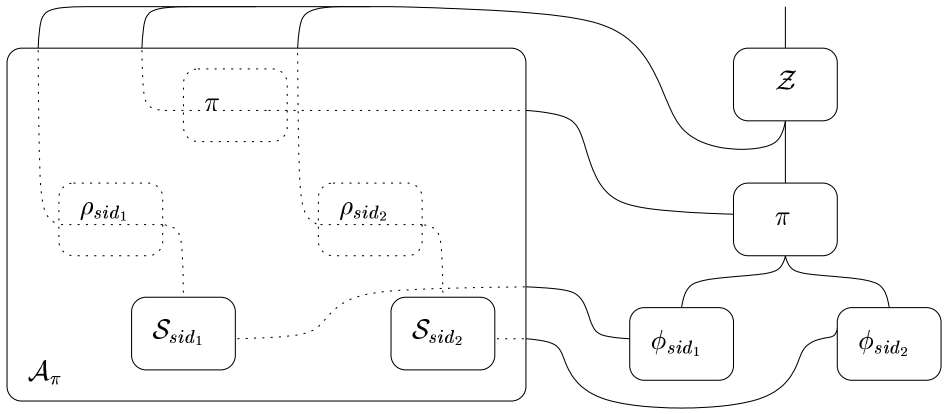

Figure 7: The operation of $\mathcal{A}_\pi$. Inputs from $\mathcal{Z}$ that represent messages of the instance of π are forwarded to the actual instance of π. Inputs directed to an instance of ρ are directed to the corresponding instance of $\mathcal{S}$. Messages from an instance of $\mathcal{S}$ are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

The validity of $\mathcal{A}_\pi$ is demonstrated, based on the validity of $\mathcal{S}$, via a hybrids argument. While the basic logic of the argument is standard, applying the argument to our setting requires some care. We sketch this argument. (The actual argument is slightly more complex; still, this sketch captures the essence of the argument.) Let m be an upper bound on the number of instances of ρ that are invoked in this interaction. Informally, for $l \leq m$ we let π_l denote the protocol where the interaction with the first l instances of ϕ remains unchanged, whereas the rest of the instances of ϕ are replaced with instances of ρ. In particular, protocol π_m is essentially identical to protocol π. Similarly, protocol π_0 is essentially identical to protocol π^{ρ}.[16]

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

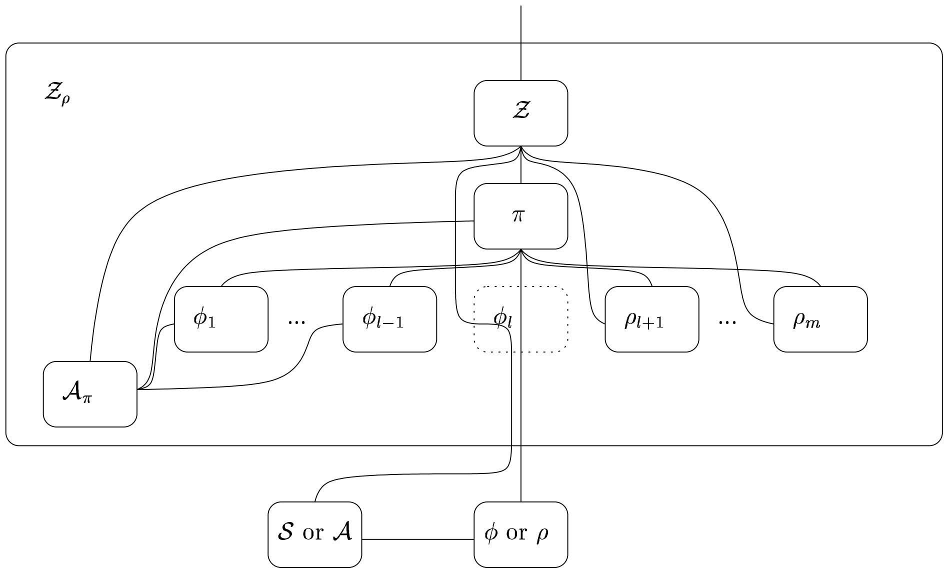

Figure 8: The operation of $\mathcal{Z}_\rho$. An interaction of $\mathcal{Z}$ with π is simulated, so that the first $l-1$ instances of ϕ remain unchanged, the lth instance is mapped to the external execution, and the remaining instances of ϕ are replaced by instances of ρ. For graphical clarity we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

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

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

5.2.2 A detailed proof

We proceed with a detailed proof of Theorem 14, along the lines of the above outline. Most of the terminology and notations were defined in Section 3.

Adversary $\mathcal{A}_\pi$

Adversary $\mathcal{A}_\pi$ proceeds as follows, interacting with parties running protocol π 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 ρ), 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 an ITI (c, id).)
2. When activated with an incoming message m from some party running π, do:
   (a) If m was sent by some party $\phi_{(sid,pid)}$ of protocol ϕ 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 π), then mimic the behavior of the dummy adversary in this case: First pass output $l = |m|+|id|$ to $\mathcal{Z}$; upon receiving input 1^l, pass output (m, id) 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 ϕ, then deliver m to this party. Similarly, when $\mathcal{S}_{(id)}$ requests to pass an output v to its environment then mimic the behavior of the dummy adversary when outputting v to $\mathcal{Z}$.

Figure 9: The adversary for protocol π.

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

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

Validity of $\mathcal{A}_\pi$. First, note that $\mathcal{A}_\pi$ is PPT. In fact, the polynomial $p(\cdot)$ bounding the running time of $\mathcal{A}_\pi$ can be set to be the polynomial bounding the running time of $\mathcal{S}$, plus a linear polynomial. (Note that $p(\cdot)$ does not depend on the number n of instances of ρ. The linear polynomial is needed to take care of relaying the messages of protocol π.) Next, assume that there exists an environment machine $\mathcal{Z}$ that violates the validity of $\mathcal{A}_\pi$ (that is, $\mathcal{Z}$ violates Equation (2)). We construct an environment machine $\mathcal{Z}_\rho$ that violates the validity of $\mathcal{S}$ with respect to a single run of ρ. (That is, $\mathcal{Z}_\rho$ violates Equation (3).) More specifically, fix some input value z and a value k of the security parameter, and assume that

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

We show that

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

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

In preparation to constructing $\mathcal{Z}_\rho$, we define the following distributions, and make some observations on $\mathcal{A}_\pi$. Consider an execution of protocol π with adversary $\mathcal{A}_\pi$ and environment $\mathcal{Z}$. Let $t = t(k,|z|)$ be an upper bound on the number of instances of ϕ within π in this execution. (The bound t is used in the analysis only. The parties need not be aware of t. Also, t is polynomial in $k,|z|$ since $\mathcal{Z}$ is PPT.) For $0 \leq l \leq t$, Let the l-hybrid model for running protocol π denote the extended system of ITMs that is identical to the basic model of computation, with the exception that the control function is modified as follows. (Recall that the control function of an extended system of ITMs determines, among other things, the target ITIs of external-write requests.) The external-write requests to tapes of the first l instances of ϕ to be invoked are treated as usual. The external-write requests to the tapes of all other instances of ϕ are directed to the corresponding instances of parties running ρ. That is, let sid_i denote the SID of the ith instance of ϕ to be invoked in an execution. Then, given an external-write request made by an some ITI to $\phi_{(sid_i,pid)}$ (i.e., to the party running ϕ with identity (sid_i,pid)) for some pid, where $i > l$, the control function writes the requested value to the requested tape of $\rho_{(sid_i,pid)}$. (As usual, if no such ITI 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,ρ) to be delivered to the party running ρ with identity id, then $\mathcal{Z}^{(l)}$ passes input (m,id,c,a) to the adversary, where a is set as follows. Let $id = (sid,pid)$. If sid is the SID of one of the first l instances of ρ, then $\mathcal{Z}^{(l)}$ sets $a = 1$. Otherwise, $a = 0$.

We observe that, when $\hat{\mathcal{A}}_\pi$ interacts with $\mathcal{Z}^{(l)}$ and parties running π in the l-hybrid model, then it internally runs at most l instances of the simulator $\mathcal{S}$. (These are the instances that correspond to the first l instances of protocol ϕ with which $\hat{\mathcal{A}}_\pi$ interacts.) The remaining instances of $\mathcal{S}$ are replaced by interacting with the actual parties or sub-parties of the corresponding instances of ρ. Consequently, we have that the output of $\mathcal{Z}^{(t)}$ from an interaction with π and $\hat{\mathcal{A}}_\pi$ in the t-hybrid

Environment $\mathcal{Z}_\rho$

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

Procedure Simulate(s, l)

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

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

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

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

model is distributed identically to the output of $\mathcal{Z}$ from an interaction with π and $\mathcal{A}_\pi$ in the basic model, i.e. $\text{EXEC}^t_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(t)}} = \text{EXEC}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}$. Similarly, the output of $\mathcal{Z}^{(0)}$ from an interaction with π and $\hat{\mathcal{A}}_\pi$ in the 0-hybrid model is distributed identically to the output of $\mathcal{Z}$ from an interaction with π^ρ in the basic model of computation, i.e. $\text{EXEC}^0_{\pi,\hat{\mathcal{A}}_\pi,\mathcal{Z}^{(0)}} = \text{EXEC}_{\pi,\mathcal{D},\mathcal{Z}}$. Consequently, Inequality (4) can be rewritten as:

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

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

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

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

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

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

and

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

Equations (8) and (9) follow from inspecting the code of $\mathcal{Z}_\rho$ and $\mathcal{A}_\pi$. In particular, if $\mathcal{Z}_\rho$ interacts with parties running ϕ then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ is distributed identically to the view of $\mathcal{Z}$, run within $\mathcal{Z}^{(l)}$, when interacting with π and $\hat{\mathcal{A}}_\pi$ in the l-hybrid model. Similarly, if $\mathcal{Z}_\rho$ interacts with parties running ρ then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ is distributed identically to the view of $\mathcal{Z}$ run within $\mathcal{Z}^{(l)}$, when interacting with π and $\hat{\mathcal{A}}_\pi$ in the $(l-1)$-hybrid model. (Here it is important to note that, since both ρ and ϕ are subroutine respecting, the sub-parties of the external instance of ρ or ϕ do not pass outputs to nor receive inputs from ITIs outside this instance.)

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

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

as desired.

5.3 Discussion and extensions

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

A quantitative statement of the UC theorem. Combining the proof of Theorem 14 with the proof of Claim 10 (equivalence of UC-emulation and UC emulation with respect to the dummy adversary), it is easy to verify that, if ρ (k,ϵ,g)-UC-emulates ϕ for some value k of the security parameter, then protocol $\pi^{\rho/\phi}$ $(k,t\epsilon,g)$-UC-emulates protocol π, where t is a bound on the number of instances of ρ in π^ρ. That is, the emulation slack increases by a factor of t, and the simulation overhead remains the same. To see that the simulation overhead remains unchanged, we note that Proposition 4 (page 33) guarantees that the polynomial bounding the running time of the constructed environment $\mathcal{Z}_\rho$ is no larger than the polynomial bounding the running time of the given environment $\mathcal{Z}$.

On composability with respect to uniform-complexity inputs. Recall that a uniform-complexity variant of the definition of emulation (Definition 5) considers only environments that take external input that contains no information other than its length, e.g. inputs of the form 1^n for some n. We note that the UC theorem still holds even for this 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 14) is stated only for the case of replacing instances of a *single* protocol ϕ 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". Similarly, in the special case where each ϕ_i is an ideal protocol for an ideal functionality $\mathcal{F}_i$, one can write a single ideal functionality that captures all the $\mathcal{F}_i's$.

An alternative proof of the UC theorem. The above proof of Theorem 14 constructs, in a single step, a simulator $\mathcal{A}_\pi$ that handles all the instances of ρ. An alternative proof might proceed as follows:

1. Prove Theorem 14 for the case where protocol π calls only a single instance of ϕ, or equivalently when only one instance of ϕ, out of potentially many instances called by π, is replaced with an instance of ρ. Call this theorem the "single-instance UC theorem". Proving this theorem is considerably simpler than the current proof; in particular, the hybrids argument is not necessary. Furthermore, in this case the UC theorem preserves both the emulation slack ϵ and the simulation overhead g.

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

Here one should take into account the degradation in the quality of the emulation, namely the growth of the emulation slack ϵ and the simulation overhead g. Recall that the emulation slack increases additively under iterative applications of UC-emulation (Section 4.2 on page 44), so the overall slack remains negligible as long as the initial slack is negligible and only a polynomial number of instances are composed. On the other hand, when composing t instances, the overall simulation overhead is the t-wise composition of the original overhead with itself. This means that in general the overhead is guaranteed to be polynomial only if t is constant. However, when there is a polynomial overall bound on the communication of all the instances of ϕ and ρ, Claim 9 guarantees that the overall simulation overhead remains polynomial for any polynomial number of composed protocols. (Still, that this argument is quite wasteful in the concrete value of the simulation overhead, since the current proof guarantees that the overhead remains unchanged, regardless of the number of instances of ϕ.)

5.3.1 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 ρ_1 UC-emulates protocol ϕ_1, and protocol ρ_2 UC-emulates protocol ϕ_2 using calls to ϕ_1, then for any protocol π that uses calls to ϕ_2 it holds that the composed protocol $\pi^{(\rho_2^{\rho_1/\phi_1})/\phi_2} = \textsc{uc}(\pi, \textsc{uc}(\rho_2, \rho_1, \phi_1), \phi_2)$ UC-emulates π.

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. Also, the above alternative proof methodology for the UC theorem applies also to the case of nested protocols. This means that, as long as there is a polynomial overall bound on the communication of all the protocols, Claim 9 guarantees that the overall simulation overhead is polynomial for any polynomial nesting. However, when there is no a priori polynomial bound on the communication of the protocol, the simulation overhead may no longer be polynomially bounded. 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] for their variant of the UC framework.

For simplicity, we state the theorem below for the case where the replaced protocols at all levels are the same (denoted ϕ), and the replacing protocols at all levels are the same (denoted ρ). 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 the depth of the nesting depends only on the security parameter k. This avoids the case where different parties have different values for the "depth of nesting". That is, for protocols ρ and ϕ where ρ is a ϕ-hybrid protocol, let $\rho^{\langle 0\rangle} = \rho$, and let $\rho^{\langle i+1\rangle} = \textsc{uc}(\rho^{\langle i\rangle}, \rho, \phi)$. Then we have:

Theorem 17 (Universal composition: polynomial nesting) *Let $t : \mathbf{N} \to \mathbf{N}$ be a polynomial. Let ϕ be a protocol, and let ρ be a ϕ-hybrid protocol that UC-emulates ϕ. Then protocol $\rho^{\langle t(k)\rangle}$ UC-emulates ϕ.*

Proof (sketch): The proof is a straightforward extension of the proof of the UC theorem (Theorem 14). Here we only sketch these extensions. As there, we use security with respect to dummy adversaries (see Claim 10). That is, we construct an adversary $\mathcal{A}^*$ and show that $\textsc{exec}_{\rho,\mathcal{A}^*,\mathcal{Z}} \approx \textsc{exec}_{\rho^{\langle t\rangle},\mathcal{D},\mathcal{Z}}$ for all environments $\mathcal{Z}$.

Let $\mathcal{A}$ be the simulator, guaranteed from the fact that ρ UC-emulates ϕ, such that $\textsc{exec}_{\phi,\mathcal{A},\mathcal{Z}} \approx \textsc{exec}_{\rho,\mathcal{D},\mathcal{Z}}$ for all $\mathcal{Z}$. The instance of ρ 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 $\mathcal{A}^*$ is a natural extension of adversary $\mathcal{A}_\pi$ from the proof of Theorem 14, where the top level instance of ρ plays the role of protocol π. That is, $\mathcal{A}^*$ internally runs an instance of $\mathcal{S}$ for each instance of ϕ that is replaced by an instance of ρ 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 ρ 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 ρ 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 ρ are forwarded to $\mathcal{Z}$.

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

4. Messages sent by an instance of $\mathcal{S}$ to the corresponding instance of ϕ, where this instance is at depth greater than 1, are forwarded to the instance of $\mathcal{S}$ that corresponds to the instance of ρ invoked this instance of ϕ. (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 ϕ are forwarded to the instance of $\mathcal{S}$ that simulates the instance of ρ that replaces this instance of ϕ. (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 $\mathcal{A}^*$ is practically identical to the analysis of $\mathcal{A}_\pi$ at the proof of Theorem 14. First, we note that $\mathcal{A}^*$ is PPT. This is so since the overall number of instances of $\mathcal{S}$'s run by $\mathcal{A}^*$ is polynomial, each instance is PPT, and the input length of each instance is polynomial in the input length of $\mathcal{A}^*$. (Notice that there is no "nesting" in the calls to the instances of $\mathcal{S}$'s. That is, $\mathcal{A}^*$ directly calls all the instances of $\mathcal{S}$.) Now, assume that there is an environment $\mathcal{Z}$ such that $\text{EXEC}_{\rho^{\langle t\rangle},\mathcal{D},\mathcal{Z}} \not\approx \text{EXEC}_{\rho,\mathcal{A}^*,\mathcal{Z}}$. Then we define the hybrid environments $\mathcal{Z}^{(j)}$ and construct the distinguishing environment $\mathcal{Z}_\rho$ in exactly the same way as there. Also, as there, the number of hybrids is the overall number of instances of all the instances of ρ. This number may be considerably larger than $t(k)$; still, it is polynomial. $\square$

5.3.2 Universal composition with joint state

Informally speaking, the UC theorem implies that if a protocol ϕ UC-realizes some functionality $\mathcal{G}$ then multiple concurrent instances of ϕ UC realize multiple concurrent 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 instances of ϕ have mutually disjoint local states and local randomness. That is, all the sub-parties of an instance of ϕ must pass or receive inputs and outputs only with parties an sub-parties of this instance. In contrast, in many cases we have a system where multiple concurrent instances of some protocol ϕ use the same instance of an underlying subroutine, ρ. 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. In these cases it is impossible to "de-compose" the system into protocol instances with disjoint local states; thus the UC theorem cannot be directly applied to deduce the security of the system from the security of a single instance.

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 the following. Let γ be a protocol that UC-realizes functionality $\mathcal{G}$ using an ideal functionality $\mathcal{F}$, and let ρ be a protocol that UC realizes, within a single instance, multiple instances of $\mathcal{F}$. Then the protocol that consists of multiple concurrent instances of γ, where all the calls to $\mathcal{F}$ made by all instances of γ are replaced by calls to a single instance of ρ, UC-realizes multiple instances of $\mathcal{G}$. See Figure 11 for a graphic depiction.

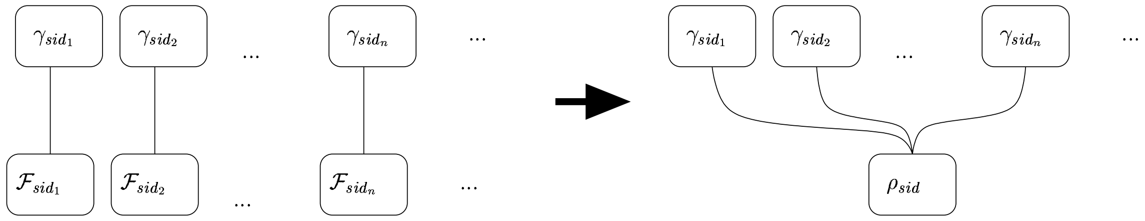

Figure 11: Universal composition with joint state. The instances of γ (in the left figure) are analyzed assuming that each instance has access to a separate instance of $\mathcal{F}$. Later, all instances of $\mathcal{F}$ are replaced by a single instance of ρ (right figure).

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.[17] We start with an $\mathcal{F}$-hybrid protocol π, and a protocol ρ that UC-realizes $\hat{\mathcal{F}}$. (Using the above terminology, protocol π represents the multi-instance version of protocol γ.) Then the composed protocol $\pi^{[\rho/\mathcal{F}]} = \text{JUC}(\pi, \mathcal{F}, \rho)$ is identical to π with the exception that, at the onset of the computation, $\pi^{[\rho/\mathcal{F}]}$ instructs each party to invoke a instance of ρ 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 ρ, and each output $(sid, ssid, v)$ of the instance sid of ρ 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 π. That is:

Theorem 18 ([CR03]) *Let $\mathcal{F}$ be an ideal functionality, let π be an $\mathcal{F}$-hybrid protocol, let ρ 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 π.*

Let us exemplify the use of the JUC theorem when 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:

[17]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.

- Capture the functionality expected from a single instance of a key-exchange protocol via an ideal functionality, $\mathcal{F}_{\text{KE}}$.
- Capture the behavior of a single instance of a signature scheme via a functionality, $\mathcal{F}_{\text{SIG}}$.
- Construct a protocol, ρ, 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.
- Construct a protocol, γ, that realizes $\mathcal{F}_{\text{KE}}$ in the $\mathcal{F}_{\text{SIG}}$-hybrid model. It is stressed that each instance of protocol γ 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.
- Let π denote the protocol that consists of multiple independent instance of γ. Then use the JUC theorem to conclude that protocol $\pi^{[\rho/\mathcal{F}_{\text{SIG}}]}$ emulates protocol π. 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 γ within π use the same joint instance of protocol ρ. In fact, notice that protocol π 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].

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

Proposition 19 *Let $\mathcal{C}$ be a self-simulatable class of ITMs, and let π, ρ, ϕ be multi-party protocols in $\mathcal{C}$ such that ρ UC-emulates ϕ with respect to class $\mathcal{C}$. Then protocol $\pi^{\rho/\phi}$ UC-emulates protocol π with respect to class $\mathcal{C}$.*

Furthermore, the UC theorem holds with respect to statistical and perfect emulation. That is, if ρ statistically (resp., perfectly) UC-emulates ϕ with respect to class $\mathcal{C}$ of adversaries then $\pi^{\rho/\phi}$ statistically (resp., perfectly) UC-emulates protocol π with respect to class $\mathcal{C}$. (Note that these statements do not follow from Proposition 19, 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 from the same class. We exemplify this point for the case where all entities in the system are bound to be PPT, except for the protocol ϕ which is not

PPT.[18] More specifically, we present an ideal functionality $\mathcal{F}$ that is not PPT, and a PPT protocol ρ that UC-realizes $\mathcal{F}$ with respect to PPT environments. Then we present an $\mathcal{F}$-hybrid protocol π, that uses only *two* instances of the ideal protocol for $\mathcal{F}$, and such that $\pi^{\rho/\mathcal{F}}$ does not emulate π. In fact, for *any* PPT ρ' we have that $\pi^{\rho'/\mathcal{F}}$ does not emulate π.

In order to define $\mathcal{F}$, we first recall the definition of pseudorandom ensembles of evasive set, defined in [GK89] for a related purpose. An ensemble $\mathcal{S} = \{S_k\}_{k \in \mathbf{N}}$ where each $S_k = \{s_{k,i}\}_{i \in [2^k]}$ 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 $x \overset{\text{R}}{\leftarrow} s_{k,i}$ is computationally indistinguishable from $x \overset{\text{R}}{\leftarrow} \{0,1\}^k$. (b) $\mathcal{S}$ is evasive, that is for any PPT algorithm A and for any $z \in \{0,1\}^*$, we have that $\text{Prob}_{i \overset{\text{R}}{\leftarrow} [2^k]}[A(z,i) \in s_{k,i}]$ 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 $i \overset{\text{R}}{\leftarrow} [2^k]$ 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 $r \overset{\text{R}}{\leftarrow} s_{k,i'}$.

Protocol ρ for realizing $\mathcal{F}$ is simple: Given security parameter k it outputs $i \overset{\text{R}}{\leftarrow} [2^k]$. Given an input $x \in \{0,1\}^k$, it outputs $r \overset{\text{R}}{\leftarrow} \{0,1\}^k$. It is easy to see that ρ UC-realizes $\mathcal{F}$: Since $\mathcal{S}$ is evasive, then the probability that the input x is in the set $s_{k,i}$ is negligible, thus $\mathcal{F}$ outputs `success` only with negligible probability. Furthermore, $\mathcal{F}$ outputs a pseudorandom k-bit value, which is indistinguishable from the output of ρ.

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

6 UC formulations of some computational models

Definition 13 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 Sections 1.2 and 2.2.)

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 hy-

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

brid 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 security goals 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 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.) Next we capture a number of set-up assumptions, such as the common reference string model and several variants of key registration and set-up. 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 Writing conventions for ideal functionalities

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

Also, it will often be convenient to include within the SID the identity of the initiator (namely, of the party that invokes $\mathcal{F}$), or alternatively of all participants. Including the initiator's identity in the SID has the advantage that it helps guarantee global uniqueness of the SID, as long as the rest of the SID is locally unique for the initiator. Including the intended identities of other participants provides a simple mechanism for parties to know whether they should participate in this protocol instance. (Of course, this method is applicable only when the initiator knows the identities of the other intended participants in advance.) Note however that the SID is typically a public value, thus if it is desired that the identities of the participants be kept secret then some aliasing mechanism should be used (say, the SID includes a list of aliases and the functionality is privately given the identity that corresponds to each alias).[19]

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

[19]An alternative method for the initiator to guarantee that it invokes a "fresh" instance of $\mathcal{F}$ is to make sure that the initiator's identity (and potentially also the identities of all other participants) is included explicitly as parameters in the code of $\mathcal{F}$. This method of specifying the identities has the advantage that it facilitates keeping the identities secret. Still, for the purpose of the work we adopt the technique of including the identities in the SID of $\mathcal{F}$ since it makes the binding between the instances of $\mathcal{F}$ and the identities of participants more explicit.

functionality upon receiving such a message is not determined by the model. We say that an ideal functionality $\mathcal{F}$ is standard corruption if, upon receiving a `(corrupt` P`)` message from the adversary $\mathcal{S}$, $\mathcal{F}$ first locally records P as corrupted, and then (i.e., in the next activation) returns to the adversary all the inputs and outputs of P so far. In addition, from this point on, whenever $\mathcal{F}$ gets an input value v from P, it forwards v to the adversary, and receives a "modified input value" v' from the adversary. Also, all output values intended for P are sent to the adversary instead. This behavior captures the standard behavior of the ideal process upon corruption of a party in existing definitional frameworks. All functionalities presented in this work are standard-corruption unless explicitly said otherwise.[20]

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. If the output is public, then the value v is included in the note to the adversary. If the output is private then v is not not mentioned in this note. Furthermore, the note contains a unique identifier that distinguishes it from all other messages sent by $\mathcal{F}$ to the adversary in this execution. When the adversary replies to the note (say, by echoing the unique identifier), $\mathcal{F}$ outputs the value v to P.

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

Running arbitrary code. It is often convenient to let an ideal functionality $\mathcal{F}$ receive a description of an arbitrary code c from parties, or even directly from the adversary, and then run this code while inspecting some properties of it. One use of such technique is for writing ideal functionalities with only minimal, well-specified requirements from the implementation. For instance, $\mathcal{F}$ may receive (say, from the adversary) a code for an algorithm and will run this algorithm as long as some set of security or correctness properties are satisfied. (If a required property is violated, $\mathcal{F}$ will output an error message.) Examples where this technique is used are the signature and encryption functionalities, presented in Section 7.2. Another use for this "programming technique" is for $\mathcal{F}$ to avoid determining some output value, while guaranteeing that the adversary does not know this value. Yet another use is for restricting the communication between the adversary and the environment (See Section 6.5).

At first glance, this technique seems problematic in that $\mathcal{F}$ is expected to run algorithms of arbitrary polynomial running time, whereas its runtime is bounded by some fixed polynomial. We get around this problem by assuming that the entity that provides the code c explicitly invokes an ITI that runs c, while taking inputs and providing outputs to $\mathcal{F}$. This way, the "burden of providing sufficient resources to run c" is shifted to the entity that provides c. Note however that

[20]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.

the length of the outputs of c should be bound by the polynomial bounding $\mathcal{F}$, otherwise $\mathcal{F}$ will not be able to read these outputs. Also, recall that the basic model allows $\mathcal{F}$ to know the real code run by this ITI. In particular, $\mathcal{F}$ can make sure that c does not provide any illegitimate output to parties other than $\mathcal{F}$.

6.2 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 assume ideally authenticated message transmission can be cast as hybrid protocols with ideal access to an "authenticated message transmission functionality". This functionality, denoted $\mathcal{F}_{\text{AUTH}}$, is presented in Figure 12. $\mathcal{F}_{\text{AUTH}}$ first waits to receive an input $(\texttt{Send}, sid, m)$ from a party with PID S and SID sid. We require that the PIDs of the sender and the receiver are encoded in the SID. (As discussed above, this convention essentially provides each party with a "reserved name-space" for SIDs of instances of $\mathcal{F}_{\text{AUTH}}$ where it is the sender. It also allows the receiver to verify that it is the intended receiver, and guarantees that the receiver identity is fixed in advance, even when the sender is corrupted.) $\mathcal{F}_{\text{AUTH}}$ then generates a public delayed output $(\texttt{Send}, sid, m)$ to R. That is, $\mathcal{F}_{\text{AUTH}}$ first sends this value to the adversary. When the adversary responds, $\mathcal{F}_{\text{AUTH}}$ writes this value to the subroutine output tape of R. In addition, if the adversary instructs to corrupt the sender before the output value was actually delivered to R, then $\mathcal{F}_{\text{AUTH}}$ allows the adversary to provide new, arbitrary message m'. That , $\mathcal{F}_{\text{AUTH}}$ writes $(\texttt{Send}, sid, m')$ to R, and halts. Corrupting the receiver has no effect.

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

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

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

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, $\mathcal{F}_{\text{AUTH}}$ 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. Fourth, $\mathcal{F}_{\text{AUTH}}$ allows the adversary to change the contents of the message, as long as the sender is corrupted *at the time of delivery,* even if the sender was uncorrupted at the point when it sent the message. This provision captures the fact that in general the received value is not determined until the point where the recipient actually generates its output. [21] Finally, notice that $\mathcal{F}_{\text{AUTH}}$ generates an output for the receiver without requiring the receiver to provide any input. This means that the SID is determined exclusively by the sender, and there is no need to agree on the SID in advance.

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

Claim 20 ([C04]) *There exist no useful protocols that UC-realize $\mathcal{F}_{\text{AUTH}}$ in the bare model.*

Still, there are a number of ways to realize $\mathcal{F}_{\text{AUTH}}$ given some standard set-up assumptions. Examples include ideally authenticated communication between parties at a preliminary stage, or alternatively a trusted "registration service", where parties can register public values (e.g., keys) that can be authentically obtained by other parties upon request. The latter assumption is captured by ideal functionality, $\mathcal{F}_{\text{REG}}$, presented in Figure 13.

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

1. Upon receiving input $(\texttt{Register}, sid, v)$ verify that $sid = (P, sid')$. If sid is not of that form, or this is not the first input form P, then ignore this input. Else, send $(\texttt{Registered}, sid, v)$ to the adversary and record the value v.
2. Upon receiving input $(\texttt{Retrieve}, sid)$ from party P', send a delayed output $(\texttt{Retrieve}, sid, v)$ to P'. (If no value v is recorded then set $v = \perp$.)

Figure 13: The ideal registration functionality, $\mathcal{F}_{\text{REG}}$

For simplicity, $\mathcal{F}_{\text{REG}}$ is formulated for registering a single public value. Different values can be registered via different instances of $\mathcal{F}_{\text{REG}}$. Also, it is stressed that F_{REG} does not perform any checks on the registered value; it simply acts as a public bulletin board. (In particular, no "proof of possession of secret key" is required.) Consequently, when running in the $\mathcal{F}_{\text{REG}}$-hybrid model, a party can register with some instance of $\mathcal{F}_{\text{REG}}$ using, e.g., the same public value as that of some other party, in another instance of $\mathcal{F}_{\text{REG}}$. Still, the present minimal formulation suffices for realizing $\mathcal{F}_{\text{CERT}}$ and subsequently $\mathcal{F}_{\text{AUTH}}$.[22]

In [C04] 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]), plus access to $\mathcal{F}_{\text{REG}}$. That is:

Claim 21 ([C04]) *If there exist signature schemes secure against chosen message attacks then there exists an $\mathcal{F}_{\text{REG}}$-hybrid protocol that UC-realizes $\mathcal{F}_{\text{AUTH}}$.*

[21] Previous formulations of $\mathcal{F}_{\text{AUTH}}$ (e.g., [C01]) failed to let the adversary change the delivered message and recipient identity 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].

[22] Formally speaking, the UC theorem does not apply to an $\mathcal{F}_{\text{REG}}$-hybrid protocol, since such a protocol is not subroutine respecting, in the sense that it contains a subroutine (namely, $\mathcal{F}_{\text{REG}}$) that provides outputs to ITIs which are not part of the specific protocol instance. In [DPW05] the UC theorem is extended to handle this case. See more discussion in Section 6.6.

We remark that in [C04] the claim is stated and proved in terms of protocols that use the ideal signature functionality, $\mathcal{F}_{\text{SIG}}$, rather than unforgeable signature schemes. See more details there and in Section 7.2.1. Also, 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 π (that assumes authenticated communication) corresponds to composing π with a protocol ρ that UC-realizes $\mathcal{F}_{\text{AUTH}}$. In fact, the protocol used in the proof of Claim 21 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 instances of a protocol that realizes a single instance of $\mathcal{F}_{\text{AUTH}}$. First, using JUC, it is possible to realize multiple instances of $\mathcal{F}_{\text{AUTH}}$ using a single instance of a signature scheme per party. 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 7.1.1 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 more generally 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 14. The behavior of $\mathcal{F}_{\text{SMT}}$ is similar to that of $\mathcal{F}_{\text{AUTH}}$ with the following exception. $\mathcal{F}_{\text{SMT}}$ is parameterized by a leakage function $l : \{0,1\}^* \to \{0,1\}^*$ that captures the allowed information leakage on the transmitted plaintext m. That is, the adversary only learns the leakable information $l(m)$ rather than the entire m. (In fact, $\mathcal{F}_{\text{AUTH}}$ can be regarded as the special case of $\mathcal{F}^l_{\text{SMT}}$ where l is the identity function.)

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

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

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

Figure 14: 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 π_E. When invoked within S with input $(\texttt{Send}, sid, R, m)$ with $m \in D$, π_E first sends an $(\texttt{init}, sid)$ message to R. R then runs algorithm gen, gets the secret key sk and the public key pk, and sends (sid, pk) back to S. Next, S sends $(sid, enc(pk, m))$ to R and returns. Finally, upon receipt of (sid, c), π_E within R outputs $dec(sk, c)$ and returns.

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 it is sufficient for realizing $\mathcal{F}_{\text{SMT}}$. given a domain D of plaintexts, let l_D be the "leakage function" that, given input x, returns $\perp$ if $x \in D$ and returns x otherwise. Then:

Claim 22 *If E is semantically secure for domain D as in* [GM84, G01] *then π_E UC realizes $\mathcal{F}_{\text{SMT}}^{l_D}$ in the presence of non-adaptive adversaries.*

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

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 further discussion ins Section 7.2.2 and [N02, CHK05].

In more efficient realizations of $\mathcal{F}_{\text{SMT}}$ in the case of a long conversation among two parties, 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 a single instance of 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 common and convenient abstraction of communication networks is that of *synchronous* communication. Roughly speaking, here the computation proceeds in *rounds,* where in each round each party receives all the messages that were sent to it in the previous round, and generates outgoing messages for the next round. There are of course many variants of the synchronous model. We concentrate here on modeling a basic variant, which provides the following guarantee:

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

This guarantee implies in essence two other guarantees:

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

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

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

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

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

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

Figure 15: The synchronous communication functionality, $\mathcal{F}_{\text{SYN}}$.

Synchronous variants of the UC framework are presented in [N03, HM04a]. Here we provide an alternative way of capturing synchronous communication within the UC framework: We show how synchronous communication can be captured by having access to an ideal functionality $\mathcal{F}_{\text{SYN}}$ that provides the above guarantees. We first describe $\mathcal{F}_{\text{SYN}}$, and then discuss and motivate some aspects of its design. $\mathcal{F}_{\text{SYN}}$ is presented in Figure 15. It expects its SID to include a list $\mathcal{P}$ of parties among which synchronization is to be provided. At the first activation, $\mathcal{F}_{\text{SYN}}$ initializes a round number r to 1, and sends a delayed public output `init` to all parties in $\mathcal{P}$. (The purpose of this output is to notify the parties in $\mathcal{P}$ that a synchronous session with the given SID has started; in particular, it frees the calling protocol from the need to agree on the SID in advance.) Next, $\mathcal{F}_{\text{SYN}}$ responds to three types of inputs: Given input of the form $(\texttt{Send}, sid, M)$ from party $P \in \mathcal{P}$, $\mathcal{F}_{\text{SYN}}$ interprets M as a list of messages to be sent to other parties in $\mathcal{P}$. The list M is recorded together with the sender identity and the current round number, and is also forwarded to the adversary. Given a request $(\texttt{Advance-Round}, sid, N)$ from the adversary, where N is a list of messages sent from the corrupted parties to the uncorrupted ones, $\mathcal{F}_{\text{SYN}}$ first verifies that all uncorrupted parties sent messages in this round. If yes, then $\mathcal{F}_{\text{SYN}}$ advances the round number. If not, then the request is ignored. This provision guarantees that a round does not end before all parties have a chance to send messages. Upon receiving a $(\texttt{Corrupt}, P)$ from the adversary, for some $P \in \mathcal{P}$, $\mathcal{F}_{\text{SYN}}$ marks P as corrupted. If P gets corrupted after it has sent its messages for this round, but before the round advances, then at this time the adversary has the opportunity to "rewrite" the messages that were sent by P at this round. In any case, once the round number advances, the set of messages sent at 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. 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.[23]

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

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

[23]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.

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

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.

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

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

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

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

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

A priori, this observation may seem to prevent the use of the universal composition operation for constructing such protocols, since this operation does not allow the composed protocols to have any joint subroutines. We note, however, that protocol instances that use the same instance of $\mathcal{F}_{\text{SYN}}$ can be composed using universal composition with joint state (see Section 5.3). That is, first analyze each such protocol instance as if it is the only one using $\mathcal{F}_{\text{SYN}}$. Next, observe that one can straightforwardly realize multiple instances of $\mathcal{F}_{\text{SYN}}$ using a single instance of $\mathcal{F}_{\text{SYN}}$, whose set of participants is the union of the sets of participants of the realized instances. The JUC theorem now states that letting all the calling protocol instances use the same instance of $\mathcal{F}_{\text{SYN}}$ does not change the security. In other words, there exists a straightforward $\mathcal{F}_{\text{SYN}}$-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 one 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 16. It first expects to receive a code of an adversary, $\hat{\mathcal{A}}$. It then behaves as adversary $\hat{\mathcal{A}}$ would in the non-concurrent security model. That is, $\mathcal{F}_{\text{NC}}$ runs $\hat{\mathcal{A}}$ and follows its instructions with respect to receipt and delivery of messages between parties. As soon as $\hat{\mathcal{A}}$ terminates the execution, $\mathcal{F}_{\text{NC}}$ reports the current state of $\hat{\mathcal{A}}$ back to the external adversary, and halts. (Recall the convention from Section 6.1 regarding running arbitrary code.)

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

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

Figure 16: The non-concurrent communication functionality, $\mathcal{F}_{\text{NC}}$.

We remark that, in addition to analyzing "pure" non-concurrent security of protocols, $\mathcal{F}_{\text{NC}}$ can also be used to analyze systems where some of the components may be running concurrently with each other, where other components cannot. Here the "non-composable" components can be written as $\mathcal{F}_{\text{NC}}$-hybrid protocols. For some concrete examples see Section 7.3.2.

Non-concurrent security of protocols that assume some idealized communication model (such as, say, authenticated or synchronous communication) can be captures by using $\mathcal{F}_{\text{NC}}$ in conjunction with the ideal functionality that captures the desired model. For instance, to capture non-concurrent security of synchronous protocols, let the protocol use $\mathcal{F}_{\text{SYN}}$ as described in Section 6.4, with the exception that $\mathcal{F}_{\text{SYN}}$ interacts with $\mathcal{F}_{\text{NC}}$ instead of interacting directly with the adversary. ($\mathcal{F}_{\text{NC}}$, in turn, interacts with the adversary as described above.) We call such protocols $\mathcal{F}_{\text{NC}}$-$\mathcal{F}_{\text{SYN}}$-hybrid protocols.

Equivalence with the definition of [C00]**.** Recall the security definition of [C00], that addresses secure function evaluation in synchronous communication networks, and guarantees that security is preserved under non-concurrent composition of protocols. (See discussion in Section 1.1.) More specifically, recall that the [C00] notion, which we call non-concurrent security, is the same as UC security for $\mathcal{F}_{\text{SYN}}$-hybrid protocols 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 23 *Let π and ϕ be PPT multi-party protocols. We say that π* NC-emulates *ϕ if for any PPT adversary $\mathcal{A}$ there exists a PPT adversary $\mathcal{S}$ such that for any 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, even with authenticated communication.

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, let π be an $\mathcal{F}_{\text{NC}}$-$\mathcal{F}_{\text{SYN}}$-hybrid protocol; that is, the parties of π send messages only via $\mathcal{F}_{\text{SYN}}$, which in turn interacts with $\mathcal{F}_{\text{NC}}$ instead of with the adversary. Furthermore, π is a function evaluation protocol, namely each party generates only a single output, and all outputs are generated in the same round. Then π UC-realizes some ideal functionality $\mathcal{F}$ if and only if π realizes $\mathcal{F}$ with non-concurrent security. More generally:

Proposition 24 *Let π be a function evaluation, $\mathcal{F}_{\text{NC}}$-$\mathcal{F}_{\text{SYN}}$-hybrid protocol. Then π UC-emulates protocol ϕ if and only if π NC-emulates ϕ.*

Notice that Proposition 24, together with the UC theorem, provides an alternative (albeit somewhat roundabout) formulation of the non-concurrent composition theorem of [C00].

6.6 The Common Reference String and Key Set-Up models

It is often convenient to design and analyze protocols in models that provide some "idealized trust" in certain computational entities, or alternatively in physical phenomena. Such trust is often referred to as "cryptographic set-up assumptions". Indeed, in general only very weak security guarantees can be provided with no set-up assumptions whatsoever.

One basic set-up assumption is formulated in $\mathcal{F}_{\text{REG}}$ (Figure 13 in Section 6.2), which captures the basic functionality of a trusted "bulletin board" where parties can register public values to be associated with their identities. Here we consider two other useful set-up assumptions, the common reference string (CRS) model and the key set-up model. In fact, the key set-up model is a strict relaxation of the CRS model; still, it seems instructive to present and discuss them separately. It is also instructive to contrast these functionalities with $\mathcal{F}_{\text{REG}}$ and observe that they provide *incomparable* levels of trust.

The CRS model. The common reference string model, first proposed in [BFM89] and used extensively since, assumes that the parties have access to a common string that is guaranteed to come from a pre-specified distribution. Furthermore, it is guaranteed that the string was chosen in an "opaque" way, namely that no information related to the process of choosing this string is available to any party. A very natural distribution for the common string, advocated in [BFM89], is the uniform distribution over the strings of some length. Indeed, many physical phenomena readily yield such "opaquely chosen" strings that are believed to be uniformly distributed. Still, it is often useful to consider reference strings that are taken from other distributions.

In the Zero-Knowledge context of [BFM89], the fact that the reference string comes from an external source that is unrelated to the actual computation is captured by allowing the simulator choose the reference string as it wishes — as long as the adversary cannot distinguish this "simulated string" from a "real string" taken from the prescribed distribution. Indeed, it is this extra freedom given to the simulator which makes this model powerful.

Within the present framework, the CRS model can be captured in a natural way by modeling the reference string as coming from an appropriate ideal functionality. More specifically, we formulate functionality $\mathcal{F}_{\text{CRS}}$, presented in Figure 17 below. The functionality is parameterized by a distribution D. It expects the set $\mathcal{P}$ of recipients of the reference string to be encoded in its SID, and ignores inputs from parties not in that set. Having received input from a party $P \in \mathcal{P}$, $\mathcal{F}_{\text{CRS}}^{D}$ generates a public delayed output r to P, where r is a value that was drawn from distribution D in the first activation and fixed for the rest of the interaction.

Letting the adversary know r models the fact that r is public, and cannot be assumed secret. Prohibiting parties not in $\mathcal{P}$ from obtaining r directly from $\mathcal{F}_{\text{CRS}}$ models the fact that r is treated as local to a specific protocol instance, and is intended to be used only within this protocol instance. Other protocol instances should use different "draws" from distribution D.

Functionality $\mathcal{F}_{\text{CRS}}^{D}$

$\mathcal{F}_{\text{CRS}}^{D}$ is parameterized by distribution D. It proceeds as follows.

1. When receiving input $(\texttt{CRS}, sid)$ from party P, first verify that $sid = (\mathcal{P}, sid')$ where $\mathcal{P}$ is a set of identities, and that $P \in \mathcal{P}$; else ignore the input. Next, if there is no value r recorded then choose and record a value $r \xleftarrow{\text{R}} D$. Finally, send a public delayed output $(\texttt{CRS}, sid, r)$ to P.

Figure 17: The Common Reference String functionality

We note that restricting the set of recipients of the CRS stands in contrast with the prevalent intuitive perception of the CRS as a single string that is set in advance and available to all. In particular, the present modeling prevents the participants of a protocol execution that uses a reference string from using that string to prove claims regarding the execution to parties that did not take part in that execution. In contrast, such proofs are possible in a setting where the reference string is known to all via means that are independent of the specific protocol execution. Indeed, as shown in [DPW05], this "globally available CRS" model gives weaker guarantees to protocols that use it, and does not suffice for many of the applications of $\mathcal{F}_{\text{CRS}}$. (In essence, this stronger notion forces the adversary of the ideal protocol to use a pre-determined reference string, rather than generating a reference string by itself.) In addition, the remark in Footnote 22 on page 70 applies here as well.

It is shown in [CLOS02] that, under standard cryptographic assumptions, any ideal functionality can be realized by $\mathcal{F}_{\text{CRS}}^{D}$-hybrid protocols, where D is the uniform distribution. On the other hand, it is shown in [CF01, CKL03] that realizing $\mathcal{F}_{\text{CRS}}^{D}$ is impossible under Byzantine corruptions of a majority of the parties in the "eligibility set" $\mathcal{P}$, and D is non-trivial. Furthermore, this holds even if authenticated communication is available. That is, say that a distribution ensemble $D = \{D_k\}_{k\in\mathrm{N}}$ is non-trivial if there exists a constant c such that no element v in the support of D_k has probability more than $1 - k^{-c}$. Then:

Claim 25 ([CF01, CKL03]) *Let D be a non-trivial distribution ensemble. Then there exist no useful $\mathcal{F}_{\text{AUTH}}$-hybrid protocols that allow for Byzantine corruptions and UC-realize $\mathcal{F}_{\text{CRS}}^{D}$. Furthermore, this holds even if the adversary is restricted to corrupting only a fraction of the parties in the eligibility set $\mathcal{P}$, as long as this fraction is at least one half.*

The key set-up model. The CRS set-up assumption has the advantage that it only requires knowledge of a single short string. In particular, in contrast with $\mathcal{F}_{\text{REG}}$, it does not require parties to identify themselves or to go through a registration process before participating in a protocol. Thus, in settings where it is reasonable to assume existence of trusted reference string, this assumption is very attractive. However, when the reference string is being generated by a computational entity that may be corrupted or subverted, the CRS modeling is somewhat unsatisfactory, in that it puts complete trust in a single entity. In fact, this entity, if subverted, can completely undermine the security of the protocol by choosing the reference string from a different distribution, or alternatively by leaking to some parties some secret information related to the string. Furthermore, it can do so without being detected.

As shown in [BCNP04], the CRS set-up assumption can be relaxed in a way that minimizes the trust in any single entity, while still providing comparable power and convenience to protocols that use it. As a first step towards presenting the [BCNP04] set-up assumption, consider the following ideal functionality $\mathcal{F}_{\text{KRK}}$ (for "key registration with knowledge"), which is a strengthening of $\mathcal{F}_{\text{REG}}$: $\mathcal{F}_{\text{KRK}}$ is parameterized by a function f, that transforms private keys into public keys. (For instance, given two primes p, q, f might return $n = pq$.) Now, when a party provides a value v to $\mathcal{F}_{\text{KRK}}$, the registered value is $f(v)$. The value v is discarded and never disclosed. This in essence makes sure that a public value is registered only if the registering party "knows" an associated secret value.

On the one hand, $\mathcal{F}_{\text{KRK}}$ can be seen as a strengthening of $\mathcal{F}_{\text{REG}}$, in that it requires knowledge of a secret associated with the registered value. On the other hand, $\mathcal{F}_{\text{KRK}}$ essentially "distributes" the trust among multiple "registration authorities". This is so since a party only needs to put complete trust in the registration authority where it registers its own public key. The trust in authorities used by other parties is much lower: It is only guaranteed that the public key is "well formed", in the sense that it was generated by applying f to some value given to the functionality. There are no guarantees regarding the distribution or secrecy of keys registered by other parties.

The key set-up functionality, $\mathcal{F}_{\text{KS}}$, formulated in [BCNP04] and presented in Figure 18, is written in a way that captures the "least common denominator" of several different set-up assumptions, including the CRS assumption, the key registration with knowledge assumption, and several others. As in the case of $\mathcal{F}_{\text{KRK}}$, $\mathcal{F}_{\text{KS}}$ is parameterized by a (deterministic) function $f : \{0,1\}^* \to \{0,1\}^*$, that represents a method for computing a public key given a secret (and supposedly random) key. The functionality allows parties to register their identities together with an associated "public key". However, $\mathcal{F}_{\text{KS}}$ provides only relatively weak guarantees regarding this public key, giving the adversary considerable freedom in determining this key. (This freedom is what makes $\mathcal{F}_{\text{KS}}$ so relaxed.)

Specifically, the "public key" to be associated with a party upon registration is determined as follows. The functionality keeps a set R of "good public keys". Upon receiving a registration request from party P (that belongs to the "eligibility set" $\mathcal{P}$ encoded in the SID), the functionality first notifies the adversary that a request was made and gives the adversary the option to set the registered key to some key p that is already in R. If the adversary declines to set the registered key, then the functionality determines the key on its own, by choosing a random secret r from a given domain (say, $\{0,1\}^k$ for a security parameter k) and letting $p = f(r)$. Once the registered key p is chosen, the functionality records (P, p) and returns p to P and to the adversary. Finally, if p was chosen by the functionality itself then p is added to R. If the registering party is corrupted, then the adversary can also specify, if it chooses, an arbitrary "secret key" r. In this case, P is registered with the value $f(r)$ (but r is not added to R).

A retrieval request (made by any party in $\mathcal{P}$) for the public key of party P is answered with either an error message $\perp$ or one of the registered public keys of P, where the adversary chooses which registered public key, if any, is returned. (That is, the adversary can prevent a party from retrieving any of the registered keys of another party.)

Functionality $\mathcal{F}^f_{\text{KS}}$

$\mathcal{F}^f_{\text{KS}}$ proceeds as follows, given function f and security parameter k. At the first activation a set R of strings is initialized to be empty.

Registration: When receiving input $(\texttt{Register}, sid)$ from a party P, verify that $sid = (\mathcal{P}, sid')$ for some set $\mathcal{P}$ of identities and that $P \in \mathcal{P}$; else ignore the input. Next, send $(\texttt{Register}, sid, P)$ to the adversary, and receive a value p' from the adversary. Then, if $p' \in R$ then let $p \leftarrow p'$. Else, choose $r \overset{\text{R}}{\leftarrow} \{0,1\}^k$, let $p \leftarrow f(r)$, and add p to R. Finally, record (P, p) and return (sid, p) to P and to the adversary.

Registration by a corrupted party: When receiving input $(\texttt{Register}, sid, r)$ from a corrupted party $P \in \mathcal{P}$, record $(P, f(r))$. In this case, $f(r)$ is *not* added to R.

Retrieval: When receiving a message $(\texttt{Retrieve}, sid, P)$ from party $P' \in \mathcal{P}$, send $(\texttt{Retrieve}, sid, P, P')$ to the adversary and obtain a value p in return. If (P, p) is recorded then return (sid, P, p) to P'. Else, return $(sid, P, \perp)$ to P'.

Figure 18: The Key Registration functionality

Notice that the uncorrupted parties do not obtain any secret keys associated with their public keys, whereas the corrupted parties may know the secret keys of their public keys. Furthermore, $\mathcal{F}_{\text{KS}}$ gives the adversary a fair amount of freedom in choosing the registered keys. It can set the keys associated with corrupted parties to be any arbitrary value (as long as the functionality received the corresponding private key). The adversary can also cause the keys of both corrupted and uncorrupted parties to be identical to the keys of other (either corrupted or uncorrupted) parties. Still, $\mathcal{F}_{\text{KS}}$ guarantees two basic properties: **(a)** the public keys of good parties are "safe" (in the sense that their secret keys were chosen at random and kept secret from the adversary), and **(b)** the public keys of the corrupted parties are "well-formed", in the sense that the functionality received the corresponding private keys.

In [BCNP04] the general feasibility result of [CLOS02] for $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{AUTH}})$-hybrid protocols is reproduced with respect to $(\mathcal{F}_{\text{KS}}, \mathcal{F}_{\text{AUTH}})$-hybrid protocols, for the case of *non-adaptive* party corruptions. In [DPW05] the same results are reproduced with respect to a variant of $\mathcal{F}_{\text{KS}}$ that is not restricted by an eligibility set, and provides the public keys to any party in the network. The case of adaptive corruptions is also covered there.

Realizing $\mathcal{F}_{\text{KS}}$. $\mathcal{F}_{\text{KS}}$ can be realized given a number of set-up assumptions:

Realizing $\mathcal{F}_{\text{KS}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model. $\mathcal{F}_{\text{KS}}^f$ can be realized in the $\mathcal{F}_{\text{CRS}}^D$-hybrid model, where $D = D_k$ is the distribution of $f(r)$ for r that is uniform in $\{0,1\}^k$. The protocol is straightforward: On input either $(\texttt{Register}, sid)$ or $(\texttt{Retrieve}, sid, P)$, party P sends $(\texttt{CRS}, sid)$ to $\mathcal{F}_{\text{CRS}}$ and returns the obtained value. We have:

Claim 26 *The above $\mathcal{F}_{\text{CRS}}^D$-hybrid protocol UC-realizes $\mathcal{F}_{\text{KS}}^f$.*

Proof: Without loss of generality we assume that the environment $\mathcal{Z}$ expects to interact with the dummy adversary. Consider the following ideal-process adversary $\mathcal{S}$, that interacts in the ideal process for $\mathcal{F}_{\text{KS}}$. When notified by $\mathcal{F}_{\text{KS}}$ that the first party has asked to register, $\mathcal{S}$ declines to intervene, lets this party obtain the value p, and reports p to the environment $\mathcal{Z}$. From now on, whenever a party asks to register, $\mathcal{S}$ instructs $\mathcal{F}_{\text{KS}}$ to set the public key of that party to p. Clearly the view of any environment $\mathcal{Z}$ from the interaction with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{KS}}$ is identical to its view of an interaction with parties running the protocol in the $\mathcal{F}_{\text{CRS}}$-hybrid model, with any adversary. □

Realizing $\mathcal{F}_{\text{KS}}$ given a randomized registration service. Consider a setting where the parties have access to a registration service where parties can register and obtain public keys that were chosen at random according to a given distribution (i.e., the public key is $f(r)$ for an $r \overset{\text{R}}{\leftarrow} \{0,1\}^k$). This is similar to $\mathcal{F}_{\text{KRK}}^f$, except that here **(a)** the public keys are chosen by the authority, and **(b)** the registering party does not obtain the corresponding private keys. We let $\mathcal{F}_{\text{RKR}}$ (for random key registration) denote the ideal functionality that captures this registration service.

$\mathcal{F}_{\text{KS}}$ can be realized in this setting via a similar protocol as for realizing $\mathcal{F}_{\text{KS}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model: On input either $(\texttt{Register}, sid)$ or $(\texttt{Retrieve}, sid, P)$, party P requests a key from $\mathcal{F}_{\text{RKR}}^f$ an returns the obtained value. It can be readily seen that this protocol UC-realizes $\mathcal{F}_{\text{KS}}^f$. (The simulator is the same as in the proof of Claim 26, except that it always declines to determine the value of the public key.)

Realizing $\mathcal{F}_{\text{KS}}$ given a registration service with knowledge. Next, consider the above key registration with knowledge set-up, namely $\mathcal{F}_{\text{KRK}}$. This set-up assumption reduces the trust in the service: no longer is the registration entity trusted to make truly random choices. It is only trusted to verify "well-formedness" of the keys. In particular, the keys of corrupted parties are no longer guaranteed to be randomly chosen. They also don't need to be distinct. We show how to realize $\mathcal{F}_{\text{KS}}$ in this setting. On input $(\texttt{Register}, sid)$, party P chooses a random $r \overset{\text{R}}{\leftarrow} \{0,1\}^k$, provides r to the registration authority, computes $p = f(r)$, *erases* r, and returns p. To see that this protocol UC-realizes $\mathcal{F}_{\text{KS}}^f$, construct the following adversary $\mathcal{S}$ that interacts with $\mathcal{F}_{\text{KS}}^f$: $\mathcal{S}$ will be similar to the above one with respect to uncorrupted registering parties (i.e., $\mathcal{S}$ lets $\mathcal{F}_{\text{KS}}^f$ choose the corresponding keys). When the registering party is corrupted, $\mathcal{S}$ obtains the secret value r sent by the corrupted party to $\mathcal{F}_{\text{KRK}}$, and requests $\mathcal{F}_{\text{KS}}$ to register r for the corrupted party.

Realizing $\mathcal{F}_{\text{KS}}$ using traditional proofs of knowledge. Finally, we demonstrate that it is possible to realize $\mathcal{F}_{\text{KRK}}^f$ (and thus also $\mathcal{F}_{\text{KS}}^f$) via traditional (non-UC) proofs of knowledge of the private key, under the assumption that the proofs of knowledge occur when there is no related network activity. Formally, the setup assumptions are $\mathcal{F}_{\text{REG}}$ and $\mathcal{F}_{\text{NC}}$ (See Figure 16), where the use of $\mathcal{F}_{\text{NC}}$ is very limited. To realize $\mathcal{F}^{f}_{\text{KRK}}$, the registering party will prove knowledge of the private key to the registration functionality. The proof of knowledge will take place using $\mathcal{F}_{\text{NC}}$ in order to guarantee that all other protocol executions are suspended while a proof of knowledge takes place. (We remark that $\mathcal{F}_{\text{NC}}$ has a different flavor than the other set-up functionalities, in that it does not involve any protocol action. Rather, it represents an assumption on the behavior of the adversary.)

6.7 Various corruption models

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

The modeling of Byzantine corruption behavior was given in Section 4.1. This section describes how other standard corruption models are captured within the present framework. We address non-adaptive, passive (semi-honest), and proactive corruptions. While these corruption models 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 13 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 13 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 different reaction patterns of parties to corruption requests. Security against such attacks is then defined as usual (i.e., by realizing an ideal functionality), where the corruption model is chosen appropriately. For instance, the ability of the adversary to observe or modify certain memory locations, or to detect whenever a certain internal operation (such as modular multiplication) takes place, can be directly modeled by having the party send an appropriate message to the adversary.

7 UC formulations of some primitives

This section demonstrates how a number of commonplace cryptographic tasks can be captured within the present framework. We also discuss issues that are immediately relevant to the actual formalization. Some of the covered material appears in various other papers, albeit often in different form. Whenever relevant, we point to the appropriate sources.

In most cases, the ideal functionalities here are formulated somewhat differently than the ones in previous versions 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 tasks. It is stressed, though, that the formulations here are not "set in stone"; they can (and should) be modified to fit different settings.

As pointed out earlier, in the present framework both the notion of a"cryptographic primitive" and the notion of a "computational model" are captured by the same technical construct, namely an ideal functionality. This unified modeling makes the distinction between the material covered here and in Section 6 somewhat arbitrary. The present partitioning attempts to follow the standard thought.

We first address the tasks of guaranteeing authenticated and secure communication. The basic case of guaranteeing authenticity and secrecy on a message-by-message basis was covered in Sections 6.2 and 6.3, as part of the presentation of the respective computational models. Here we concentrate on the task of "secure communication sessions" and the related task of key-exchange. We then proceed to the basic tasks of public-key encryption and digital signatures. Next we address two-party primitives such as "coin-tossing" (i.e., generating a common random string), commitment, oblivious transfer, and zero-knowledge. Finally, we formulate ideal functionalities that capture the multi-party tasks of Verifiable Secret Sharing and Secure Function Evaluation.

Each one of the ideal functionalities presented below corresponds to a *single instance* of the primitive it represents. This results in functionalities that are relatively simple and easy to work with. The composition theorem guarantees that protocols that securely realize each such functionality remain secure even when many instances are running concurrently, by different subsets of the parties, and in an adversarially scheduled way.

7.1 Secure Communication Sessions

It is often natural to partition the communication between parties into *sessions*, where in each session two parties exchange a number of messages. For instance, a session may be the set of messages exchanged by an instance of some protocol between two parties. It thus makes sense to protect (i.e., provide authenticity and secrecy for) all the messages in a session using a single instance of some protocol. Indeed, protocols for realizing secure sessions are considerably more efficient than protocols that protect each message separately. Here we provide a specification of the security properties of such protocols.

Essentially, the interface of a secure communication session protocol should consist of a "session initiation" request, which is followed by requests to send messages in this session, plus receiving messages that arrive from the network.

The basic security requirements are the authenticity and secrecy of the messages sent in the session. However, providing rigorous specification for this task involves a number of choices. In fact, multiple variants of this task exist, where each variant best fits a different scenario. Let us mention some of these choices: One choice is whether to require that the full identity of the peer to the session be part of the session initiation request, or alternatively to let the identity of the peer be discovered during the session. A related choice is whether to guarantee some level of anonymity for the communicating peers, either from each other or from third parties. Another choice is whether to require that both parties request to establish the session with each other before letting any message be sent on the session. A related choice is whether to guarantee that the parties know that the session has been established. Yet another choice is whether to guarantee *forward secrecy,* namely that messages sent in a session remain secret even if one of the peers becomes corrupted later; a related choice is whether to allow the parties to explicitly *terminate* a session, with the implication that no further messages are to be sent or received on that session.

Several formulations of secure session protocols exist, with varying levels of detail and different properties, e.g. [sh99, CK01, CK02, NMO05]. Here we formulate a very basic variant, captured by the ideal secure communication session functionality, $\mathcal{F}_{\text{SCS}}$, presented in Figure 19. $\mathcal{F}_{\text{SCS}}$ is parameterized by a leakage function $l : \{0,1\}^* \to \{0,1\}^*$, that plays a similar role as in $\mathcal{F}_{\text{SMT}}$ (see Section 6.3). Upon receiving the first $(\texttt{Establish-Session}, sid)$ input from some party I (called the *initiator*), it verifies that I and an additional identity R are included in the SID of $\mathcal{F}_{\text{SCS}}$, otherwise this input is ignored. It then marks the initiator as *active*, marks R as the *responder,* and sends a public delayed output $(\texttt{Establish-Session}, sid)$ to R. Next, upon receiving the input

$(\texttt{Establish-Session}, sid)$ from R, $\mathcal{F}_{\text{SCS}}$ records R as active. Now, when receiving a request from a party $P \in \{I, R\}$ to send a message m, and if P is recorded as active, $\mathcal{F}_{\text{SCS}}$ lets $\mathcal{A}$ know that P asked to send a message, together with $l(m)$, the leakable information on m. Also, if P is active then $\mathcal{F}_{\text{SCS}}$ sends a *private delayed output* m to R, along with the identity of I. (Recall that in delayed outputs the adversary decides when the message is actually delivered to R; see Section 6.1.)

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

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

1. Upon receiving an input $(\texttt{Establish-Session}, sid)$ from party I, verify that $sid = (I, R, sid')$ for some R, record I as active, record R as the responder, and send a public delayed output $(\texttt{Establish-Session}, sid)$ to R.
2. Upon receiving $(\texttt{Establish-Session}, sid)$ from party R, verify that R is recorded as the responder, and record R as active.
3. Upon receiving a value $(\texttt{Send}, sid, m)$ from party $P \in \{I, R\}$, send $(\texttt{Sent}, sid, P, l(m))$ to the adversary. In addition, if P is active then send a private delayed output $(\texttt{Sent}, sid, P, m)$ to the other party in $\{I, R\}$.

Figure 19: The Secure Session functionality, $\mathcal{F}_{\text{SCS}}$

$\mathcal{F}_{\text{SCS}}$ does not notify a participant when its peer requests to establish the session. Furthermore, there is no explicit provision for session termination, nor is there any privacy guarantee. Also, there is no "mutual authentication" guarantee here, in the sense that the initiator can establish a session and start sending messages to the responder, without having any guarantee that the responder received these messages or is even aware of the existence of the initiator. Still. $\mathcal{F}_{\text{SCS}}$ guarantees I that if it requested to establish a session with R then each message that is received via this session t times was sent by R at least t times in this session, and furthermore any messages sent by I on this session are delivered only to R (and the adversary obtains only the allowed leakage on these messages). R gets the same guarantees.

Another characteristic of $\mathcal{F}_{\text{SCS}}$ is that its interface with the initiator is different than its interface with the responder, in that the responder gets an *output* from $\mathcal{F}_{\text{SCS}}$ before it is expected to provide input. This seems to best reflect the way communication sessions are initiated in reality. In particular, it is the initiator who determines the SID. The responder learns the SID when receiving the first output from $\mathcal{F}_{\text{SCS}}$; it can then decide whether to participate in the interaction with the proposed SID.[24]

$\mathcal{F}_{\text{SCS}}$ behaves like a "standard corruption"functionality (as defined in Section 6.1). In particular, if a party gets corrupted after it has sent a message and before the message is delivered, then the adversary can avoid delivering this message altogether, and have the corrupted party generate other messages instead.

A typical methodology for securely realizing $\mathcal{F}_{\text{SCS}}$ starts with running a Key-Exchange protocol, and then encrypts and authenticates each message via symmetric encryption and message authentication codes using the shared key; see some more details in Section 7.1.1. This methodology is

[24]In contrast, in prior formulations of $\mathcal{F}_{\text{SCS}}$ (and $\mathcal{F}_{\text{KE}}$, see below) both the initiator and the responder were expected to provide input to the functionality before receiving any output. This in effect required the calling protocol to make sure that the parties agree on the SID beforehand, either via a preliminary exchange of nonces or by prior convention. The current formulation is more flexible in that it does not require such prior agreement. We thank Ralf Küsters for suggesting the present formulation.

analyzed in [CK01, K01, CK02]. A somewhat different methodology for realizing $\mathcal{F}_{\text{SCS}}$ is analyzed in [NMO05].

7.1.1 Key Exchange

Key exchange is a task where two parties interact in order to generate a common random value (a key) that remains unknown to everyone except the two parties. The main use of this task is for providing a "session key" for symmetric encryption and authentication of messages in order to obtain secure communication sessions. Consequently, in typical use, multiple instances of a key exchange protocol may be running concurrently between multiple pairs of parties. There exists a large body of work on the security requirements from key exchange protocols. Here we mention only [BR93, BCK98, Sh99, CK01, CK02, HMS03], which are more closely related to the present formulation.

In the present framework, a secure key exchange protocol is taken to be a protocol that securely realizes an ideal key exchange functionality. Several formulations of the key exchange functionality in the UC framework, with somewhat different properties and stresses, appear in the literature, e.g. [C01, CK02, CK02a, HMS03, CH04]. In addition, a UC formulation of the related task of *password-based* key exchange appears in [CHKLM05].

The choices to be made when writing such an ideal functionality are similar to the ones made for the secure communication session functionality, $\mathcal{F}_{\text{SCS}}$. As there, we concentrate on a very basic variant that guarantees only a minimal set of properties. Functionality $\mathcal{F}_{\text{KE}}$, presented in Figure 20, proceeds as follows. Similarly to $\mathcal{F}_{\text{SCS}}$, upon receiving the first $(\texttt{Establish-Key}, sid, R)$ input from some party I (called the *initiator*), it verifies that I and an additional identity R are included in the SID, otherwise this input is ignored. It then marks the initiator as *active*, marks R as the *responder,* and sends a public delayed output $(\texttt{Establish-Key}, sid)$ to R. Next, upon receiving the input $(\texttt{Establish-Session}, sid)$ from R, $\mathcal{F}_{\text{KE}}$ records R as active. Now, when receiving a value $(\texttt{Key}, sid, P, \tilde{k})$ from the adversary, $\mathcal{F}_{\text{KE}}$ first checks that $P \in \{I, R\}$ and that P is active. If not, then P gets no output. If P is active and the two peers are currently uncorrupted, then P obtains a truly random and secret key κ for that session. If any of the peers is corrupted then P receives the key $\tilde{k}$ determined by the adversary.

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

1. Upon receiving an input $(\texttt{Establish-Key}, sid)$ from party I, verify that $sid = (I, R, sid')$ for some identity R, record I as active, record R as the responder, and send a public delayed output $(\texttt{Establish-Key}, sid)$ to R.
2. Upon receiving $(\texttt{Establish-Key}, sid)$ from party R, verify that R is recorded as the responder, and record R as active.
3. Upon receiving a message $(\texttt{Key}, sid, P, \tilde{k})$ from the adversary, for $P \in \{I, R\}$ do:
   (a) If P is active and neither I, R are corrupted then do: If there is no recorded key κ then choose $\kappa \overset{\text{R}}{\leftarrow} \{0,1\}^k$ and record κ. Next, output $(\texttt{Key}, sid, \kappa)$ to P.
   (b) Else, if P is active and either of I, R is corrupted then output $(\texttt{Key}, sid, \tilde{k})$ to P.
   (c) Else (P is not active), do nothing.

Figure 20: The Key Exchange functionality, $\mathcal{F}_{\text{KE}}$

Let us briefly note some properties of $\mathcal{F}_{\text{KE}}$. first, an instance of $\mathcal{F}_{\text{KE}}$ handles the generation of a single session key. Security for the multi-session case is guaranteed via the UC and JUC theorems. Also, if any of the participants is corrupted at the time that its peer obtains the key, then there is no guarantee on the distribution or secrecy of this key. Also, similarly to $\mathcal{F}_{\text{SCS}}$, $\mathcal{F}_{\text{KE}}$ does not guarantee "mutual authentication", in the sense that a party I can obtain a session key with R even if R did not request to exchange a key with I. Also, $\mathcal{F}_{\text{KE}}$ requires both parties to know the full identities of their peers in advance, and does not provide any anonymity for their identities, neither from each other nor from third parties. Furthermore, since we take it to be a "standard corruption" functionality (see Section 6.1), it does not provide "forward secrecy" for the exchanged keys. Still, $\mathcal{F}_{\text{KE}}$ does provide the basic guarantees of key exchange: If party I obtained a key k from a session with requested peer R, and both I and R are uncorrupted, then k is ideally random and unknown to the adversary. Furthermore, if R obtains a key from this session then R has I as its peer and the key is k. As in the case of $\mathcal{F}_{\text{SCS}}$, $\mathcal{F}_{\text{KE}}$ does not require prior agreement on the SID; instead, the initiator determines the SID, and the responder learns the SID from interacting with the functionality.

It is often convenient to augment the definition of $\mathcal{F}_{\text{KE}}$ by letting the parties obtain other outputs from the exchange, in addition to the session key. Such outputs may include additional information regarding the identities of the peers (e.g., additional certificates or aliases). We leave such extensions out of scope.

On realizing $\mathcal{F}_{\text{KE}}$. The requirement that a protocol realizes $\mathcal{F}_{\text{KE}}$ is strictly stronger than other known definitions, such as those of [BR93, CK01]. Nonetheless, several natural and widely used protocols securely realize $\mathcal{F}_{\text{KE}}$. Furthermore, functionality $\mathcal{F}_{\text{KE}}$ can be relaxed so that securely realizing it becomes *equivalent* to the definition of [CK01]. See more details in [CK02].

Note that $\mathcal{F}_{\text{KE}}$ cannot be realized in the bare, unauthenticated model without the aid of some basic "authenticated set-up" mechanism. Such mechanisms can be represented via some other ideal functionalities. Examples include pre-shared keys among parties and public-key infrastructure such as certified public keys for public-key encryption or signatures. Here a single instance of the underlying authentication mechanism (i.e., a single pre-shared key, or a single verification or encryption key) is typically used for generating multiple session keys, namely for multiple instances of $\mathcal{F}_{\text{KE}}$. At first glance, this seems to preclude viable single-session analysis; Still, using universal composition with joint state (see Section 5.3.2), it is possible to analyze each instance of a protocol π for realizing $\mathcal{F}_{\text{KE}}$ as stand alone, even when multiple instances of π use the same instance of the underlying authentication mechanism. See more details in [CR03, C04].

From key exchange to secure sessions. A formalization of secure channels and a proof that standard secure session protocols using key-exchange are secure is given in [CK01, CK02]. We restate from [CK02]:

Let MAC be a secure Message Authentication Code function, and let ENC be a symmetric encryption scheme that is semantically secure against chosen plaintext attacks. Let $\text{SCS}_{\text{MAC,ENC}}$ be the following $\mathcal{F}_{\text{KE}}$-hybrid secure session protocol. When activated with input $(\texttt{Establish-Session}, sid, R)$, the initiator I sends an $(\texttt{Establish-Key}, sid_{\text{KE}}, R, aux)$ input to $\mathcal{F}_{\text{KE}}$, where sid_{KE} is derived from sid in some generic way to distinguish it from sid (say, by adding the string "KE"). Similarly, When the responder R receives an output $(\texttt{Establish-Key}, sid_{\text{KE}}, R)$ from $\mathcal{F}_{\text{KE}}$, it generates output $(\texttt{Establish-Session}, sid, R)$, and when receiving in response an input $(\texttt{Establish-Session}, sid)$ it inputs $(\texttt{Establish-Key}, sid_{\text{KE}})$ to $\mathcal{F}_{\text{KE}}$. When party $P \in \{I, R\}$ gets the key κ from $\mathcal{F}_{\text{KE}}$, it partitions κ into four disjoint pieces, to obtain keys $\kappa_e, \kappa_a, \kappa'_e, \kappa'_a$. ($\kappa_e$ and κ_a will be used to encrypt and authenticate the session traffic from P and the other two keys will be used to protect the session traffic to P.) Next, upon receiving an input value $(\texttt{Send}, sid, m)$, P computes $c = \text{ENC}_{\kappa_e}(m)$, $a = \text{MAC}_{\kappa_a}(c, l)$, and sends (sid, c, l, a) to R. Here l is a counter that is incremented for each message sent. When receiving a message (sid, c, l, a), $\text{SCS}_{\text{MAC,ENC}}$ within P first verifies that $a = \text{MAC}_{\kappa'_a}(c, l)$ and that no message with counter value l was previously received in this session. If so, then it locally outputs $\text{DEC}_{\kappa'_e}(c)$. Otherwise, it outputs nothing (or an error message).

Claim 27 *Let* MAC *be a secure Message Authentication Code function, and let* ENC *be a symmetric encryption scheme that is semantically secure against chosen plaintext attacks. Then, protocol* $\text{SCS}_{\text{MAC,ENC}}$ *securely realizes* $\mathcal{F}_{\text{SCS}}$ *with respect to non-adaptive party corruptions.*

If ENC is of a special form (essentially as in [BH92]), or alternatively $\mathcal{F}_{\text{SCS}}$ is relaxed as described in [CK02], then Claim 27 holds also in the case of adaptive party corruptions.

7.2 Public-key encryption and signatures

This section presents ideal functionalities that are aimed at capturing the desired behavior of public-key encryption and signature schemes when used as *tools* for constructing cryptographic protocols.

Representing these primitives as ideal functionalities has a number of advantages. First, there is the usual advantage of separating the details of specific algorithms from the specification, and analyzing protocols that use signatures or public-key encryption separately from the analysis of the actual signature or encryption algorithm. This separation becomes more powerful in common settings where multiple protocol instances use the same instance of the underlying scheme (either signature or encryption). Such situations occur, for instance, when multiple instances of multiple different protocols use the same "public key infrastructure", namely the same instance of a signature or encryption scheme. Here the only way we have to analyze each protocol instance separately, rather than having to analyze the entire system as a whole is to use universal composition with joint state (JUC, see Section 5.3). To use JUC, we must have ideal functionalities that represent signature or encryption.

Another advantage of representing signature and encryption as ideal functionalities is that it provides a "cryptographically sound bridge" to formal and automated analysis of protocols that uses non-computational abstractions of signature and encryption. Indeed, these functionalities play a key part in formal and automated analysis that guarantees UC security of protocols. (See e.g. [CH04, P05].)

The formulations of the encryption and signature functionalities are in line with the tradition of representing encryption and signature schemes as a tuple of non-interactive algorithms rather than as an interactive protocol. Indeed, the protocols that realize these functionalities do not involve any communication among the parties. They consist of a set of non-interactive algorithms that respond to any input with locally computed output. The "interactive part", namely the use of encryption and signature schemes in interaction is left to the "high-level protocols" that make use of the schemes. This style of formalism facilitates comparing the power of the ideal functionalities to the power of known formulations of these primitives. Indeed, we show that, for signature schemes, realizing the signature functionality is essentially equivalent to existential unforgeability against chosen message attacks (as defined in [GMRi88]). For encryption schemes, realizing the public-key encryption functionality in the presence of non-adaptive adversaries is essentially equivalent to security against adaptive chosen ciphertext attacks (CCA) [DDN00, RS91, BDPR98].

7.2.1 Digital Signatures

Digital signature schemes allow a distinguished party (the signer) to attach tags (signatures) to documents, so that everyone can verify, by locally running some public algorithm, that the signature was generated by no one else but the signer. Digital signature schemes were first proposed in Diffie and Hellman [DH76]; their basic security requirements were formulated in Goldwasser, Micali and Rivest [GMRi88].

Here we treat digital signatures as a cryptographic task within a larger protocol environment, thus providing another view of the security of such schemes. We concentrate on the security properties of "bare" signature schemes, without any public-key infrastructure. That is, binding is provided only between a signature and a public key, rather than between a signature and the identity of the signer. This treatment essentially regards signature schemes as a "technical tool" within other protocols, rather than a complete service. (Additional discussion on binding signatures to identities and its applications appears at the end of this section.)

Functionality $\mathcal{F}_{\text{SIG}}$. The basic idea of $\mathcal{F}_{\text{SIG}}$ is to provide a "registry service" where the signer S can register (*message,signature*) pairs. Any party that provides the right verification key can check whether a given pair is registered. However, as much ac the basic idea is simple, formalizing it in a way that is not over-restrictive, and at the same time guarantees our intuitive notion of an unforgeable signature scheme, has proven to be tricky (see discussion in [C04]). The formulation here takes a somewhat different approach than that of previous formulations in the literature, including [C04]. This is done to simplify the exposition, at the price of some loss in generality. (A more detailed comparison appears below.) We note that the [C04] treatment of $\mathcal{F}_{\text{CERT}}$ and its use for realizing $\mathcal{F}_{\text{AUTH}}$ can be adapted to the current formalization of $\mathcal{F}_{\text{SIG}}$ in a straightforward way. We leave the details to be completed by the interested reader.

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

Key Generation: Upon receiving a value $(\texttt{KeyGen}, sid)$ from some party S, verify that $sid = (S, sid')$ for some sid'. If not, then ignore the request. Else, hand $(\texttt{KeyGen}, sid)$ to the adversary. Upon receiving $(\texttt{Algorithms}, sid, s, v)$ from the adversary, where s is a description of a PPT ITM, and v is a description of a *deterministic* polytime ITM, output $(\texttt{Verification Algorithm}, sid, v)$ to S.

Signature Generation: Upon receiving a value $(\texttt{Sign}, sid, m)$ from S, let $\sigma = s(m)$, and verify that $v(m, s) = 1$. If so, then output $(\texttt{Signature}, sid, m, \sigma)$ to S and record the entry (m, σ). Else, output an error message to S and halt.

Signature Verification: Upon receiving a value $(\texttt{Verify}, sid, m, \sigma, v')$ from some party V, do: If $v' = v$, the signer is not corrupted, $v(m, \sigma) = 1$, and no entry (m, σ') for any σ' is recorded, then output an error message to S and halt. Else, output $(\texttt{Verified}, sid, m, v'(m, \sigma))$ to V.

Figure 21: The basic signature functionality, $\mathcal{F}_{\text{SIG}}$.

Functionality $\mathcal{F}_{\text{SIG}}$ is presented in Figure 21. It takes three types of inputs, which correspond to the three basic modules of a signature scheme: key generation, signature generation, and signature verification. Having received a key generation request from a party S, $\mathcal{F}_{\text{SIG}}$ first verifies that the identity S appears in the SID. (This convention essentially guarantees that each party can invoke $\mathcal{F}_{\text{SIG}}$ with an SID that no other party can use. An alternative convention is to incorporate the signer's identity as a parameter in the code of $\mathcal{F}_{\text{SIG}}$. In either case, *anonymity* for the signer can be guaranteed by using a random alias rather than the name itself.) Next, $\mathcal{F}_{\text{SIG}}$ asks the adversary to provide two descriptions of algorithms: A polynomial-time probabilistic signing algorithm s, and a polynomial-time *deterministic* verification algorithm v. It then outputs to S the description of the signing algorithm s. Finally, it invokes algorithms s and v, and maintains a running instance of each. (Recall the convention from Section 6.1 regarding running arbitrary code.) It is stressed that, while the verification algorithm is public and given to the environment (via S), the signing algorithm does not appear in the interface between $\mathcal{F}_{\text{SIG}}$ and S. Rather, it is treated as an "implementation detail."[25]

Upon receiving a request from party S (and only party S) to sign a message m, $\mathcal{F}_{\text{SIG}}$ first obtains a "formal signature string" σ by running s on input m and fresh random input. It then verifies that $v(m,\sigma) = 1$. If so, it outputs the signature string σ to S and records the pair (m,σ). Else, $\mathcal{F}_{\text{SIG}}$ outputs an error message and halts.

Upon receiving a request from some arbitrary party V to verify a signature σ on message m with verification algorithm (or, key) v', $\mathcal{F}_{\text{SIG}}$ proceeds as follows. It first checks if the input consists of a forgery, namely if $v' = v$, $v(m,\sigma) = 1$, and S is uncorrupted and never signed m in the past (namely, the pair (m,σ') is not recorded, for any σ'). If so, $\mathcal{F}_{\text{SIG}}$ outputs an error message and halts. Else, it outputs $v'(m,\sigma)$.

$\mathcal{F}_{\text{SIG}}$ is a standard corruption functionality, with an additional stipulation. That is, when a party P is corrupted, $\mathcal{F}_{\text{SIG}}$ records this fact, and reports to the adversary all the signing and verification requests made by P. If P is the signer S then the adversary gets also the signing algorithm s together with its current state.

The fact that the verification algorithm v is deterministic guarantees *consistency,* namely that all verification requests for the same triple (m,σ,v') are answered in the same way, even if made by different parties at different times. Verifying that $v(m,\sigma) = 1$ at signature generation time guarantees *completeness*, namely that if a signature was generated "honestly" (i.e., via $\mathcal{F}_{\text{SIG}}$) then it will verify. *Unforgeability* is guaranteed by the check for forgery at verification time. Below we make some additional remarks on the formulation of $\mathcal{F}_{\text{SIG}}$.

A single instance formulation. Functionality $\mathcal{F}_{\text{SIG}}$ captures a "single instance" of a signature scheme, namely a single pair of signing and verification keys (algorithms). This formulation simplifies the specification and analysis, both of protocols that realize $\mathcal{F}_{\text{SIG}}$ and protocols that use $\mathcal{F}_{\text{SIG}}$. Security in a setting where multiple instances of signing and verification keys (namely, multiple instances of $\mathcal{F}_{\text{SIG}}$) are used simultaneously, potentially with different signing parties, is guaranteed via the UC theorem. Furthermore, since multiple instances of $\mathcal{F}_{\text{SIG}}$ which have the same signer can be realized given a single instance of $\mathcal{F}_{\text{SIG}}$ [CR03], the JUC theorem guarantees the security of protocols where multiple instances use the same signing and verification key (namely, the same instance of $\mathcal{F}_{\text{SIG}}$). This approach should be contrasted with the multi-instance formulation of [BPW03a], where all instances of a signature scheme in the system (and other primitives) are part of a single instance of an ideal functionality, and a single-instance treatment is not possible.

Determining the values of the verification key and the signatures. Functionality $\mathcal{F}_{\text{SIG}}$ lets the adversary determine the values of the verification key and the legitimate signatures, via the algorithms

[25]We often fail to make the distinction between a "verification (resp., signing) key" and a "verification (resp., signing) algorithm". Indeed, the difference between the two is only an issue of representation, and is immaterial to the formal treatment.

s and v. This reflects the fact that the intuitive notion of security of signature schemes does not make any requirements on these values. In particular, the signature values may depend on the identity of the signer, on the signed message, or even on all the messages signed so far. There is also no requirement that signatures on different messages be different from each other. (Indeed, observe that the security properties guaranteed by $\mathcal{F}_{\text{SIG}}$ hold even if all signature strings are identical. Said otherwise, making sure that strings are different from each other is only a technical tool that allows realizing the abstract requirement in an algorithmic way.)

An alternative (and more restrictive) formulation would require that signatures depend only on the currently signed message (say, by requiring that algorithm s does not maintain state between invocations). This requirement was proposed in [BPW03a]. In the present formulation, requiring that s is stateless also guarantees that the signature value is "well spread", in the sense that the probability of having two different messages with the same signature must be negligible. An even more restrictive requirement is that the signature value have some pre-determined distribution. This requirement is reminiscent of verifiable random functions [MRV99]; however, verifiable random functions only guarantee that the signature value "appears random" to parties *other than the signer.*

Note that, in principle, the verification algorithm (or, key) v that $\mathcal{F}_{\text{SIG}}$ outputs at key generation need not be the same algorithm as the algorithm v used in processing the signature generation and verification requests. We unify the two algorithms for simplicity, since no tangible generality seems to be lost here.

Verification with an incorrect verification algorithm (key). If the verification algorithm v' presented by the verifier is not the registered one (i.e., v) then $\mathcal{F}_{\text{SIG}}$ provides no guarantees regarding the result of the verification process. This captures the fact that the basic functionality of a signature scheme only binds messages and signatures to verification keys, rather than to other entities such as party identities. It is the responsibility of the protocol that uses $\mathcal{F}_{\text{SIG}}$ to make sure that the verifying party has the correct verification key.

Allowing multiple signatures for a message. When the signer activates $\mathcal{F}_{\text{SIG}}$ multiple times for signing the same message, $\mathcal{F}_{\text{SIG}}$ allows algorithm s to generate multiple different signatures for that message. This reflects the fact that the standard security requirement from signature schemes does not prohibit schemes that, in different activations, generate different signatures for the same message. A more restrictive variant would mandate that each message may have at most a single valid signature.

Allowing public modification of signatures. When presented with a verification request for (m, σ, v), where m was legitimately signed but with a different signature σ', $\mathcal{F}_{\text{SIG}}$ allows $v(m, \sigma)$ to be either 0 or 1. This reflects the fact that the basic notion of security of signature schemes makes no requirement on the verification procedure in such a case. In particular, it may be possible to publicly generate new valid signatures for a message from existing ones. An alternative (and more restrictive) formulation would force rejection whenever σ was not generated by the signing algorithm as a signature on m. (In our formalization this would mean that $\mathcal{F}_{\text{SIG}}$ outputs an error message if $v(m, \sigma) = 1$ whenever (m, σ) is not already recorded.) This stronger requirement is considered in e.g. [GO92].

Allowing Corrupted signers to claim signatures. If the signer is corrupted, and $\mathcal{F}_{\text{SIG}}$ is asked to verify (m, σ, v), then $\mathcal{F}_{\text{SIG}}$ allows $v(m, \sigma) = 1$, even if m was never before singed. This feature captures the fact that the basic security properties from signature schemes do not prevent a corrupted signer from causing the verification algorithm to accept any signature as valid for any message. Let us clarify this point via an example. Take any "secure" signature scheme s and modify it into a signature scheme s' that is identical to s except that a bit b is perpended to the verification key. If $b = 0$ then the verification procedure remains unchanged. But if $b = 1$ then the verification procedure always accepts its input signature as a valid signature for its input message. We claim that the scheme s' is still "secure". (In particular, if s is CMA-secure then so is s'.) Still, s' allows a a corrupted party P to register with a public key that begins with a "1". In this case, P can claim any signature for any message as "its own". (As artificial as the scheme s' may look, similar properties are shared by existing signature schemes. For instance when using any RSA-based signature scheme, a corrupted party can publish a verification key that specifies an RSA modulus $N = 1$. In this case, the verification algorithm would accept any message-signature pair.)

Immediate output generation. Note that in all its activations, $\mathcal{F}_{\text{SIG}}$ generates output to the calling party immediately, without allowing the adversary to delay the output. In fact, the adversary does not even learn that a signature request and a response occurred. This formulation reflects the security of schemes where key generation, signature generation, and verifications are all local operations that do not involve interaction. To allow also implementations where these operations are carried out by distributed protocols, the functionality should be relaxed appropriately.

On the differences from previous formulations. The main difference between the present formulation of $\mathcal{F}_{\text{SIG}}$ and the previous ones in the literature, including the one in [C04], is that there the adversary plays the role of algorithms s and v. That is, instead of locally running s or v, $\mathcal{F}_{\text{SIG}}$ asks the adversary for the corresponding output value. That formulation is arguably more "elegant," and also more general in that the signature value is allowed to depend on general events occurring in the system, rather than being generated by pre-specified algorithms based only on the signed message and some local randomness. In particular, that formulation is more suitable for capturing the security of protocols where the signature is generated in a distributed way rather than by a locally-run algorithm.

However, this more general formulation requires $\mathcal{F}_{\text{SIG}}$ to perform some more complicated accounting to guarantee the same security properties. For instance, consistency and completeness cannot be guaranteed by simply relying on the fact that v is deterministic. Another drawback of that formulation is that it lets the adversary know the value of each signed message and signature string generated by $\mathcal{F}_{\text{SIG}}$. This property becomes problematic in applications where signatures are to be encrypted and are expected to remain unknown to the adversary.

Finally, we note that "hybrid" formulations are also possible, where the algorithms s and v are allowed to receive "periodic state updates" from the adversary, or alternatively "leak information" to the adversary in a controlled way.

Equivalence with the [GMRi88] notion of security. We first briefly restate the [GMRi88] notion, while making explicit some requirements that remain implicit there. A signature scheme is a triple of PPT algorithms $\Sigma = (gen, sig, ver)$, where sig may maintain local state between activations.

Definition 28 *A signature scheme $\Sigma = (gen, sig, ver)$ is called* EU-CMA *if the following properties hold for any negligible function $\nu()$, and all large enough values of the security parameter k.*

Completeness: For any message m, $\text{Prob}[(\bar{s},\bar{v}) \leftarrow gen(1^k);\ \sigma \leftarrow sig(\bar{s},m);\ 0 \leftarrow ver(m,\sigma,\bar{v})] < \nu(k)$.

Consistency (Non-repudiation): For any m, the probability that $gen(1^k)$ generates $(\bar{s},\bar{v})$ and $ver(m,\sigma,\bar{v})$ generates two different outputs in two independent invocations is smaller than $\nu(k)$. (This requirement is implicit in [GMRi88]. *Indeed, it holds trivially in their case, where ver is a local and deterministic algorithm.)*

Unforgeability: For any PPT forger F, $\text{Prob}[(\bar{s},\bar{v}) \leftarrow gen(1^k);\ (m,\sigma) \leftarrow F^{sig(\bar{s},\cdot)}(\bar{v}); 1 \leftarrow ver(m,\sigma,\bar{v})$ *and F never asked sig to sign m*$] < \nu(k)$.

Next, we describe how to translate a signature scheme $\Sigma = (gen, sig, ver)$ into a protocol π_Σ in the present setting. This is done as follows: When party S, running π_Σ, receives an input $(\texttt{KeyGen}, sid)$, it verifies that $sid = (S, sid')$ for some sid'. If not, it ignores the input. It then runs algorithm gen, keeps the signing key $\bar{s}$ and outputs the verification algorithm $ver_{\bar{v}}$. When S receives an input $(\texttt{Sign}, sid, m)$ for sid of the form $sid = (S, sid')$, it sets $\sigma = sig(\bar{s}, m)$ and outputs $(\texttt{Signature}, sid, m, \sigma)$. When any party gets an input $(\texttt{Verify}, sid, m, \sigma, v')$, it outputs $(\texttt{Verified}, sid, m, v'(m,\sigma))$. When a party is corrupted, it reveals its internal state, which includes all past signing and verification requests and answers, and for S also the state of the signing algorithm (including the signing key and the randomness used to sign past messages). We show:

Claim 29 *Let $\Sigma = (gen, sig, ver)$ be a signature scheme. Then π_Σ securely realizes $\mathcal{F}_{\text{SIG}}$ if and only if Σ is* EU-CMA.

Proof: We adapt the proof from [C04] to the present formulation of $\mathcal{F}_{\text{SIG}}$. Assume that Σ violates Definition 28. We show that π_Σ does not securely realize F_{SIG}. This is done by constructing an environment Z and an adversary $\mathcal{A}$ such that for any ideal-process adversary $\mathcal{S}$, $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and π_Σ or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{SIG}}$. In all cases, $\mathcal{Z}$ corrupts no party and passes no inputs to $\mathcal{A}$. (Since protocol π_Σ never sends any messages, this means that $\mathcal{A}$ is never activated.) In addition:

(I) Assume Σ is not complete, i.e. there exists m such that $\text{Prob}[(\bar{s},\bar{v}) \leftarrow gen(1^k);\ \sigma \leftarrow sig(\bar{s},m);\ 0 \leftarrow ver(m,\sigma,\bar{v})] < 1 - \nu(k)$ for infinitely many k's. Then $\mathcal{Z}$ simply sets $sid = (S, 0)$ and activates some party S with input $(\texttt{KeyGen}, sid)$, followed by $(\texttt{Sign}, sid, m)$, obtains v and σ, and then activates some party V with $(\texttt{Verify}, sid, m, \sigma, v)$ and outputs the returned verification value. Clearly $\mathcal{Z}$ outputs 1 whenever it interacts with $\mathcal{F}_{\text{SIG}}$, whereas in the interaction with π_Σ $\mathcal{Z}$ outputs 0 with non-negligible probability.

(II) Assume Σ is not consistent. Then $\mathcal{Z}$ operates similarly except that it activates V twice with $(\texttt{Verify}, sid, m, \sigma, v)$, and outputs 1 iff the two answers are the same. Again, when interacting with $\mathcal{F}_{\text{SIG}}$, $\mathcal{Z}$ outputs 1 always, whereas in the interaction with π_Σ $\mathcal{Z}$ outputs 0 with non-negligible probability.

(III) Assume Σ is not unforgeable, i.e. there exists a successful forger G for Σ. Then $\mathcal{Z}$ proceeds as above, except that it internally runs an instance of G and hands it the verification algorithm v obtained from S. From now on, whenever G asks its oracle to sign a message m, $\mathcal{Z}$ activates S with input $(\texttt{Sign}, sid = (S,0), m)$, and reports the output to G. When G generates a pair (m,σ), $\mathcal{Z}$ Proceeds as follows. If m was signed before then $\mathcal{Z}$ outputs 0 and halts. Else, $\mathcal{Z}$ activates some party with input $(\texttt{Verify}, sid, m, \sigma, v)$ and outputs the verification result. It can be readily seen that, when $\mathcal{Z}$ interacts with π_Σ, G sees exactly a standard chosen message attack on Σ, thus $\mathcal{Z}$ outputs 1 with non-negligible probability. However, if $\mathcal{A}$ is interacting with the ideal process for $\mathcal{F}_{\text{SIG}}$, then $\mathcal{Z}$ never outputs 1.

Now, assume that π_Σ does not realize $\mathcal{F}_{\text{SIG}}$. Using the equivalent notion of security with respect to the dummy adversary, we have that for any ideal-process adversary $\mathcal{S}$ there exists an environment $\mathcal{Z}$ that can tell whether it is interacting with $\mathcal{F}_{\text{SIG}}$ and $\mathcal{S}$, or with π_Σ and the dummy adversary $\mathcal{D}$. We show that Σ violates Definition 28. Since $\mathcal{Z}$ succeeds for any $\mathcal{S}$, it also succeeds for the following "generic" $\mathcal{S}$. Recall that there are no messages sent in the protocol, thus $\mathcal{S}$ does not interact with $\mathcal{Z}$ at all, except to corrupt parties. In fact, the only two activities of $\mathcal{S}$ are (a) to provide $\mathcal{F}_{\text{SIG}}$ with algorithms s and v at key generation, and (b), to react to corruption requests made by $\mathcal{Z}$. To generate s and v, $\mathcal{S}$ runs $(\bar{s}, \bar{v}) \leftarrow gen(1^k)$, and lets $s(\cdot) = sig_{\bar{s}}(\cdot)$, and $v(\cdot) = ver_{\bar{v}}(\cdot)$. When $\mathcal{Z}$ instructs to corrupt party P, $\mathcal{S}$ sends to $\mathcal{F}_{\text{SIG}}$ a corruption message for P, and forwards to $\mathcal{Z}$ the information provided by $\mathcal{F}_{\text{SIG}}$. (This information consists of the signing and verification requests made by P so far, and if P is the signer then also the current state of algorithm s.)

Now, assume that scheme Σ is both complete and consistent (otherwise the theorem is proven). We demonstrate that it is not unforgeable, by constructing a forger G. This is done as follows. G runs a simulated instance of $\mathcal{Z}$, and simulates for $\mathcal{Z}$ an interaction with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{SIG}}$ (where G plays the role of both $\mathcal{S}$ and $\mathcal{F}_{\text{SIG}}$ for $\mathcal{Z}$).

Like $\mathcal{S}$, G runs a simulated instance of $\mathcal{A}$. However, when $\mathcal{Z}$ activates some party P with input $(\texttt{KeyGen}, sid)$ with $sid = (P, sid')$ for some sid', G returns $v = ver_{\bar{v}}$ to $\mathcal{Z}$, where $\bar{v}$ is the public verification key v in G's input. Also, when $\mathcal{Z}$ activates P with input $(\texttt{Sign}, sid, m)$, G asks its signing algorithm for a signature σ on m, and returns σ to $\mathcal{Z}$. Whenever the simulated $\mathcal{Z}$ activates some party with input $(\texttt{Verify}, sid, m, \sigma, v)$, G checks whether (m, σ) constitute a forgery (i.e., whether m was never signed before and $ver(\bar{v}, m, \sigma) = 1$). If (m, σ) is a forgery, then G outputs that pair and halts. Else it continues the simulation. If $\mathcal{A}$ asks to corrupt the signer then G halts with a failure output.

We analyze the success probability of G. Let B denote the event that, in a run of π_Σ with $\mathcal{Z}$ and $sid = (S, sid')$, the signer S generates a public key v, and some party is activated with a verification request $(\texttt{Verify}, sid, m, \sigma, v)$, where $ver(m, \sigma, v) = 1$, and S is uncorrupted and never signed m. Since Σ is complete and consistent, we have that as long as event B does not occur, $\mathcal{Z}$'s view of an interaction with $\mathcal{A}$ and π_Σ is statistically close to its view of an interaction with $\mathcal{S}$ and $\mathcal{F}_{\text{SIG}}$ in the ideal process. (The views may differ in case of a completeness or consistency error. But these happen only with negligible probability.) However, we are guaranteed that $\mathcal{Z}$ distinguishes with non-negligible probability between the interaction with π_Σ and the interaction with $\mathcal{S}$ and $\mathcal{F}_{\text{SIG}}$ Thus we are guaranteed that, when $\mathcal{Z}$ interacts with $\mathcal{A}$ and π_Σ, event B occurs with non-negligible probability.

It remains to observe that, from the point of view of $\mathcal{Z}$, the interaction with the forger G looks the same as an interaction with π_Σ. Thus, we are guaranteed that event B will occur with non-negligible probability. Notice that event B can occur only *before* the signer S is corrupted. This means that whenever event B occurs, G outputs a successful forgery. $\square$

We remark that the adversary $\mathcal{A}$ constructed in the proof of the "only if" direction is non-adaptive, whereas the environment $\mathcal{Z}$ considered in the proof of the "if" direction can be adaptive. This means that for any signature scheme Σ, protocol π_Σ securely realizes $\mathcal{F}_{\text{SIG}}$ against adaptive adversaries if and only if it securely realizes $\mathcal{F}_{\text{SIG}}$ against non-adaptive adversaries. Also, it is interesting to note that the proof of Claim 29 does not involve any data erasures.

Modeling certification. An ideal functionality that provides ideal binding between a signature on a message and the identity of the signer is formalized in [C04]. Using common terminology, this functionality, called $\mathcal{F}_{\text{CERT}}$, corresponds to providing signatures accompanied by "certificates" that bind the signatures directly to the signer's identity (rather than to a public key). Indeed, in [C04] it is shown how $\mathcal{F}_{\text{CERT}}$ can be realized given access to $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{REG}}$ (see Figure 13), where the latter functionality represents ideally authenticated communication with a trusted "certification authority".

$\mathcal{F}_{\text{CERT}}$ can in turn be used as a basic building block for guaranteeing authenticated communication. That is, it is shown there how to realize $\mathcal{F}_{\text{AUTH}}$ by an $\mathcal{F}_{\text{CERT}}$-hybrid protocol. Similarly, $\mathcal{F}_{\text{CERT}}$ can be used to realize $\mathcal{F}_{\text{KE}}$ via standard signature-based protocols (such as the ISO 9798-3 standard, see [CK02]). This allows separating the details of the signature scheme and the certification authority protocol from the analysis of the key exchange protocol itself. In addition, in order to allow analyzing each instance of the protocols for $\mathcal{F}_{\text{AUTH}}$ and $\mathcal{F}_{\text{KE}}$ separately from all other instances, it is possible to let each protocol instance use its own dedicated instance of $\mathcal{F}_{\text{CERT}}$. In [CR03] it is then shown how to apply this analysis to the setting where all protocol instances use the same instance of $\mathcal{F}_{\text{CERT}}$; this is done using the universal composition with joint state (JUC) theorem. In particular, [CR03] construct a simple protocol that UC-realizes multiple instances of $\mathcal{F}_{\text{CERT}}$ with the same signer given a single instance of $\mathcal{F}_{\text{CERT}}$. The crux of that protocol is to include, when signing some text for some instance of $\mathcal{F}_{\text{CERT}}$, the SID of that instance of $\mathcal{F}_{\text{CERT}}$ in the signed text. (In fact, [CR03] address $\mathcal{F}_{\text{SIG}}$ rather than $\mathcal{F}_{\text{CERT}}$, but the treatment it the same.)

7.2.2 Public-Key Encryption

One use of public-key encryption is for obtaining protocols for secure message transmission (i.e., protocols for realizing $\mathcal{F}_{\text{SMT}}$) as described in Section 6.3. However, that application uses each public key for encrypting only a single message, thus it does not capture in full the functionality provided by public-key encryption. This section describes an ideal functionality, $\mathcal{F}_{\text{PKE}}$, that is aimed at capturing the general functionality provided by public-key encryption schemes when used as tools within cryptographic protocols. In particular, $\mathcal{F}_{\text{PKE}}$ guarantees the secrecy of encrypted messages in a setting where the same public key is used to encrypt multiple messages by multiple parties to a single recipient, and where the adversary may obtain the output of the decryption algorithm on inputs of its choice. Indeed, realizing $\mathcal{F}_{\text{PKE}}$ (against non-adaptive adversaries) turns out to be essentially equivalent to security against chosen ciphertext attacks (CCA) [DDN00, RS91, BDPR98].

This section is organized as follows. We first present a basic formulation of $\mathcal{F}_{\text{PKE}}$ and motivate some of the definitional choices involved. This is followed by demonstrating the equivalence with CCA security. Next we briefly review related issues and variants, including using $\mathcal{F}_{\text{PKE}}$ to realize many-to-one secret communication, binding the decryption key to the decryptor's identity, realizing $\mathcal{F}_{\text{PKE}}$ in the presence of adaptive adversaries, relaxing $\mathcal{F}_{\text{PKE}}$ to allow "replayability" of encryptions, and using $\mathcal{F}_{\text{PKE}}$ to enable symbolic analysis of protocols.

Functionality $\mathcal{F}_{\text{PKE}}$. The idea of $\mathcal{F}_{\text{PKE}}$ is to allow parties to obtain "idealized ciphertexts" for messages, such that the ciphertexts bear no computational relation to the messages, but at the same time the designated decryptor can present these ciphertexts and retrieve the original messages. This will be implemented by having $\mathcal{F}_{\text{PKE}}$ maintain a "centralized database" of encrypted messages and the corresponding ciphertexts.

As in the case of $\mathcal{F}_{\text{SIG}}$, implementing this idea in a way that is not too restrictive and still maintains the intuitive notion of security expected from public-key encryption requires some care.

Also, as in the case of $\mathcal{F}_{\text{SIG}}$, the formulation here is somewhat different than previous formulations in the literature, including [CKN03] and prior versions of this work. This is done mainly to simplify the exposition, at the price of some generality.

Functionality $\mathcal{F}_{\text{PKE}}$ is presented in Figure 22. It is written in a way that can be realized by protocols that involve only local operations (key generation, encryption, decryption.) All the communication is left to the protocols that call $\mathcal{F}_{\text{PKE}}$. The functionality is parameterized by a domain M of messages to be encrypted. The domain may be small (such as, say, $\{0,1\}$) or large (such as, say, $\{0,1\}^k$, or Z_p for a k-bit prime). For simplicity of exposition, we concentrate on the case where it is required that the encryption leaks no information on the encrypted message, other than the fact that it is in M. The more general case, where the encryption may leak some additional information on the plaintext, can be captured by parameterizing $\mathcal{F}_{\text{PKE}}$ with an appropriate leakage function.

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

$\mathcal{F}_{\text{PKE}}$ proceeds as follows, given domain M of plaintexts. Let $\mu \in M$ be a fixed message.

Key Generation: Upon receiving a value $(\texttt{KeyGen}, sid)$ from some party D, verify that $sid = (D, sid')$ for some sid'. If not, then ignore the request. Else, hand $(\texttt{KeyGen}, sid)$ to the adversary. Upon receiving $(\texttt{Algorithms}, sid, e, d)$ from the adversary, where e, d are descriptions of PPT ITMs, output $(\texttt{Encryption Algorithm}, sid, e)$ to D.

Encryption: Upon receiving a value $(\texttt{Encrypt}, sid, m, e')$ from any party E, do: If $m \notin M$ then output an error message to E. Else, if $e' \neq e$, or the decryptor D is corrupted, then let $c = e'(m)$. Else, let $c = e'(\mu)$ and record (m, c). Output $(\texttt{Ciphertext}, sid, c)$ to E.

Decryption: Upon receiving a value $(\texttt{Decrypt}, sid, c)$ from D (and D only), do: If there is a recorded entry (m, c) for some m then return $(\texttt{Plaintext}, m)$ to D. Else, return $(\texttt{Plaintext}, d(c))$. (If there is more than one m recorded for c then output an error message.)

Figure 22: The public-key encryption functionality, $\mathcal{F}_{\text{PKE}}$.

$\mathcal{F}_{\text{PKE}}$ takes three types of input: key generation, encryption, and decryption. Having received a key generation request from a party D, $\mathcal{F}_{\text{PKE}}$ first verifies that the identity D appears in the SID. (As in the case of $\mathcal{F}_{\text{SIG}}$, this convention essentially guarantees that each party can invoke $\mathcal{F}_{\text{PKE}}$ with an SID that no other party can use. An alternative convention is to incorporate the decryptor's identity as a parameter in the code of $\mathcal{F}_{\text{PKE}}$. In either case, *anonymity* for the decryptor can be guaranteed by using a random alias rather than the name itself.) Next, $\mathcal{F}_{\text{PKE}}$ asks the adversary to provide two descriptions of PPT algorithms: An encryption algorithm e and a decryption algorithm d. (Note that both e and d can be probabilistic.) It then outputs to D the description of the encryption algorithm e. Note that, in contrast with the case of $\mathcal{F}_{\text{SIG}}$, here no running instances of e or d are kept. Rather, a new instance of e (resp., d) is used for each new encryption (resp., decryption) operation. In particular, there is no state kept between encryption and decryption operations. Also, while the encryption algorithm is public and given to the environment (via D), the decryption algorithm does not appear in the interface between $\mathcal{F}_{\text{PKE}}$ and D. Rather, it is treated as an "implementation detail."

Upon receiving a request from some arbitrary party E to encrypt a message m with encryption algorithm (key) e', $\mathcal{F}_{\text{SCOM}}$ proceeds as follows. If m is not in the domain M then $\mathcal{F}_{\text{SCOM}}$ outputs an error message to E. Else, $\mathcal{F}_{\text{PKE}}$ outputs a "formal ciphertext" c to E, where c is computed as follows. If $e' = e$ and the decryptor D is uncorrupted, then $c = e(\mu)$, where $\mu \in M$ is some *fixed* message (say, the lexicographically first message in M). In this case, the pair (m, c) is recorded for future decryption. (Choosing c independently of m guarantees ideal secrecy for m.) If either $e' \neq e$ or D is corrupted then $c = e'(m)$. In this case, no secrecy is guaranteed, since c may depend on m in arbitrary ways. Also, there is no need to record (m, c) since correct decryption is not guaranteed.

Upon receiving a request from party D (and only party D) to decrypt a message m, $\mathcal{F}_{\text{PKE}}$ first checks if there is a record (m, c) for some m. If so, then it returns m as the decrypted value. This guarantees perfectly correct decryption for messages that were encrypted via this instance of $\mathcal{F}_{\text{PKE}}$. (If there is more than a single message m recorded with this c then unique decryption is not possible. In this case $\mathcal{F}_{\text{PKE}}$ outputs an error message.) If no (m, c) record exists for any m, this means that c was not generated "legitimately", via this instance of $\mathcal{F}_{\text{PKE}}$. So, no correctness guarantees are provided, and $\mathcal{F}_{\text{PKE}}$ returns the value $d(c)$.

$\mathcal{F}_{\text{PKE}}$ is a standard corruption functionality, with some additional stipulations. That is, when a party P is corrupted, $\mathcal{F}_{\text{PKE}}$ records this fact and reports to the adversary all the encryption and decryption requests made by P. In addition, the adversary receives the random choices made by e when computing the ciphertexts obtained by P. If P is the decryptor D then the adversary gets also d together with its current state. (This means that $\mathcal{F}_{\text{PKE}}$ does not guarantee "forward secrecy" of encrypted messages.)

We discuss some points regarding the formulation of $\mathcal{F}_{\text{PKE}}$. Some of these points are similar to points raised with respect to $\mathcal{F}_{\text{SIG}}$ and will be mentioned only briefly. Other points are specific to $\mathcal{F}_{\text{PKE}}$.

A single instance formulation. As in the case of $\mathcal{F}_{\text{SIG}}$, functionality $\mathcal{F}_{\text{PKE}}$ captures a "single instance" of an encryption scheme, namely a single pair of encryption and decryption keys (algorithms), with similar simplifying effects as there. We note that, using the same technique as that of [CR03], multiple instances of $\mathcal{F}_{\text{PKE}}$ that have the same decryptor can be realized using a single instance of $\mathcal{F}_{\text{PKE}}$. Thus the JUC theorem can be applied to $\mathcal{F}_{\text{PKE}}$ in the same way as it is applied to $\mathcal{F}_{\text{SIG}}$. (Essentially, the idea is to include the SID of the calling instance of $\mathcal{F}_{\text{PKE}}$ in each encrypted message. See more details in [CR03, CH04].)

Determining the values of the encryption algorithm and the ciphertexts. Functionality $\mathcal{F}_{\text{PKE}}$ lets the adversary determine the values of the encryption key and the legitimate ciphertexts, via algorithm e. This reflects the fact that the intuitive notion of security of encryption schemes does not make any requirements on these values, as long as the ciphertexts do not depend on the legitimately encrypted messages (up to the allowed leakage).

As in the case of $\mathcal{F}_{\text{SIG}}$, the algorithm e given to D at key generation need not be the same as the algorithm e used in the processing of encryption requests. We unify the two algorithms here only for sake of simplicity. In fact, the proof of security of some encryption schemes (e.g., to obtain adaptively secure encryption, see below) does require the two algorithms to be different.

Immediate output generation. In all activations, $\mathcal{F}_{\text{PKE}}$ generates output to the calling party immediately, without allowing the adversary to delay the output. In fact, the adversary does not even learn that an encryption or decryption occurred, unless the active party is corrupted. This reflects the security of schemes where key generation, encryption and decryption are all local operations that do not involve interaction. To allow implementations where these operations are carried out by distributed protocols, the functionality should be relaxed appropriately.

On stateless decryption and providing "ideal non-malleability". Non-malleability of encryption schemes, as coined in [DDN00], is the property that prevents an adversary from being able to modify, given only the decryption algorithm, a ciphertext $c = e(m)$ into a different ciphertext c' such that c' decrypts to a value that is related to m. Functionality $\mathcal{F}_{\text{PKE}}$ guarantees non-malleability in an ideal way. Indeed, consider a ciphertext c that was generated by $\mathcal{F}_{\text{PKE}}$ in response to a request to encrypt a message m with the correct encryption algorithm e, and assume that m is chosen locally by some party and not disclosed. Then, any ciphertext c' that is generated by the adversary *before* c is decrypted does not depend on m in any way. Since the decryption function is stateless, if follows that the value $d(c')$ that c' decrypts to does not depend on m in any way.

On the differences from previous formulations. As in the case of $\mathcal{F}_{\text{SIG}}$, the main difference between the present formulation of $\mathcal{F}_{\text{PKE}}$ and previous ones in the literature is that there the adversary plays the role of algorithms e and d. That is, instead of locally running e or d, $\mathcal{F}_{\text{PKE}}$ asks the adversary for the corresponding output value. The tradeoff between simplicity and generality is similar to the one there, with the following exception. Letting the adversary determine the decryption values of illegitimate ciphertexts *at decryption time* allows the adversary to cause an illegal ciphertext c to decrypt to a value that depends on the values the legitimate ciphertexts decrypt to, even when these legitimate ciphertexts were generated and decrypted *after* c was determined. In contrast, the present formalization of decryption by a stateless decryption algorithm means that any ciphertext, even illegitimate one, decrypts to a value determined only by the ciphertext itself.

Still, the weaker guarantee of the prior versions is meaningful, essentially since the adversary has to explicitly provide the decryption value to $\mathcal{F}_{\text{PKE}}$ (which means that illegitimate ciphertexts only decrypt to values that are known ot the adversary in advance.) Indeed, as demonstrated in [CKN03], this more relaxed formulation suffices for many natural applications of public-key encryption. See more discussion following the proof of Claim 30 below.[26]

Equivalence with CCA security. Equivalence between realizing $\mathcal{F}_{\text{PKE}}$ and CCA-security for public-key encryption schemes is shown in [CKN03, HMS03a]. For self containment, we repeat the result here with respect to the present formulation of $\mathcal{F}_{\text{PKE}}$.

Let $\Sigma = (gen, enc, dec)$ be an encryption scheme with message domain M. (Recall that gen is the key generation algorithm, enc is the encryption algorithm and dec is the decryption algorithm.) A first requirement from Σ is correct decryption, namely that for $(\bar{e}, \bar{d}) \leftarrow gen(1^k)$ and all m, $dec(\bar{d}, enc(\bar{e}, m)) = m$ except of negligible probability. Very loosely, Σ is said to be secure against adaptive chosen ciphertext attacks (or, CCA-secure) if no attacker F can succeed in the following experiment. Algorithm gen is run to generate an encryption key $\bar{e}$ and a decryption key $\bar{d}$. F is given algorithm $enc(\bar{e}, \cdot)$ and access to a decryption oracle $dec(\bar{d}, \cdot)$. At some point F generates a pair of messages $m_0, m_1 \in M$. In response, a bit $b \overset{R}{\leftarrow} \{0,1\}$ is chosen and F is given $c = enc(\bar{e}, m_b)$. From this point on, F may continue querying its decryption oracle, under the condition that it does not ask for a decryption of c. F succeeds if it guesses b with probability that is non-negligibly more than one half. See more details in [DDN00, RS91, BDPR98].

Recall the following transformation from an encryption scheme $\Sigma = (gen, enc, dec)$ to a protocol π_Σ that is geared toward realizing $\mathcal{F}_{\text{PKE}}$. This is done as follows: When party D, running π_Σ, receives an input (`KeyGen`,sid), it verifies that $sid = (D, sid')$ for some sid'. If not, it ignores the input. It then runs algorithm gen, keeps the decryption key $\bar{d}$ and outputs the encryption algorithm $enc(\bar{e}, \cdot)$. When any party gets an input (`Encrypt`,sid, m, e'), it outputs (`Ciphertext`,$sid, enc(\bar{e}, m)$). (If

[26]We thank Jonathan Herzog for pointing out this difference between the two formulations of $\mathcal{F}_{\text{PKE}}$.

$m \notin M$ then it outputs an error massage.) When D receives an input (Decrypt, sid, c) for sid of the form $sid = (D, sid')$, it outputs (Plaintext, $sid, dec(\bar{d}, c)$). (At this point we do not specify the behavior of parties upon corruption, since we only consider this protocol for non-adaptive adversaries. See more discussion on the adaptive case below.) We show:

Claim 30 *Let $\Sigma = (gen, enc, dec)$ be an encryption scheme over domain M. Then Σ is CCA-secure if and only if π_Σ UC-realizes $\mathcal{F}_{\text{PKE}}$ with respect to domain M and non-adaptive adversaries.*

Proof: We first show that if π_Σ securely realizes $\mathcal{F}_{\text{PKE}}$ in the presence of non-adaptive adversaries then S is CCA-secure. Assume that there exists an adversary F that predicts the bit b correctly with probability $1/2 + \epsilon$, in a CCA interaction with scheme S. We construct an environment $\mathcal{Z}$ that distinguishes with probability ϵ between an interaction in the ideal process for $\mathcal{F}_{\text{PKE}}$ with any ideal-process adversary $\mathcal{S}$, and an interaction with the dummy adversary $\mathcal{D}$ and π_Σ. Environment $\mathcal{Z}$ invokes an *instance* of $\mathcal{F}$, and proceeds as follows, in a network of two uncorrupted parties E, D.

1. Initially, $\mathcal{Z}$ activates D with input (KeyGen, sid) for $sid = (D, 0)$, obtains the encryption algorithm e and hands e to F.
2. When F generates the two test plaintexts (m_0, m_1), $\mathcal{Z}$ chooses $b \overset{\text{R}}{\leftarrow} \{0, 1\}$, activates E with input (Encrypt, sid, e, m_b), obtains a ciphertext c^*, and hands c^* to F as the test ciphertext.
3. When F asks its decryption oracle to decrypt a ciphertext $c \neq c^*$, $\mathcal{Z}$ activates D with input (Decrypt, sid, c), obtains a plaintext m, and hands m to F.
4. When F outputs a bit b', $\mathcal{Z}$ outputs $b \oplus b'$ and halts.

Analyzing $\mathcal{Z}$, notice that if $\mathcal{Z}$ interacts with the dummy adversary $\mathcal{D}$ and parties running π_Σ, then the instance of F within $\mathcal{Z}$ sees in fact a CCA interaction with scheme Σ. Thus, in this case $b' = b$ with probability at least $1/2 + \epsilon$.

In contrast, when $\mathcal{Z}$ interacts with the ideal protocol for $\mathcal{F}_{\text{PKE}}$ and *any* adversary, the view of the instance of F within $\mathcal{Z}$ is statistically independent of b, thus in this case $b' = b$ with probability exactly one half. To see why F's view is independent of b recall that the view of $\mathcal{F}$ consists of the text ciphertext c^*, plus the decryptions of all the ciphertexts generated by F (except for the decryption of c^*). However, $c^* = e(\mu)$ for the fixed message μ is independent of b. Furthermore, All the ciphertexts c generated by F are independent of b, thus their decryption $d(c)$ are independent of b.

It remains to show that if S is CCA-secure then π_Σ securely realizes $\mathcal{F}_{\text{PKE}}$. Let $\Sigma = (gen, enc, dec)$ be a CCA-secure encryption scheme. We show that π_Σ securely realizes $\mathcal{F}_{\text{PKE}}$. Using the equivalent notion of security with respect to the dummy adversary, we construct an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it interacts with $\mathcal{F}_{\text{PKE}}$ and $\mathcal{S}$ or with parties running π_Σ and the dummy adversary $\mathcal{D}$.

Recall that $\mathcal{D}$ takes three types of messages from $\mathcal{Z}$: either to corrupt parties, or to report on messages sent in the protocol, or to deliver some message. However, since we are dealing with non-adaptive adversaries, there are no party corruption instructions. Furthermore, since protocol π_Σ involves no sending of messages, there are no requests to report on or deliver messages. In fact, there is no communication between $\mathcal{Z}$ and $\mathcal{D}$ at all. Thus, the only way activity of $\mathcal{S}$ is to provide the algorithms e and d to $\mathcal{F}_{\text{PKE}}$. For this purpose, $\mathcal{S}$ runs algorithm gen to obtain keys $\bar{e}, \bar{d}$, and sets $e = enc(\bar{e}, \cdot)$ and $d = dec(\bar{d}, \cdot)$.

Analyzing $\mathcal{S}$, first note that if D is corrupted then the views of $\mathcal{Z}$ of the two interactions are identical. Next, consider the case where D is uncorrupted, and assume for contradiction that there is an environment $\mathcal{Z}$ that distinguishes between the real and ideal interactions. We use $\mathcal{Z}$ to construct an adversary F that breaks the CCA security of the encryption scheme S. More precisely, assume that for some value of the security parameter k we have $\text{EXEC}_{\mathcal{F}_{\text{PKE}},\mathcal{S},\mathcal{Z}}(k) - \text{EXEC}_{\pi_\Sigma,\mathcal{D},\mathcal{Z}}(k) > \epsilon$. We show that F guesses the bit b correctly in the CCA game with probability $1/2 + e/2p$, where p is the total number of messages that were encrypted throughout the run of the system. (Without loss of generality, we assume that in each execution of the protocol $\mathcal{Z}$ asks to encrypt exactly p messages.)

Adversary F proceeds as follows, given a decryption algorithm e, and having access to a decryption oracle O F first randomly chooses a number $h \overset{\text{R}}{\leftarrow} \{1, ..., p\}$. Next, F runs $\mathcal{Z}$ on the following simulated interaction with a system running π_Σ (and the dummy adversary $\mathcal{D}$). Let m_i denote the ith message that $\mathcal{Z}$ asks to encrypt in an execution.[27]

1. When $\mathcal{Z}$ activates some party P_i with input $(\texttt{KeyGen}, id)$, F lets P_i output the value e from F's input.

2. For the first $h - 1$ times that $\mathcal{Z}$ asks to encrypt some message, m_i, F lets the encrypting party return $c_i = e(m_i)$.

3. At the hth time that $\mathcal{Z}$ asks to encrypt a message, m_h, F queries its encryption oracle with the pair of messages (m_h, μ), where $\mu \in M$ is the fixed message used above, and obtains the test ciphertext c_h. It then hands c_h to $\mathcal{Z}$ as the encryption of m_h.

4. For the remaining $p-h$ times that $\mathcal{Z}$ asks to encrypt some message, m_i, F lets the encrypting party return $c_i = e(\mu)$.

5. Whenever the decryptor P_i is activated with input $(\texttt{Decrypt}, id, c)$ where $c = c_i$ for some i, F lets P_i return the corresponding plaintext m_i. (This holds for the case $i = h$ as well as $i \neq h$.) If c is different from all the c_i's then F queries its decryption oracle on c, obtains a value v, and lets P_i return v to $\mathcal{Z}$.

6. When $\mathcal{Z}$ halts, F outputs whatever $\mathcal{Z}$ outputs and halts.

Analyzing the success probability of F is done via a standard hybrids argument. Let the random variable H_i denote the output of $\mathcal{Z}$ from an interaction that is identical to an interaction with $\mathcal{S}$ in the ideal process, with the exception that the first i ciphertexts are computed as an encryption of the real plaintexts, rather than encryptions of μ.

It is easy to see that H_0 is statistically close to the output of $\mathcal{Z}$ in the ideal process, and H_p is identical the output of $\mathcal{Z}$. (This follows from the fact that the scheme Σ guarantees that $d(e(m)) = m$ except with negligible probability.) Furthermore, in a run of F, if the value c_h that F obtains from its encryption oracle is an encryption of m_h then the output of the simulated $\mathcal{Z}$ has the distribution of H_{h-1}. If c_h is an encryption of μ then the output of the simulated $\mathcal{Z}$ has the distribution of H_h. The theorem follows. $\square$

Remark: The case of stateful decryption. Consider the natural generalizations of CCA security and $\mathcal{F}_{\text{PKE}}$ to the case where the decryption algorithm d may maintain state across decryption

[27]Without loss of generality we assume that $\mathcal{Z}$ only asks to encrypt messages with the public key e that was generated by the decrypting party. Indeed, when $\mathcal{Z}$ asks to encrypt a message m with a public key $e' \neq e$, it receives a value $c = enc_{e'}(m, r)$ that it can compute by itself.

operations. Note that Claim 30 continues to hold even in this case; in fact, the proof remains the same. However, in this case neither notion is adequate for guaranteeing non-malleability, or more generally security in a general protocol setting. Indeed, when d may depend on past decrypted values, it is easy to construct CCA-secure schemes where it is possible, given a ciphertext c, to generate a "bogus" ciphertext c', such that if c' is decrypted after c then c' and c decrypt to related values. (In particular, letting d to be stateful results in a strictly weaker security guarantee than that provided by $\mathcal{F}_{\text{PKE}}$, even in its relaxed version from [CKN03].)

On Replayable Encryption. The "non-malleability" guarantee provided by $\mathcal{F}_{\text{PKE}}$ is very strong: Given a ciphertext $c = enc(m)$, it is impossible to generate a ciphertext c' that decrypts to any value m' that is related to m (via some pre-defined relation), without knowing m' in advance. In many scenarios, however, it suffices to use a somewhat relaxed guarantee, that allows generating from c ciphertexts $c' \neq c$ that decrypt to *the same plaintext* m, even without knowing m. Other relations are prohibited. In fact, the ability to generate "replayed versions" of c without knowing m may in fact be useful in some applications. This relaxed version of $\mathcal{F}_{\text{PKE}}$ (and, correspondingly, of CCA security) is formulated in [CKN03] under the name Replayable CCA (RCCA) security, and is shown to suffice for a number of commonplace applications of $\mathcal{F}_{\text{PKE}}$ and CCA security. It is also shown that natural relaxations of existing CCA-secure encryption schemes result in RCCA-secure schemes. An RCCA-secure encryption scheme with the extra property that even the decryptor cannot distinguish the "replayed ciphertext" c' from an independently generated encryption of m is constructed in [G04].

In [CKN03] RCCA security is formulated via a relaxed version of $\mathcal{F}_{\text{PKE}}$, called $\mathcal{F}_{\text{RPKE}}$. (Relations with appropriate relaxations of standard CCA security are also shown.) In terms of the present formulation of $\mathcal{F}_{\text{PKE}}$, $\mathcal{F}_{\text{RPKE}}$ is obtained by adding the following instructions to $\mathcal{F}_{\text{PKE}}$. At any point during the execution, the adversary may provide a pair (c, c') of ciphertexts. Upon receiving this pair, and if there is an entry (m, c) in the current database of $(plaintext, ciphertext)$ pairs, then the entry (m, c') is added to the database. Else the input is ignored. Note that no honest party may be aware that (m, c') was added to the database.

Realizing many-to-one secure communication. A quintessential application of public-key encryption schemes is to provide a means for multiple parties to send messages privately to a predetermined recipient R. For this purpose, the recipient publicizes its public encryption key (presumably, in an authenticated way), and anyone that wants to send a message privately to R, encrypts the message with R's encryption key and sends the ciphertext to R (again, presumably over an authenticated channel).

Previous versions of this work, starting with [C01], provide the following formalization of this application. First, an ideal functionality that captures the task of many-to-one secure communication is formulated. This functionality, $\mathcal{F}_{\text{M-SMT}}$, is similar to $\mathcal{F}_{\text{SMT}}$, except that it delivers an unbounded number of messages, coming from any party, as long as the messages are directed at one distinguished receiver R. It is then demonstrated that the above simple protocol, when using $\mathcal{F}_{\text{PKE}}$ as the underlying encryption mechanism, UC-realizes $\mathcal{F}_{\text{M-SMT}}$. Furthermore, this holds unconditionally, and even against computationally unbounded environments, and even when *adaptive* party corruptions are allowed. In [CKN03] it is demonstrated that the same result holds even if $\mathcal{F}_{\text{PKE}}$ is replaced with its replayable counterpart, $\mathcal{F}_{\text{RPKE}}$. We leave full details out of scope of this work.

Security against adaptive adversaries. Recall that, in the case of non-adaptive party corruptions, any CCA-secure encryption scheme can be used in a straightforward way to realize $\mathcal{F}_{\text{PKE}}$. However, when the adversary is allowed to corrupt parties during the course of the computation, and obtain their internal state, realizing $\mathcal{F}_{\text{PKE}}$ is a much harder problem. In fact, a simple argument (from [N02]) shows that, in an adaptive corruptions setting, there exist no realizations of $\mathcal{F}_{\text{M-SMT}}$ that allow parties to send an unbounded number of messages to R without having R send messages back. This holds even in a model that allows parties to erase unnecessary local data. (Essentially, it is shown that in such a protocol the decryption key should be longer than the overall length of decrypted messages.)

Several approaches to circumvent this basic impossibility exist. Non-committing encryption [CFGN96, B97, DN00] can be used to UC-realize a variant of $\mathcal{F}_{\text{PKE}}$ where both encryption and decryption keys are longer than the security parameter times the overall number of bits to be encrypted. This approach is the only one that does no involve data erasures. A solution that requires R to interact with each potential sender is described in [BH92]. A solution that allows encrypting an unbounded number of messages non-interactively with a single short public key is described in [CHK05]. Here, however, the decryption key is updated periodically, and the total length of encrypted messages sent between any two key updates must remain bounded. Also, once the decryption key is updated, ciphertexts aimed at prior periods can no longer be decrypted, even by the legitimate decryptor. (The public key remains fixed throughout the computation.)

Binding the public key to the recipient identity. Analogously to $\mathcal{F}_{\text{SIG}}$, $\mathcal{F}_{\text{PKE}}$ does not provide binding between the public encryption key and the decryptor identity. That is, if some party uses some instance of $\mathcal{F}_{\text{PKE}}$ to encrypt a message with "the wrong encryption key" (i.e., an encryption key that is different than the key given by $\mathcal{F}_{\text{PKE}}$ to the decryptor), then no security is guaranteed for that plaintext. This reflects the fact that $\mathcal{F}_{\text{PKE}}$ does not provide means for the decryptor to distribute the public key to other parties.

A higher-level abstraction of public-key encryption, where the public encryption key is ideally bound to the decryptor identity, is described in [CH04]. This abstraction takes the form of the certified public key encryption functionality, $\mathcal{F}_{\text{CPKE}}$, which is analogous to the way $\mathcal{F}_{\text{CERT}}$ from [C04] abstracts $\mathcal{F}_{\text{SIG}}$. Essentially, in $\mathcal{F}_{\text{CPKE}}$ there is no key generation interface, and only the identity of the decryptor needs to be provided at encryption requests. The public encryption key is no longer part of the interface at all; rather, it is treated as an implementation detail. As in the case of $\mathcal{F}_{\text{CERT}}$, $\mathcal{F}_{\text{CPKE}}$ can be realized in a straightforward way given $\mathcal{F}_{\text{PKE}}$ and an "ideal key registration functionality", $\mathcal{F}_{\text{REG}}$. See [CH04] for more details.

Using $\mathcal{F}_{\text{PKE}}$ for symbolic security analysis of protocols. As sketched in the related work section of the introduction (Section 1.4.2), $\mathcal{F}_{\text{CPKE}}$ has been used as a basis for computationally sound symbolic analysis of protocols that use public-key encryption as their only cryptographic primitive [CH04].

We note that the key property of $\mathcal{F}_{\text{CPKE}}$ that enables this application is that it provides an *unconditionally secure* idealized encryption service to the protocols that use them. That is, secrecy, non-malleability, and correct decryption of encrypted messages are guaranteed even in the presence of computationally unbounded protocols and environments. Indeed, while symbolic analysis provides a relatively simple and automatable methodology for analyzing security of protocols, it cannot effectively handle computational issues and requires unconditional formulations of cryptographic primitives. Consequently, in this application, $\mathcal{F}_{\text{PKE}}$ and $\mathcal{F}_{\text{CPKE}}$ serve as a "bridge" between the unconditional, symbolic analysis of an abstract protocol and the computational realizations of that protocol.

7.3 Two-party tasks

We consider four commonplace two-party tasks: Commitment, Coin Tossing, Zero Knowledge, and Oblivious Transfer. These tasks are treated as tasks for two parties in a larger multiparty network. That is, security should be guaranteed even when many pairs of parties may be executing multiple instances of these protocols concurrently, without being aware of other potential protocols and executions. Indeed, this setting raises numerous security concerns, such as malleability and concurrency issues, that are not relevant when running two-party protocols in a stand-alone setting. In addition, in an adaptive corruption setting, the case where both communicating parties are eventually corrupted becomes relevant.

In the present framework, each of the four tasks is captured by an ideal functionality, respectively denoted $\mathcal{F}_{\text{COM}}$, $\mathcal{F}_{\text{CT}}$, $\mathcal{F}_{\text{ZK}}$ and $\mathcal{F}_{\text{OT}}$. We present these functionalities and briefly discuss the known facts regarding their realizability. Essentially, we have:

- All of these functionalities are UC-realizable by multi-party protocols with authenticated communication, as long as only a minority of the parties are corrupted.

- None of these functionalities is UC-realizable in the bare model by useful two-party protocols, even if authenticated communication is available. (Recall that a two-party protocol is a protocol where at most two parties send messages. A protocol is useful if there exist at least one party that generates output with some environment and adversary.)

- Any of these four functionalities suffices for UC-realizing any other functionality of the four, via a two-party protocol. That is, for any two functionalities there exists a protocol that realizes one functionality, using ideal calls to the other. (Some of these realization are statistical, namely work even for computationally unbounded environments. Others are computational. See details within.)

- Any of these four functionalities suffices for realizing any "well-formed" two-party functionality (i.e., any functionality that adheres to a number of syntactic rules, and where no more than two parties receive output) via a two-party protocol.

See more discussion and related work in Sections 1.4.3 and 1.4.5.

7.3.1 Commitment

Informally, commitment is a two-party protocol that has two phases: a commit phase, where the receiver of the commitment obtains some information which amounts to a "commitment" to an unknown value, and a reveal phase, where the receiver obtains an "opening" of the commitment to some value, and verifies whether the opening is valid. Roughly speaking, the security guarantee is that once the commit phase ends, there is only a single value that the receiver will consider as a valid opening (and, of course, that an honest committer can convince an honest receiver in the validity of an opening).

Figuratively, we envision that the committer provides the receiver with the digital equivalent of a "sealed envelope" containing a value x. From this point on, the committer cannot change the value inside the envelope, and, as long as the committer does not assist the receiver in opening the envelope, the receiver learns nothing about x. When both parties cooperate, the value x is retrieved in full.

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

1. Upon receiving an input $(\texttt{Commit}, sid, x)$ from C, verify that $sid = (C, R, sid')$ for some R, else ignore the input. Next, record x and generate a public delayed output $(\texttt{Receipt}, sid)$ to R. Once x is recorded, ignore any subsequent `Commit` inputs.
2. Upon receiving an input $(\texttt{Open}, sid)$ from C, proceed as follows: If there is a recorded value x then generate a public delayed output $(\texttt{Open}, sid, x)$ to R. Otherwise, do nothing.
3. Upon receiving a message $(\texttt{Corrupt-committer}, sid)$ from the adversary, send x to the adversary. Furthermore, if the adversary now provides a value x', and the `Receipt` output was not yet written on R's tape, then change the recorded value to x'.

Figure 23: The Ideal Commitment functionality, $\mathcal{F}_{\text{COM}}$

In the present formalization commitment protocols are protocols that securely realize the following "ideal commitment functionality," denoted $\mathcal{F}_{\text{COM}}$ and presented in Figure 23. ($\mathcal{F}_{\text{COM}}$ was first formalized in [CF01].) The commitment phase is modeled by having $\mathcal{F}_{\text{COM}}$ receive an input $(\texttt{Commit}, sid, x)$, from some party C (the committer), where x is the value committed to. As usual, the SID sid is expected to encode the identities of the committer C and receiver R. In response, $\mathcal{F}_{\text{COM}}$ records x and lets R *and the adversary* $\mathcal{S}$ know that C has committed to some value, and that this value is associated with SID sid. This is done by generating a *public delayed output* $(\texttt{Receipt}, sid)$ to R. (I.e., $\mathcal{F}_{\text{COM}}$ first sends this value to the adversary; when receiving confirmation from the adversary, $\mathcal{F}_{\text{COM}}$ outputs this value to R.) The opening phase is initiated by the committer sending a value $(\texttt{Open}, sid)$ to $\mathcal{F}_{\text{COM}}$. In response, $\mathcal{F}_{\text{COM}}$ generates a *public delayed output* $(\texttt{Open}, sid, x)$ to R.

When the adversary corrupts the committer, $\mathcal{F}_{\text{COM}}$ reveals the recorded value x to the adversary. In addition, if the `Receipt` value was not yet delivered R, then $\mathcal{F}_{\text{COM}}$ allows the adversary to modify the committed value. (This last stipulation captures the fact that the committed value is fixed only at the end of the commit phase, thus if the committer is corrupted during that phase then the adversary might still be able to modify the committed value.[28])

As usual functionality $\mathcal{F}_{\text{COM}}$ corresponds to only a single instance of a commitment protocol. The composition theorem guarantees that a protocol that securely realizes this functionality will be secure even in the multi-instance cases and in conjunction with any application. This in particular implies that known security requirements, such as non-malleability [DDN00] and security for selective decommitment [DNRS99], are met. For instance, committing to a string can be done by committing separately to each of the bits of the string, where all these commitments can be done in parallel.

Note that $\mathcal{F}_{\text{COM}}$ reveals the decommitted value to the adversary, thus capturing the standard assumption that the decommitted value is public. The concern that the opening of the commitment should be available only to the receiver can be captured by modifying $\mathcal{F}_{\text{COM}}$ so that the adversary does not receive the opened value x.

[28]The formulation of $\mathcal{F}_{\text{COM}}$ in [CF01] and in previous versions of this work do not include this instruction, and are thus essentially unrealizable in an adaptive corruption setting. Ivan Damgaard and Jesper Nielsen, Phil MacKenzie and Ke Yang, and Dennis Hofheinz have independently drawn our attention to this point. Thanks to all.

On the feasibility of realizing $\mathcal{F}_{\text{COM}}$. We summarize the relevant results. The first result is negative:

Claim 31 ([CF01]) *There do not exist useful two-party protocols that UC-realize $\mathcal{F}_{\text{COM}}$. This holds even assuming authenticated communication (i.e., for $\mathcal{F}_{\text{AUTH}}$-hybrid protocols).*

In contrast, [CF01] show an $(\mathcal{F}_{\text{AUTH}}, \mathcal{F}_{\text{CRS}})$-hybrid protocol that UC-realizes $\mathcal{F}_{\text{COM}}$ if trapdoor permutations exist. (In general, the distribution of the reference string depends on the trapdoor permutation in use. If dense cryptosystems exist then the distribution can be made uniform. The eligibility set consists of the two participating parties.) The protocol is non-interactive, in the sense that both the commitment and the opening phases consist of a single message sent from the committer to the receiver.

This result is then strengthened as follows. Recall that in the case of $\mathcal{F}_{\text{CRS}}$-hybrid protocols the universal composition operation means that each instance of the protocol uses a different instance of $\mathcal{F}_{\text{CRS}}$. This means that the overall number of reference strings must be at least linear in the number of applications of the commitment protocol. In order to guarantee universal composability while using only a single short reference string, a variant of $\mathcal{F}_{\text{COM}}$, called $\mathcal{F}_{\text{MCOM}}$, is defined, that handles multiple commitment and decommitment operations within a single instance. Then protocols are constructed, that UC-realize $\mathcal{F}_{\text{MCOM}}$ given a single instance of an $\mathcal{F}_{\text{CRS}}$. That is, these protocols can handle multiple commitment "sessions" using on a single, short common string. See Section 1.4.5 on page 15 for a brief survey of the known protocols for realizing $\mathcal{F}_{\text{MCOM}}$.

7.3.2 Coin tossing

The task of "tossing a common coin" is one of the first and more fundamental protocol problems considered in the literature (see, e.g., [B82]). In its simplest form, the task calls for two mutually distrustful parties to generate a common unbiased random bit. It can be naturally generalized to more than two parties and to having the parties generate a string drawn from an arbitrary (samplable) pre-determined distribution. It can also be somewhat relaxed by allowing the generated string to drawn from some distribution that is adversarially chosen from some family of distributions that satisfy a certain set of conditions.

We capture coin tossing via an ideal functionality, $\mathcal{F}_{\text{CT}}$, which is in fact the same as the common reference string functionality $\mathcal{F}_{\text{CRS}}^{D}$ (Figure 17 on page 78), for the case where D is the uniform distribution over $\{0, 1\}$, and the set eligibility set $\mathcal{P}$ consists of the two participants in the protocol. (Recall that $\mathcal{P}$ is determined by the party which invokes $\mathcal{F}_{\text{CT}}$, namely the initiator of the protocol.)

Generating a polynomially long common random string (for any arbitrary polynomial) can be done by running multiple instances of $\mathcal{F}_{\text{CT}}$ in parallel and concatenating the resulting bits. This in particular means that it is possible to realize $\mathcal{F}_{\text{COM}}$ and $\mathcal{F}_{\text{MCOM}}$ by $\mathcal{F}_{\text{CT}}$-hybrid protocols. In addition, as demonstrated in [CR03], it is possible to UC-realize $\mathcal{F}_{\text{CT}}$ by casting the Blum coin-tossing-by-phone protocol [B82] as an $\mathcal{F}_{\text{COM}}$-hybrid protocol. In fact, the resulting protocol UC-realizes $\mathcal{F}_{\text{CT}}$ in a perfect sense, namely even unbounded environments have exactly the same views when interacting with the protocol and when interaction with $\mathcal{F}_{\text{CT}}$. Furthermore, as shown there, the communication complexity of the Blum protocol is close to the best possible in the following sense: In any protocol that uses a common random string of length s to generate a common random string of length $s + t$, both parties must send at least t bits of communication.

7.3.3 Zero Knowledge

Zero-knowledge, as defined in [GMRa89], is a task for two parties, a prover and a verifier, that have a binary relation R (which is polynomial in the length of the first argument) and a common input x. The prover also receives an additional input, w. If $R(x,w)=1$ then the verifier should accept. If there exists no w such that $R(x,w)=1$ then the verifier should reject. There is no requirement on the output of the verifier in the case where there exists w s.t. $R(x,w)=1$ but the prover does not have such w as input. (In the setting of [GMRa89], where the prover is computationally unbounded, this is a moot point, since the prover can always find such w in case it exists. When the prover is PPT, this distinction becomes significant.) Furthermore, the verifier should "learn nothing" from the protocol except of whether there exists a w such that $R(x,w)=1$. Informally, a zero-Knowledge protocol is a proof of knowledge if in addition it is possible to "extract" w from any (possibly adversarial) prover that manages to make the verifier accept with non-negligible probability. See [G01] for many more details on and variants of zero-knowledge and proofs of knowledge.

We propose an ideal functionality, $\mathcal{F}_{\text{ZK}}$, that is aimed at capturing the security properties of this task, while guaranteeing universal composability. Functionality $\mathcal{F}_{\text{ZK}}$ is described in Figure 24. It is parameterized by a binary relation R, and expects a single input, $(\texttt{Prove}, sid, x, w)$ from a prover P, where the identities of P and the verifier V are encoded in sid. If $R(x,w)$ holds, then $\mathcal{F}_{\text{ZK}}$ generates a public delayed output $(\texttt{Verified}, sid, x)$ to V. If $R(x,w)$ does not hold then $\mathcal{F}_{\text{ZK}}$ generates no output. If the adversary instructs to corrupt P then it learns w. Furthermore, if no output was yet written to V then the adversary is allowed to change the values of x and w. (This provision reflects the fact that if the prover is corrupted before the verifier has generated output then it might still modify the outcome of the interaction.)

Functionality $\mathcal{F}^R_{\text{ZK}}$

$\mathcal{F}^R_{\text{ZK}}$ is parameterized by a binary relation R. It proceeds as follows.

1. Upon receiving an input $(\texttt{Prove}, sid, x, w)$ from some party P, verify that $sid = (P, V, sid')$ for some V, else ignore the input. Next, if $R(x,w)$ holds then generate a public delayed output $(\texttt{Verified}, sid, x)$ to V, else do noting. From now on, ignore future $(\texttt{Prove}\ldots)$ inputs.
2. Upon receiving a message $(\texttt{Corrupt-prover}, sid)$ from the adversary, send w to the adversary. Furthermore, if the adversary now provides a value (x', w') such that $R(x', w')$ holds, and no output was yet written to V, then output $(\texttt{Verified}, sid, x')$ to V.

Figure 24: The Zero-Knowledge functionality, $\mathcal{F}_{\text{ZK}}$

Two differences between $\mathcal{F}_{\text{ZK}}$ and the traditional formulation of zero-knowledge are, first, that there the input x is known a priori to both parties, whereas here V learns x from the protocol, and second that there V has both "accept" and "reject" outputs, whereas here only one output is possible, corresponding to "accept". The "reject" output is turned into outputting nothing at all. However, neither difference is substantial; indeed, it is easy to turn protocols for one formulation into protocols in the other formulation. The present formulation seems more convenient in a general protocol setting, as exemplified, e.g., in use of $\mathcal{F}_{\text{ZK}}$ in [CLOS02].[29]

$\mathcal{F}_{\text{ZK}}$ also has a flavor of a proof of knowledge, in that even adversarial provers must explicitly

[29]We thank Yehuda Lindell for proposing the present formulation of $\mathcal{F}_{\text{ZK}}$.

provide the "witness" w to $\mathcal{F}_{\text{ZK}}$ in order to make the verifier accept. This means that if the protocol instructs the to accept x, then there is an adversary that corrupts the prover and is able to explicitly give $\mathcal{F}_{\text{ZK}}$ a value w that stands in the relation with x.

As usual, $\mathcal{F}_{\text{ZK}}$ is defined for a single instance, namely for a single proof. The UC theorem guarantees that, if a protocol UC-realizes $\mathcal{F}_{\text{ZK}}$ for some relation R, then security of this protocol is guaranteed even when many instances of a ZK protocol are running concurrently, in an interleaved way and with related inputs. This in particular guarantees that protocols that UC-realize $\mathcal{F}_{\text{ZK}}$ are Concurrent Zero Knowledge protocols as in [F91, DNS98]. Moreover, they are strong proofs of knowledge as in [DDOPS01].

It may also be interesting to compare $\mathcal{F}_{\text{ZK}}$ to $\mathcal{F}_{\text{AUTH}}$ (Figure 12 on page 69). Indeed, both functionalities allow one party to send a public message to another party in an authenticated way. The difference is that $\mathcal{F}_{\text{ZK}}$ also guarantees to the recipient that the sender "knows" some secret value associated with the public message. As seen in Claim 32 below, this seemingly small difference is actually very significant.

Finally, we remark that realizing $\mathcal{F}_{\text{ZK}}$ by protocols that communicate via the non-concurrent execution functionality $\mathcal{F}_{\text{NC}}$ (Section 6.5) is essentially equivalent to the standard notion of computationally sound zero-knowledge proofs of knowledge (CSZKPoK), say in [G01]. That is, any CSZKPoK protocol, cast as an $\mathcal{F}_{\text{NC}}$-hybrid protocol, realizes $\mathcal{F}_{\text{ZK}}$, and vice versa. (One caveat here is that the proof of knowledge guaranteed by $\mathcal{F}_{\text{ZK}}$ is not necessarily black-box, as usually required for proofs of knowledge.)

Realizing $\mathcal{F}_{\text{ZK}}$. As demonstrated in [CKL03], $\mathcal{F}_{\text{ZK}}^R$ cannot be realized by useful two parties $\mathcal{F}_{\text{AUTH}}$-hybrid protocols, unless R is trivial in the sense that it is possible to determine in PPT, given x, whether there exists w such that $R(x, w)$ holds. On the positive side, an $\mathcal{F}_{\text{COM}}$-hybrid protocol that UC-realizes $\mathcal{F}_{\text{ZK}}^R$ for any NP relation R is given in [CF01]. The protocol is essentially the Blum three-message zero-knowledge protocol for Hamiltonicity, cast as an $\mathcal{F}_{\text{COM}}$-hybrid protocol (namely, the commitments are replaced by instances of $\mathcal{F}_{\text{COM}}$). Furthermore, the protocol UC-realizes $\mathcal{F}_{\text{ZK}}$ in a statistical way, namely even against unbounded environments, and even in the presence of adaptive party corruptions. A non-interactive (namely, one message), $\mathcal{F}_{\text{CRS}}$-hybrid protocol that UC-realizes $\mathcal{F}_{\text{ZK}}$ for any NP relation is given in [CLOS02]. The protocol uses trapdoor permutations, and only withstands non-adaptive party corruptions. (Adaptive security is guaranteed if the prover erases its local state before sending out the proof.) In [BCNP04] it is shown how to UC-realize $\mathcal{F}_{\text{ZK}}$ by non-interactive, $\mathcal{F}_{\text{KS}}$-hybrid protocols with non-adaptive corruptions.

Using $\mathcal{F}_{\text{ZK}}$. As shown in [CLOS02], it is possible to realize any "well-formed" two-party ideal functionality by $\mathcal{F}_{\text{ZK}}^R$-hybrid protocols for some NP relation R. Their protocol assumes existence of one-way functions, and withstands adaptive party corruption with no data erasures. (Essentially, an ideal functionality is well-formed if all its outputs are delayed, its code depends on the identities of the corrupted parties only in order to determine the information revealed to these parties, and furthermore it reveals all past internal states whenever all participants are corrupted.)

Claim 32 ([CLOS02]) *If one-way functions exist then there exists an NP relation R such that any well-formed standard corruption ideal functionality can be realized by $\mathcal{F}_{\text{ZK}}^R$-hybrid protocols. This holds even in the presence of adaptive corruptions and if no data erasure is possible.*

7.3.4 Oblivious Transfer

Oblivious Transfer (OT) [R81, EGL85] is a task for two parties, a sender with input $x_1, ..., x_l$, and a receiver with input $i \in \{1, .., l\}$. The receiver should learn x_l (and nothing else) and the sender should learn nothing. OT was shown to be complete for general function evaluation, in the sense that any function can be securely evaluated given ideal access to OT [K89].

In the present framework, a secure OT protocol is a protocol that UC-realizes the "ideal OT" functionality, $\mathcal{F}_{\text{OT}}$, presented in Figure 25. The functionality is parameterized by l, the number of inputs to be provided by the sender, and the domain D of each input. (Using standard terminology, $\mathcal{F}_{\text{OT}}^{l,D}$ captures 1-out-of-l OT of elements in domain D.) Upon receiving input $(\texttt{Sender}, sid, x_1, ..., x_l)$ from the sender S, it first verifies that the identities of the sender and the receiver R are encoded in the SID and that $x_i \in D$ for all i. Then, it records $x_1, ..., x_l$ and notifies the adversary and the receiver R that an interaction has been initiated. When R provides input $(\texttt{Receiver}, sid, i)$ with $i \in [l]$, $\mathcal{F}_{\text{OT}}^{l,D}$ generates a *secret delayed output* x_l to R. (That is, the adversary can delay the delivery of x_l, but it does not learn this value.) When either party is corrupted, $\mathcal{F}_{\text{OT}}^{l,D}$ reveals to the adversary the input of that party. If in addition the corrupted party is the sender, and the receiver did not yet receive its output, then $\mathcal{F}_{\text{OT}}^{l,D}$ lets the adversary to substitute the sender's input $x_1, ..., x_l$ with new values. (This captures the fact that if the sender is corrupted before the protocol execution is complete, then it might still be able to influence the output of the receiver, albeit obliviously of the receiver's input.)

As usual, $\mathcal{F}_{\text{OT}}$ is defined for a single instance, namely the transfer of a single value from S to R. Security in a multi-instance setting is guaranteed via the UC theorem. Finally, we note that the present formulation of $\mathcal{F}_{\text{OT}}^{l,D}$ is geared towards the case where the sender is the initiator of the execution. Indeed, in this case the receiver learns the SID from the execution itself and no initial coordination is necessary. A formulation geared towards the case where the receiver initiates the interaction is analogous.

Functionality $\mathcal{F}_{\text{OT}}^{l,D}$

$\mathcal{F}_{\text{OT}}^{l,D}$ proceeds as follows, parameterized by a number $l \in \mathbf{N}$ of inputs and an input domain D.

1. Upon receiving input $(\texttt{Sender}, sid, x_1, ..., x_l)$ from party S, verify that $sid = (S, R, sid')$ for some identity R, and that each $x_i \in D$; else ignore the input. Next, record $x_1, ..., x_l$ and generate a public delayed output $(\texttt{Sender}, sid)$ to R. Ignore further $(\texttt{Sender}, ...)$ inputs.
2. Upon receiving input $(\texttt{Receiver}, sid, i)$ from R, where $i \in [l]$, wait until a value $x_1, ..., x_l$ is recorded, and then send a private delayed output $(\texttt{Output}, sid, x_i)$ to R and halt.
3. Upon receiving a message $(\texttt{Corrupt}, sid, P)$ from the adversary, where $P \in \{S, R\}$, send $x_1, ..., x_l$ to the adversary. Furthermore, if S is corrupted, the adversary now provides values $x'_1, ..., x'_l$ with each $x'_j \in D$, and no output was yet written to R, then output $(\texttt{Output}, sid, x'_i)$ to R and halt.

Figure 25: The Oblivious Transfer functionality, $\mathcal{F}_{\text{OT}}$

It follows from Claim 32 that $\mathcal{F}_{\text{OT}}^{l,D}$ can be realized by $\mathcal{F}_{\text{ZK}}$-hybrid protocols under standard hardness assumptions, for any l that is polynomial in the security parameter, and any domain D where elements can be represented in polynomial time. Furthermore, there is a simple protocol for realizing $\mathcal{F}_{\text{COM}}$ given access to $\mathcal{F}_{\text{OT}}^{2,\{0,1\}}$ (namely to 1-out-of-2 OT of bits), thus demonstrating that the unrealizability results for $\mathcal{F}_{\text{COM}}$ apply to $\mathcal{F}_{\text{OT}}$ as well:

Claim 33 *There exists a $\mathcal{F}_{\text{OT}}^{2,\{0,1\}}$-hybrid protocol that UC-realizes $\mathcal{F}_{\text{COM}}$ with statistical security.*

Proof: Consider the following protocol:

1. On input $(\texttt{Commit}, sid, b)$, for $b \in \{0,1\}$, the committer chooses $2k$ random bits $\{a_{i,j}\}_{i\in\{0,1\},j\in[k]}$, such that $a_{0,j} + a_{1,j} = b$ for all j. Next, the committer invokes k instances of $\mathcal{F}_{\text{OT}}^{2,\{0,1\}}$, where in the jth instance it obliviously transfers one out of $(a_{0,j}, a_{1,j})$ to the verifier. That is, for each $j \in [k]$, the committer passes input $(\texttt{Sender}, sid|j, a_{0,j}, a_{1,j})$ to $\mathcal{F}_{\text{OT}}^{2,\{0,1\}}$. (Here $|$ denotes string concatenation.)

2. Having received an output $(\texttt{Sender}, sid|j)$ from $\mathcal{F}_{\text{OT}}^{2,\{0,1\}}$, the verifier inputs $(\texttt{Receiver}, sid|j, r_j)$ to $\mathcal{F}_{\text{OT}}^{2,\{0,1\}}$, where $r_j \overset{R}{\leftarrow} \{0,1\}$. Upon receiving output $(\texttt{Output}, sid|j, a_{r_j}, j)$ from $\mathcal{F}_{\text{OT}}^{2,\{0,1\}}$ for all $j \in [k]$, the verifier records $\{(r_j, a_{r_j})\}_{j\in[k]}$ and outputs $(\texttt{Receipt}, sid)$.

3. Having received input $(\texttt{Open}, sid)$, the committer sends $\{a_{0,j}, a_{1,j}\}_{j\in[k]}$ to the verifier. This transmission is done in authenticated way. (One way to obtain authentication is of course to use $\mathcal{F}_{\text{AUTH}}$. Alternatively, use $\mathcal{F}_{\text{OT}}^{2,\{0,1\}}$ itself and let the first transmitted value (i.e., x_1) be the actual information to be transmitted.)

4. Having received $\{a'_{0,j}, a'_{1,j}\}_{j\in[k]}$ from the committer, the receiver verifies that the received values agree with the recorded ones (i.e., that $a_{r_j,j} = a'_{r_j,j}$ for all j), and that the exists a bit b such that $b = a'_{0,j} + a'_{1,j}$ for all j. If so, then the receiver outputs $(\texttt{Open}, sid, b)$.

We claim that the above protocol, denoted OT2COM, statistically UC-realizes $\mathcal{F}_{\text{COM}}$. For this purpose, we construct the following adversary $\mathcal{S}$, that interacts with $\mathcal{F}_{\text{COM}}$ and is geared towards simulating the behavior of the dummy adversary interacting with OT2COM. Here we only give a sketch of the construction of $\mathcal{S}$. Full specification of $\mathcal{S}$ and proof of validity is left as an exercise.

If the receiver is corrupted, then at the commitment stage $\mathcal{S}$ receives the indices $\{r_j\}_{j\in[k]}$ from the environment, and returns completely random values $\{a_{r_j,j}\}_{j\in[k]}$ to the environment (in the name of $\mathcal{F}_{\text{OT}}$). Then, at the opening stage, $\mathcal{S}$ obtains the committed bit b and gives to the environment (in the name of $\mathcal{F}_{\text{AUTH}}$) values $\{a_{1-r_j,j}\}_{j\in[k]}$ so that $b = a'_{0,j} + a'_{1,j}$ for all j.

If the committer is corrupted, then at the commitment stage $\mathcal{S}$ obtains $\{a_{0,j}, a_{1,j}\}_{j\in[k]}$ from the environment, and hands the bit $b = a_{0,1} + a_{1,1}$ as the committed bit to $\mathcal{F}_{\text{COM}}$. (Note that there is no need at this point to check consistency of the k sums $\{a_{0,j} + a_{1,j}\}_{j\in[k]}$.) When the environment initiates the opening stage by sending the values $\{a'_{0,j}, a'_{1,j}\}_{j\in[k]}$, $\mathcal{S}$ chooses random $\{r_j\}_{j\in[k]}$ and performs the checks specified in the code of the verifier with respect to $\{r_j, a_{r_j,j}, a'_{0,j}, a'_{1,j}\}_{j\in[k]}$. If any of these checks fails then $\mathcal{S}$ does nothing (in this case the verifier does not accept the opening of the commitment). If all checks verify and the bit $b' = a'_{0,1} + a'_{1,1}$ agrees with the bit b that $\mathcal{S}$ gave $\mathcal{F}_{\text{COM}}$ at the commitment stage, then $\mathcal{S}$ allows $\mathcal{F}_{\text{COM}}$ to deliver the opening message to the verifier. If all checks verify and $b' \neq b$ then $\mathcal{S}$ outputs a `failed` value.

If either the committer or the verifier is corrupted, the simulator provides the environment with the appropriate simulated internal state of the corrupted party. In addition, if the committer is corrupted and the environment wishes to change the value given to any of the instances of $\mathcal{F}_{\text{OT}}$ (by sending an appropriate message to that instance), the simulator verifies that the $(\texttt{Sender...})$ output was not yet delivered in that instance, and then recomputes (and potentially modifies) the value of the bit b given to $\mathcal{F}_{\text{COM}}$. Notice that this is possible, since the simulator delivers the (Receipt...) output to the verifier only after all the (Sender...) outputs have been delivered.

We claim that, as long as $\mathcal{S}$ does not output failed, the environment's view of the interaction with $\mathcal{S}$ and $\mathcal{F}_{\text{COM}}$ is distributed identically to its view of the interaction with the dummy adversary and protocol OT2COM. Furthermore, $\mathcal{S}$ outputs failed only with probability that is exponentially small in k.

Finally, we note that it is not necessary that the verifier provides its inputs to $\mathcal{F}_{\text{OT}}$ at the commitment stage. That is, consider protocol OT2COM$'$ that it the same as OT2COM except that the verifier generates the (Receipt, sid) output as soon as it receives the (Sender, $sid|j$) outputs from all k instances of $\mathcal{F}_{\text{OT}}$. Then, the verifier provides its indices $\{r_j\}_{j\in[k]}$ to $\mathcal{F}_{\text{OT}}$ only after receiving the opening message from the committer. Then, OT2COM$'$ statistically UC-realizes $\mathcal{F}_{\text{COM}}$. □

7.4 Multiparty tasks

A quick review of prominent multi-party ideal functionalities that have been studied in the literature appears in Section 1.4.5. Here we address two primitives that have been studied extensively, namely verifiable secret sharing (VSS) and secure function evaluation (SFE). The main goal of this section is to exemplify how such multi-party tasks might be captured as ideal functionalities within the present framework. For this purpose, we treat VSS quite differently than SFE: The formulation of the VSS ideal functionality, $\mathcal{F}_{\text{VSS}}$, is geared towards asynchronous networks without guaranteed message delivery. In contrast, the SFE ideal functionality, $\mathcal{F}_{\text{SFE}}$, is geared towards capturing the more "traditional" setting of synchronous networks with guaranteed message delivery.

7.4.1 Verifiable Secret Sharing

Verifiable Secret Sharing (VSS, first proposed in [CGMA85]) is a multi-party primitive that consists of two phases. In the sharing phase, a special party (the dealer, denoted D) shares its inputs among the parties. In the opening phase, the parties reconstruct the dealer's input. It is required that at the end of the sharing phase the dealer's input remains secret. Furthermore, at that time a unique value v should be fixed, such that the value reconstructed by the parties in the opening phase equals v. Finally, if the dealer is uncorrupted then the value reconstructed by the parties should equal the dealer's input. VSS is similar to the one-to-many version of commitment (in which a single party commits to a value to multiple verifiers), with the significant difference that here the shared value should be reconstructible even without the cooperation of the dealer. VSS is typically parameterized by an access structure, namely the collection of sets of parties that are allowed by specification to jointly reconstruct the shared secret. A typical access structure is threshold, namely the collection of all sets whose size is more than some threshold.

Several definitions of VSS exist (e.g., [FM97, BGW88, GM95, C95]), but no definition guarantees secure composition with other protocols (nor with other instances of the same protocol). Also, since VSS is inherently a "two step process", it cannot be naturally captured as secure evaluation of some function. (Indeed, it is possible to construct valid VSS protocols that do not evaluate any function.)

We propose an ideal functionality, $\mathcal{F}_{\text{VSS}}$, that is aimed at capturing the VSS primitive within the present framework. $\mathcal{F}_{\text{VSS}}$ is presented in Figure 26. The sharing phase is modeled by having $\mathcal{F}_{\text{VSS}}$ receive an input (Share, sid, x), from some party D (the dealer), where x is the value committed to. The SID sid is expected to encode the identities of the dealer D and the set $\mathcal{R}$ of receivers.

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

1. Upon receiving an input $(\texttt{Share}, sid, x)$ from D, verify that $sid = (D, \mathcal{Q}, sid')$ where $\mathcal{Q} = \{Q\}$ is an access structure, namely a collection of sets of identities $Q \subset \{0,1\}^*$; else halt. Next, record x and generate a public delayed output $(\texttt{Shared}, sid)$ to the parties in $\mathcal{R} = \cup_{Q \in \mathcal{Q}} Q$. Once x is recorded, ignore any subsequent $\texttt{Share}$ inputs.
2. Upon receiving an input $(\texttt{Open}, sid)$ from party P, add P to the (initially empty) list of parties that ask to open the secret. If there is a set $Q \in \mathcal{Q}$ where all the parties in Q have provided an $(\texttt{Open}, sid)$ input, and there is a recorded shared value x, then generate a public delayed output $(\texttt{Open}, sid, x)$ to the parties in $\mathcal{R}$.
3. Upon receiving a message $(\texttt{Corrupt}, sid, P)$ from the adversary, add P to the list of parties that ask to open the secret. Furthermore, if P is the dealer D then send x to the adversary. Next, if the adversary provides a value x', and the $\texttt{Shared}$ output was not yet written on the tape of any uncorrupted party in $\mathcal{R}$, then change the recorded value to x'.

Figure 26: The Verifiable Secret Sharing functionality, $\mathcal{F}_{\text{VSS}}$

Here $\mathcal{R}$ is represented as an *access structure,* namely as a collection $\mathcal{Q} = \{Q\}$ of sets of identities of parties. Any such set $Q \subset \{0,1\}^*$ represents a quorum of parties that Will be allowed to reconstruct the secret. (We do not specify a specific representation of the access structure $\mathcal{Q}$. Still, when the protocol is PPT the representation must be polynomial in the security parameter.) In response, $\mathcal{F}_{\text{VSS}}$ records x, and lets the parties in $\mathcal{R}$ and the adversary $\mathcal{S}$ know that D has committed to some value, and that this value is associated with SID *sid*. This is done by generating a *public delayed output* $(\texttt{Receipt}, sid)$ to the parties in $\mathcal{R}$.

The opening phase is initiated as soon as a quorum $Q \in \mathcal{Q}$ of recipients have asked to open the secret. At this point, $\mathcal{F}_{\text{VSS}}$ generates a *public delayed output* $(\texttt{Open}, sid, x)$ to the parties in $\mathcal{R}$. This means that the adversary learns the opened value first, and can arbitrarily delay the output delivery to each individual party. (It is also possible to generalize $\mathcal{F}_{\text{VSS}}$ to allow disclosing the shared value only to some subset of $\mathcal{R}$.)

When the adversary corrupts a party P, and if $P \in \mathcal{R}$, then $\mathcal{F}_{\text{VSS}}$ records P as a party that requests to open the secret. (Although such a request was not explicitly made, this only makes the functionality more relaxed without losing in security. It also simplifies the interface.) If P is the dealer D then $\mathcal{F}_{\text{VSS}}$ reveals the recorded value x to the adversary. In addition, if the adversary has not yet instructed to output the $\texttt{Shared}$ value to any uncorrupted party in $\mathcal{R}$, then $\mathcal{F}_{\text{VSS}}$ allows the adversary to modify the shared value. (As in $\mathcal{F}_{\text{COM}}$, this last stipulation captures the fact that the shared value is fixed only once the first uncorrupted party completes the sharing phase, thus if the dealer is corrupted earlier then the adversary might still be able to modify the shared value.

As mentioned above, the formulation of $\mathcal{F}_{\text{VSS}}$ is geared towards realization in completely asynchronous networks with no message delivery guarantees. For instance, it is not guaranteed that all parties complete the sharing phase before the secret is opened. Furthermore, it is not guaranteed that all parties (or even any party) actually receive the opened secret. In order to reflect typical guarantees provided by VSS in a synchronous setting, $\mathcal{F}_{\text{VSS}}$ should be strengthened accordingly. One potential way to strengthen $\mathcal{F}_{\text{VSS}}$ is to explicitly require that all parties receive the $(\texttt{Shared}, sid)$ output before the secret is opened (even to the adversary). Another strengthening is to have $\mathcal{F}_{\text{VSS}}$ provide an additional $\texttt{Opening Complete}$ output to all parties in $\mathcal{R}$ once the $(\texttt{Open}, sid, x)$ output was delivered to all parties in $\mathcal{R}$. Yet another potential strengthening is to guarantee delivery of outputs by having the functionality respond to "output queries" by the parties. (This last technique is exemplified in the formulation of $\mathcal{F}_{\text{SFE}}$ below.) It should be noted that such an augmented version of $\mathcal{F}_{\text{VSS}}$ cannot be realized without access to some synchronizing ideal functionality.

On the feasibility of realizing $\mathcal{F}_{\text{VSS}}$. Claim 32 implies that $\mathcal{F}_{\text{VSS}}$ can be realized by $\mathcal{F}_{\text{CRS}}$-hybrid protocols for any polynomially representable access structure. Furthermore, it is not hard to verify that the the VSS protocols of [BGW88, FM97] UC-realize $\mathcal{F}_{\text{VSS}}$ having access to synchronous communication (e.g., $\mathcal{F}_{\text{SYN}}$), as long as less than a third of the parties are corrupted, and for an access structure representing a threshold of more than two thirds of the parties. In [B91, RB89] these results are improved to adversaries that corrupt any minority of the parties and any threshold that represents a majority of the parties. Since $\mathcal{F}_{\text{VSS}}$ does not provide guaranteed delivery, all these results apply also to the asynchronous setting.

7.4.2 Secure Function Evaluation

Secure Function Evaluation (SFE) is a multi-party primitive where each party P_i out of $P_1, ..., P_n$ has an input value x_i, and obtains an output value $f(x_1, ..., x_n, r)_i$, where $f : (\{0,1\}^*)^n \times R \to (\{0,1\}^*)^n$ is a given n-party function and $r \overset{\text{R}}{\leftarrow} R$. Several definitions for SFE exist, e.g. [GL90, MR91, B91, C00]. In typical settings for studying SFE, the number and identities of the participants are known in advance, and the communication is synchronous. We concentrate on such settings.

We propose an ideal functionality that is aimed at capturing the typical security requirements from SFE. (In particular, we aim at capturing the requirements formalized in [C00].) The functionality, $\mathcal{F}_{\text{SFE}}$, is presented in Figure 27. $\mathcal{F}_{\text{SFE}}$ is parameterized by an n-party function f as defined above. It also expects its SID to specify an ordered set $\mathcal{P} = P_1, ..., P_n$ of n parties. (As usual, the SID is determined by the initiator, namely the first party in $\mathcal{P}$ to provide input to the functionality.) Upon receiving input (Input, sid, v) from party P_i, $\mathcal{F}_{\text{SFE}}$ records v in variable x_i, and notifies the adversary that P_i provided input. Once all the currently uncorrupted parties provide their inputs, $\mathcal{F}_{\text{SFE}}$ sets the unrecorded x_i values to a default value $\perp$, chooses $r \overset{\text{R}}{\leftarrow} R$, and lets $(y_1, ..., y_n) \leftarrow f(x_1, ..., x_n, r)$. Finally, when a party P_i asks for its output value (by providing an (Output, sid) input), $\mathcal{F}_{\text{SFE}}$ responds with the output value y_i. (If y_i is not yet computed then $\perp$ is returned.)

Upon receiving from the adversary a corruption message for party P_i, $\mathcal{F}_{\text{SFE}}$ outputs a corruption message to P_i and discloses P_i's input to the adversary. Also, from now on, $\mathcal{F}_{\text{SFE}}$ lets the adversary modify x_i and, when asked, discloses P_i's output to the adversary.

Note that, as soon as all uncorrupted parties provide inputs, it is guaranteed that all uncorrupted parties will be able to obtain their outputs by querying $\mathcal{F}_{\text{SFE}}$. Also, while the adversary can control the inputs of the corrupted parties, it does so independently of the inputs of the uncorrupted parties. Furthermore, all the output values are computed at the same time (when the first party obtains its output), and based on the same inputs and random input. Once the output values are computed, modifying the input values has no effect on the outcome of the computation. Also, the present formulation of $\mathcal{F}_{\text{SFE}}$ assumes that all parties know the *sid* in advance, or run a preliminary agreement protocol to determine the *sid*. Alternatively, it is possible to formulate $\mathcal{F}_{\text{SFE}}$ so that all parties, other than the initiator, learn the *sid* from $\mathcal{F}_{\text{SFE}}$ before providing their inputs. Such a formulation eliminates the need to agree on the *sid* in advance.

Functionality $\mathcal{F}_{\text{SFE}}^f$

$\mathcal{F}_{\text{SFE}}$ proceeds as follows, given a function $f : (\{0,1\}^* \cup \{\perp\})^n \times R \to (\{0,1\}^*)^n$. At the first activation, verify that $sid = (\mathcal{P}, sid')$, where $\mathcal{P}$ is an ordered set of n identities; else halt. Denote the identities $P_1, ..., P_n$. Also, initialize variables $x_1, ..., x_n, y_1, ..., y_n$ to a default value $\perp$. Next:

1. Upon receiving input (`Input`, sid, v) from some party $P_i \in \mathcal{P}$, set $x_i = v$ and send a message (`Input`, sid, P_i) to the adversary.
2. Upon receiving input (`Output`, sid, v) from some party $P_i \in \mathcal{P}$, do:
   (a) If x_i has been set for all parties P_i that are currently uncorrupted, and $y_1, ..., y_n$ have not yet been set, then choose $r \overset{\text{R}}{\leftarrow} R$ and set $(y_1, ..., y_n) \leftarrow f(x_1, ..., x_n, r)$.
   (b) Output y_i to P_i.

Figure 27: The Secure Function Evaluation functionality for evaluating an n-party function f

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 signature, encryption, zero-knowledge, etc.; agreement primitives such as broadcast and multicast with various levels of security, and more specific tasks such as electronic voting, electronic commerce applications, and privacy preserving distributed database operations.

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 13) 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, DDMRS06], which concentrate on the task of two-party function evaluation given authenticated communication. Still, the question remains open in many other interesting settings.

Finding better set-up assumptions. Assuming trusted set-up is essential in cryptographic protocol design and analysis, in the sense that hardly any useful task can be realized with no trusted set-up whatsoever. For instance, some basic form of initial trusted communication is essential for guaranteeing any reasonable form of authenticated communication. The present framework adds another dimension to the picture, in that many useful tasks were shown to not be possible even if authenticated communication is available. On the other hand, we know that certain set-up assumptions, such as the common reference string plus key registration assumption, or alternatively the key setup assumption, or alternatively timing assumptions, suffice for general feasibility results [CLOS02, BCNP04, LPT04, DPW05]. An interesting research direction is to find other assumptions, potentially of a different flavor and more readily realizable in practice, that would suffice for meaningful feasibility together universal composability. A related goal is to find general *characterizations* of ideal functionalities that would suffice, when treated as a setup assumption, for general feasibility results in the present framework.

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. One very attractive feature of this approach is that it enables partitioning a large system to very small components where automated analysis is feasible, while guaranteeing overall security of the entire system. This way, it is conceivable to have formal and automated 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, ILM05]. 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, Rafael Pass, Birgit Pfitzmann, Tal Rabin, Charlie Rackoff, Phil Rogaway, Victor Shoup, Paul Syverson and Michael Waidner. In particular, it was Daniele who proposed to keep the basic framework minimal and model subsequent abstractions as ideal functionalities within the basic model.

References

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

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

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

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

[A04] J. Almansa. The Full Abstraction of the UC Framework. BRICS, Technical Report RS-04-15 University of Aarhus, Denmark, August 2004.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[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.

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

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

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

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

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

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

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

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

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

[c00] R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology,* Vol. 13, No. 1, winter 2000.

[c01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. (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.

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

[cddim04] R. Canetti, I. Damgaard, S. Dziembowski, Y. Ishai, T. Malkin. On Adaptive vs. Non-adaptive Security of Multiparty Protocols. *J. Cryptology,* 17(3): 153-207. 2004.

[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.

[chklm05] R. Canetti, S. Halevi, J. Katz, Y. Lindell, P. Mackenzie. Universally Composable Password-Based Key Exchange. Eurocrypt '05, 2005. Extended version at http://eprint.iacr.org/2005/196.

[ch04] R. Canetti and J. Herzog. Universally Composable Symbolic Analysis of Cryptographic Protocols (The case of encryption-based mutual authentication and key exchange). Eprint archive, http://eprint.iacr.org/2004/334.

[ck01] R. Canetti and H. Krawczyk. Analysis of key exchange protocols and their use for building secure channels. Eurocrypt '01, 2001. Extended version at http://eprint.iacr.org/2001/040.

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

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

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

[CKL03] R. Canetti, E. Kushilevitz, Y. Lindell. On the Limitations of Universally Composable Two-Party Computation without Set-up Assumptions. *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.

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

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

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

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

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

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

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

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

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

[DPW05] Y. Dodis, R. Pass and S. Walfish. Fully Simulatable Multiparty Computation. Manuscript, 2005.

[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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[K05] R. Küsters. Manuscript, 2005.

[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.

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

[Lo96] G. Lowe. Breaking and fixing the Needham-Schröder public-key protocol using CSP and FDR. *2nd International Workshop on Tools and Algorithms for the construction and analysis of systems,* 1996.

[Ly96] N. Lynch. *Distributed Algorithms.* Morgan Kaufman, San Francisco, 1996.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[sh99] V. Shoup. On Formal Models for Secure Key Exchange. manuscript, 1999. Available at: http://www.shoup.org.

[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 in the 1/6/2005 version from the 10/2001 version. 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 (i.e., ITIs) 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.

Additional changes in the 1/28/2005 version. 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 10, both in Section 4.3. Thanks to Dennis Hofheinz for pointing out the shortcomings of the previous formulation of PPT ITMs.

Additional changes in the 12/2005 version. Many discussions are updated and added. In addition, the main technical changes are:

1. The notions of security with respect to the dummy adversary and with black-box simulation were modified. Indeed, the proof of equivalence of security with dummy adversary and standard UC security was flawed, and a change in the definition was needed. See more details in Section 4.3. (Thanks to Ralf Küsters for pointing out the flaw.)

   A related change is that in prior versions the definition of security allowed the adversaries to be A-PPT rather than PPT. Here all ITMs, including the adversaries, should be PPT (namely, their running time is bounded by a function only of the input length and the security parameter.) This simplifies the definition somewhat.

2. A more concrete treatment of UC security with respect to actual numeric parameters is added, parameterizing two main quantities: the emulation slack, namely the probability of distinguishing between the emulated and the emulating executions, and the simulation overhead, namely the difference between running times of the adversaries in the compared executions.

3. Formulations of ideal functionalities for most of the primitives considered in the 10/2001 version, and some additional primitives, are added. The formulations of practically all these primitives are updated in a number of ways, see more details in the relevant sections. Still, the spirit of the formulations remains unchanged. Some of the ideal functionalities included in the 1/2005 versions were also updated.