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

ECCC

A unified framework for analyzing security of protocols

(Preliminary Version)

Ran Canetti*

December 22, 2000

Abstract

Building on known definitions, we present a unified general framework for defining and analyzing security of cryptographic protocols. The framework allows specifying the security requirements of a large number of cryptographic tasks, such as signature, encryption, authentication, key exchange, commitment, oblivious transfer, zero-knowledge, secret sharing, general function evaluation, and more. Furthermore, within this framework security of protocols is preserved under general composition with any other set of protocols that may be running *concurrently* in the same system. This holds in a number of standard models of computation, including the challenging setting of asynchronous networks where the communication is public and security holds only for computationally bounded adversaries.

Indeed, the proposed framework allows for modular design and analysis of complex protocols from relatively simple building blocks. Moreover, secure protocols are guaranteed to maintain their functionality within any application, even when an unbounded number of protocols are running concurrently in an adversarially controlled manner.

Definitions of security in this framework are often more stringent than other definitions. Nonetheless, we show that in many cases they are satisfied by known protocols. (In fact, practically *any cryptographic task* can be realized in the synchronous version of the above setting, as long as only a minority of the participants are corrupted.) In other cases satisfying the definitions is left open.

Keywords: cryptographic protocols, security analysis of protocols, concurrent composition.

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

ISSN 1433-8092

Contents

1 Introduction **1**
- 1.1 Other definitional efforts . . . 2
- 1.2 The proposed framework . . . 3
- 1.3 General applicability and satisfiability . . . 5
- 1.4 Additional related work . . . 6

2 Overview of the framework **7**
- 2.1 The basic framework . . . 7
- 2.2 The composition theorem . . . 10

3 Preliminaries and protocol syntax **11**
- 3.1 Discussion . . . 13

4 Defining security of protocols **15**
- 4.1 The "real-life" model of computation . . . 15
- 4.2 The ideal process . . . 15
- 4.3 Definition of security . . . 17
- 4.4 Extensions and variants . . . 19

5 Concurrent composition of secure protocols **24**
- 5.1 Preparing the ground . . . 24
- 5.2 The composition theorem . . . 27
- 5.3 Discussion . . . 28
- 5.4 Proof of the composition theorem . . . 29
  - 5.4.1 Proof outline . . . 29
  - 5.4.2 A detailed proof . . . 31

6 Realizing general ideal functionalities **35**

7 Some ideal functionalities **39**
- 7.1 Authenticated Communication . . . 39
- 7.2 Key Exchange . . . 40
- 7.3 Secret Communication (Encryption) . . . 41
- 7.4 Digital Signatures . . . 45
- 7.5 Commitment . . . 49
- 7.6 Oblivious Transfer . . . 50
- 7.7 Zero Knowledge . . . 51
- 7.8 Verifiable Secret Sharing . . . 52
- 7.9 Secure Function Evaluation . . . 53

8 Future directions **55**

1 Introduction

Rigorously demonstrating that a protocol "does its job" 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" by demonstrating that it satisfies the definition of security in the devised mathematical model.

However, coming up with a good mathematical model for representing protocols, and even more so formulating an appropriate definition of security, turns out to be a non-trivial and tricky business. The model should be rich enough to represent a large variety of real-life adversarial behaviors, and the definition should guarantee that the intuitive notion of security is captured, for any adversarial behavior under consideration. The formalization should be as clear and easy to work with as possible. Most importantly, a useful definition must guarantee that security is maintained when a 'secure protocol' is used as a component within a larger system.

Traditionally, cryptographic primitives were defined as stand-alone protocol problems. This allowed for relatively concise problem statement and simple analysis of protocols. However, in many cases it turned out that the initial definition was insufficient in more complex contexts, and especially when coming to deploy the defined primitive within a larger system. Examples include encryption (where semantic security [GM84] was later augmented with several flavors of security against chosen ciphertext attacks, e.g. [NY90, DDN91, RS91, BDPR98]), commitment (where the original notions were augmented with some flavors of non-malleability [DDN91, DIO98, FF00]), Zero-Knowledge protocols (where the original notions [GMRa89, GO94] were shown not to be closed under parallel and concurrent composition [GK88, F91, DNS98]), Key Exchange [BR93, BPRR95, CK00], Oblivious Transfer [R81, EGL85, GM00], and more.

Even today, we have no guarantee that many existing definitions of cryptographic primitives are sufficient when protocols are used as components of a more complex computing environment. In other words, in many cases we have no way of guaranteeing that security of a given protocol is preserved under *composition* with other protocols, or more generally when used within a larger system. Furthermore, we seem to be lacking a framework in which to argue and demonstrate such properties.

Several general definitions of secure protocols were developed over the years, e.g. [GL90, MR91, B91, BCG93, PW94, C00, HM00, PSW00, DM00, PW00]. Some of these definitions were shown to be closed under natural composition operations. These definitions are obvious candidates for such a general framework. However, the composition operations considered fall short of guaranteeing general secure composition of cryptographic protocols, especially in settings where security holds only for computationally bounded adversaries and protocols may be running concurrently. Moreover, many of these works choose to concentrate on the task of *secure function evaluation* which, in spite of its generality, does not capture the requirements of many cryptographic primitives. (Secure function evaluation is the task where a set of parties wish to jointly compute a known function of their secret inputs. This task serves as a concrete "benchmark" for many definitional works.) An exception is the work of [PW94, PSW00]; see more details below.

This work proposes a new framework for representing and analyzing cryptographic protocols. The framework is based on and extends other definitional works, mainly those of [PSW00, DM00, C00]. It allows for concise and clear specification of the security requirements of general "reactive tasks" (or *ideal functionalities,* in the terminology of this work) and for a definition of what it means for a protocol to "securely realize" a given ideal functionality. Practically any cryptographic

protocol-problem can be represented in this way. Indeed, we demonstrate how a number of well known protocol-problems can be represented in this framework. This is the first time that such a breadth of problems are cast within a single framework.

Furthermore, within this framework, security of protocols is preserved under a general composition operation (namely, *modular composition*) even when protocols are running concurrently. Therefore, here a protocol that securely realizes some task is guaranteed to maintain its security (in a well-defined sense) even when it is running as a sub-protocol of an arbitrary and complex multi-party application. Results of the same nature were shown in [MR91, C00, DM00, PW00], for more restricted settings. This is the first time that such a property is demonstrated in the realistic setting where the communication is open, security holds only against computationally bounded adversaries, and the number of protocols running concurrently is not bounded a-priori. Consequently, this work provides the first definitions of a number of cryptographic tasks (including secure function evaluation as a special case) that are closed under modular concurrent composition in the above setting.

Definitions of security in the proposed framework tend to be more stringent than other definitions of security. Nonetheless, in many cases they are satisfied by existing and natural protocols. In fact, in some standard multi-party settings practically *any ideal functionality* is securely realizable using known techniques. In other cases satisfiability of the definitions is left open.

1.1 Other definitional efforts

Numerous definitional works on security of protocols have been carried out over the years. The works of [GL90, MR91, B91, C95] are surveyed in [C00]. Here we very briefly review three definitional efforts that are most closely related to the present work. Some other related works are discussed at the end of the Introduction.

The work of Pfitzmann, Schunter, and Waidner. Pfitzmann et. al. [PW94, PSW00, PW00] were the first to formally model the security requirements of general reactive systems. In a series of works that contain many interesting ideas, they model security of reactive systems in an extended finite-state machine model of computation that is essentially equivalent to the I/O automata model of [L96]. The formalization concentrates on a synchronous scheduling of messages and on an adversary that controls a *fixed* set of parties. In particular, they introduce the notion of an honest user that 'sees' the functionality (i.e., the inputs and outputs) of a given system, and say that one system "simulates" another if the honest user cannot tell the difference between the two systems. They also state a composition theorem with respect to their framework; their composition operation has a similar flavor to the one here, but is somewhat more restricted (as explained within).

The work of Canetti. Canetti [C00] presents a definition of secure function evaluation in a variety of computational settings, including the case where the communication is public and the parties to be corrupted are chosen adaptively throughout the computation. (The communication is always synchronous and ideally authenticated.) In that setting, he introduces the notion of an "environment machine" that interacts with the adversary and the parties at certain points during the computation, and says that a protocol securely evaluates a function if no environment can tell whether it is interacting with the protocol or with an *ideal process* for evaluating the function. (The environment machine is somewhat reminiscent of the "honest user" of [PW94, PSW00], but it models a somewhat different entity, and its communication patterns with the system is different.) The definition of [C00] is shown to be closed under a composition theorem similar to the one here,

but only in the *non-concurrent* case where no more than a single protocol execution is running at any point in time.

The work of Dodis and Micali. Dodis and Micali [DM00] were the first to demonstrate a security-preserving concurrent composition operation for a general definition of security. They build on the definition of Micali and Rogaway [MR91] for information-theoretically secure function evaluation in a synchronous network where parties are corrupted adaptively throughout the computation. In that setting, they prove that their definition of security is closed under a general concurrent composition operation similar to the one in this work. 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 for some communication patterns. 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 can eavesdrop to the communication between honest parties.

1.2 The proposed framework

We briefly sketch the proposed framework and highlight some of its properties. A more comprehensive overview is postponed to Section 2. The framework is primarily based on that of [C00]. As there (and in many other works), we first formulate a mathematical model representing the real-life computing environment. This is called the real-life model. Next, in order to capture the security requirements of a given task, we formulate an ideal process for carrying out the task. Then we say that a protocol securely realizes the task at hand if running the protocol in the real-life model amounts to "emulating" the ideal process for that task.

However, here the models of computation and the notion of "emulation" are different in several respects. Let us sketch a few. First, as in [C00, Sec.6], the communication is open and ideally authenticated, and parties get broken into (or, *corrupted*) in an adaptive way throughout the computation. Here, however, we concentrate on a computational model where the scheduling of messages is *asynchronous without guaranteed delivery*. This model seems the most fitting for capturing realistic networks such as the Internet; furthermore, it simplifies the presentation somewhat. (We remark, though, that the framework can be easily adapted to deal with most other standard settings. More details are provided within.)

More importantly, we extend the real-life model of [C00], and in particular the role of the environment machine, as follows. As there, the real-life model includes the parties $P_1, ..., P_n$ running some protocol, the adversary $\mathcal{A}$ and an environment $\mathcal{Z}$. Here, however, $\mathcal{Z}$ takes a much more central role in the computation. First, it provides all inputs to the parties in an adaptive way throughout the interaction, and receives all local outputs from all parties. Second, it may interact with the adversary at any point during the computation (say, after the delivery of each message and after each corruption event). As in [C00], the output of the computation is the output of the environment. This extra power of the environment is essentially what enables security preserving concurrent composition.

Next, we extend the ideal process of [C00] to capture general "reactive tasks", where input values become known throughout the computation, and may depend on previously generated output values. This is done by replacing the trusted party in the ideal process of [C00] with a general algorithmic entity called an ideal functionality. The ideal functionality, which is formally nothing but an interactive Turing machine, interacts directly with the environment (via some simple interfaces) and with the adversary. This way, it is guaranteed that the outputs received by the environment in

the ideal process have the expected properties with respect to the inputs, even when the environment chooses the inputs in an adaptive way, depending on previous outputs.

Finally, we strengthen the notion of "emulation" as follows. In [C00] (and also, implicitly, in [PSW00, PW00]), a protocol π securely emulates a given ideal process if for any real-life adversary $\mathcal{A}$ and any environment machine $\mathcal{Z}$ there exists an adversary $\mathcal{S}$ in the ideal process such that $\mathcal{Z}$ cannot tell whether it is interacting in the real-life model with π and $\mathcal{A}$, or it is interacting in the given ideal process with $\mathcal{S}$. Here we *strengthen* the notion to require that the same ideal-process adversary $\mathcal{S}$ will suffice for all environments $\mathcal{Z}$. (That is, we require that for any $\mathcal{A}$ there exists $\mathcal{S}$ such that no $\mathcal{Z}$ can tell the difference between the two interactions.) This strengthening seems crucial for the concurrent composition theorem to work when security holds only with respect to computationally bounded adversaries, and the number of concurrently running protocols is unbounded. See more details within.

Concurrent modular composition. We show that the following property holds with respect to a protocol ρ that "emulates" the ideal process with ideal functionality $\mathcal{F}$. (We say that such a protocol securely realizes $\mathcal{F}$.) Let π be some arbitrary protocol (we think of π as an "application protocol") that operates in a model where all parties have ideal access to multiple instances of $\mathcal{F}$. That is, in this "hybrid" model the parties, the adversary and the environment interact as in the real-life model, and in addition parties can privately send messages to and receive messages from as many instances of $\mathcal{F}$ as they wish. (The different instances are distinguished via special identifiers, decided upon by π.) The copies of $\mathcal{F}$ are running at the same time and are not aware of each other.

Now, construct the composed protocol π^ρ from π by replacing each call to some new instance of $\mathcal{F}$ with an invocation of a fresh copy of ρ. Similarly, a message sent to an existing instance of $\mathcal{F}$ is replaced with an input given to the corresponding invocation of ρ, and any output of an invocation of ρ is treated as a message received from the corresponding instance of $\mathcal{F}$. We stress that many invocations of ρ may be running concurrently on related inputs. The composition theorem states that protocols π and π^ρ "have the same functionality" (that is, π^ρ "emulates" π.) In particular, if π securely realizes some ideal functionality $\mathcal{G}$ then so does π^ρ.

Following [C00], we call this composition operation "modular composition." Modular composition was first suggested by Micali and Rogaway [MR91] in their setting (they called it *reducibility*) and later adopted in [C00, DM00, PW00]. This is the first time that this theorem is stated in the case where an unbounded number of copies of a protocol are running concurrently and security holds only for computationally bounded adversaries.[1]

Let us present two interpretations of the composition theorem. According to one interpretation (which serves as the main motivation in [MR91, C00, DM00, PW00]), the composition theorem is a tool for modular design and analysis of complex protocols. 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 in a model where ideal evaluation of the sub-tasks is possible. Finally, use the composition theorem to "automatically" compose the designed protocols into a large protocol that realizes the given task from scratch. An example of a context where this interpretation is put to use is the proof of security in [CKOR00].

An alternative interpretation regards the composition theorem as a tool for gaining confidence

[1] Indeed, in [PW00] the composition theorem is stated for the case where only two protocols are running concurrently. While their theorem can be extended to address the case where some *constant* number of protocols are running concurrently, we do not know how to demonstrate secure concurrent composition in their framework for the general case where the number of protocols running concurrently is unbounded.

for the adequacy of a definition of security in a given context. As mentioned at the beginning of the Introduction, in many cases we do not know whether existing definitions of security of common cryptographic tasks are sufficiently strong to guarantee security in complex environments. In contrast, if a protocol ρ securely realizes an ideal functionality $\mathcal{F}$ within the present framework, then the composition theorem guarantees that ρ can be used to substitute $\mathcal{F}$ in essentially *any* setting and for *any* application. This is a strong guarantee.

1.3 General applicability and satisfiability

To demonstrate the general applicability of the framework, we formulate ideal functionalities that capture the security requirements of several known cryptographic primitives. We first address the task of Message Authentication. Here we extend the [CHH00, BCK98] notion of *authenticators* to the present frameworks, and re-cast this notion as composition with protocols that securely realize a specific ideal functionality. (Authenticators are compilers that transform a protocol that assumes authenticated communication into a protocol that provide essentially the same functionality in unauthenticated networks.) Next we provide ideal functionalities that capture the security requirements from Key-Exchange protocols. Protocols that securely realize this functionality provide keys that are as good as "ideally chosen keys" for any higher-level protocol and regardless of other protocols that may be running concurrently in the same system.

Next we address the tasks of Encryption (or, rather, secret message delivery) and Digital Signatures. Securely realizing the Signature ideal functionality turns out to be essentially equivalent to existential security against chosen message attacks as in Goldwasser Micali and Rivest [GMRi88]. In the case of encryption, we separately treat the case of asymmetric (i.e., public-key) and symmetric (i.e., shared-key) protocols. In the asymmetric case, securely realizing the proposed functionality turns out to be closely related (but incomparable) to security against adaptive chosen ciphertext attacks [DDN91, RS91, BDPR98]. In the symmetric case the proposed ideal functionality essentially guarantees a "secure channel" between two parties. In all cases security is guaranteed within any protocol context.

We then proceed to formulate functionalities representing "classic" two-party primitives such as Commitment, Zero-Knowledge, and Oblivious Transfer. These primitives are treated as two-party protocols in a multi-party setting; the composition theorem guarantees that security is maintained under concurrent composition (either with other copies of the same protocol or with other protocols) and under any application protocol. In particular, non-malleability (as coined in [DDN91]) is guaranteed.

Finally, we formulate ideal functionalities that capture traditional multi-party tasks such as Verifiable Secret Sharing and Secure Function Evaluation (SFE) in *synchronous* networks. In particular, we obtain the first definition of protocols for secure function evaluation that is closed under concurrent composition in a setting where all the communication is public. (Two-party SFE is obtained as a special case.)

Satisfiability of definitions in the proposed framework. Definitions of security in the proposed framework tend to be more stringent (i.e., stronger) than other definitions. Moreover, the existing proofs of security of many known protocols do not work in the present framework. Examples include most known Zero-Knowledge protocols, the protocol of [GMW87, G98] and more. This is mainly due to the fact that a common proof-technique, namely black-box simulation with rewinding of the adversary, does not work in the present framework. (Indeed, here the ideal-process adversary has to interact with the environment machine which cannot be "rewinded".)

Nonetheless, we show that practically *any* ideal functionality (except for a syntactically defined sub-class) can be realized using known techniques, as long as only a fraction of the parties involved in the protocol are corrupted. Specifically, our solution assumes that the network contains a set of parties (called "assistant parties") such that only a fraction of these parties can ever be corrupted. The assistant parties have no local inputs or outputs; they only assist other parties in realizing the given functionality. We start with a *synchronous* setting; here we use the construction of [BGW88] with the simplification of [GRR98], and in addition encrypt each message using an appropriate encryption scheme. (A similar technique is used in [HM00] in a somewhat different setting.) The construction can be adapted to an asynchronous setting using the techniques of [BCG93, BKR94].

1.4 Additional related work

A treatment of secure certified mail protocols, based on the framework of [PSW00], is given in [PSW00a]. In a work that is concurrent to and independent of ours, Pfitzmann and Waidner [PW00a] extend the framework of [PSW00] to the case of asynchronous networks and adaptive adversaries, and propose a treatment of public-key encryption. See more details on their treatment of encryption in Section 7.3.

The pioneering work of Dolev, Dwork and Naor [DDN91] points out some important security concerns that arise when running cryptographic protocols within a larger system. In particular, they define and construct encryption schemes secure against chosen ciphertext attacks, non-malleable commitment schemes, and more. That work provides motivation for the present one. In particular, making sure that the concerns pointed out in [DDN91] are answered plays a central role in the present framework.

Hirt and Maurer [HM00] define and construct secure protocols in an information-theoretic setting where the adversaries are unbounded and the communication is ideally private. In that setting, they define a notion of "simulating" a party via a multi-party protocol. That notion is reminiscent of our notion of realizing an ideal functionality.

A large body of work on analyzing security of protocols using techniques for formal verification of computer programs has been carried out over the years (a very partial list of works includes [DY83, BAN90, M94a, KMM94, L96, AG97]). In particular, basing themselves on the π-calculus of [MPW92], Abadi and Gordon [AG97] provide a calculus (called *spi-calculus*) for arguing about security of authentication protocols. This line of analysis does not take into account computational limitations of adversaries. In order to make analysis possible, cryptographic primitives and protocol actions are modeled via symbolic operations on formal variables with well-defined, deterministic, syntactic properties. This in effect amounts to representing cryptographic primitives as "ideal black boxes". Consequently, these works do not capture the realistic and inherent weaknesses of cryptographic primitives as functions on bit-strings and cannot be used to guarantee security of protocols (say, based on some underlying hardness assumption), as is done here. Nonetheless, these works have proven to be instrumental in finding security weaknesses in some protocols and for gaining assurance in the security of others. In addition, these works are typically amenable to automation of the analysis; this is a great advantage.

Another promising definitional approach using general formal methods is pursued by Lincoln, Mitchell, Mitchell and Schedrov [LMMS98, LMMS99]. They introduce a different variant of the π-calculus, in an effort to incorporate random choices and computational limitations of the adversary in the model. In that setting, their approach is similar to ours. They define a notion of *observational equivalence* (which is reminiscent of our notion of emulation), and say that a real-life process is secure if it is observationally equivalent to an "ideal process" where the desired functionality is guaranteed.

Their model does not seem to allow for easy specification of the security requirements of different tasks, nor do they state a composition theorem. However, as is the case with the other works based on formal-methods, this model has the great advantage that it is amenable to automation of the analysis process using existing tools. See more on possible connections between these works and the present one in Section 8.

Organization. Section 2 presents an overview of the framework, definition of security, and composition theorem. Following some preliminary definition in Section 3, the definition of security is presented in Section 4. The composition theorem and its proof are presented in Section 5. Section 6 demonstrates how to securely realize any ideal functionality in some settings, and Section 7 presents and discusses a number of ideal functionalities that capture some known cryptographic primitives. Finally, Section 8 suggests some directions for future research.

2 Overview of the framework

This section presents an overview of the framework, the definition of security, and the composition theorem. An attempt is made to make this overview as self-contained as possible. Nonetheless, some familiarity with [C00] (especially Section 2) is recommended.

2.1 The basic framework

The basic framework sets the syntax of protocols, formalizes the process of protocol execution in the "real-life" model of computation, and then formalizes what it means for a protocol to *securely realize* a given ideal functionality.

Protocol syntax. Following [GMRa89, G95], we represent protocols (or, rather, a set of computer programs running on different computers in a network) as a set of interactive Turing machines (ITMs). Specifically, the input and output tapes model inputs and outputs that are received from and given to other programs running on the same machine, and the communication tapes model messages sent to and received from the network. This natural modeling facilitates studying modular composition of protocols and allows for easy incorporation of computational complexity considerations. (Adversarial entities are also modeled as ITMs; we concentrate on a non-uniform complexity model where the adversaries have an arbitrary additional input, or an "advice".) In Section 3 we briefly discuss other methods for representing protocols and motivate our choice.

The adversarial model. The framework is quite general, and can be applied to practically any model of computation studied in the literature. However, for sake of concreteness we concentrate on one specific model; we chose a model that seems to best capture existing open networks like the Internet (admittedly in a very simplified way). Using standard terminology, the characteristics of this model are:

- The network is asynchronous, without guaranteed delivery of messages.
- The communication is public (i.e., all messages can be seen by the adversary) but ideally authenticated (i.e., messages cannot be modified by the adversary).[2]

[2]Indeed, in realistic networks the communication is typically *unauthenticated,* in the sense that messages may be

- The adversary is adaptive in corrupting parties, and is active (or, *Byzantine*) in its control over corrupted parties.
- The adversary and environment are restricted to probabilistic polynomial time (or, "feasible") computation.

We remark that this adversarial model is reminiscent of the *authenticated-links* model of [BCK98]. Extensions of the treatment to other adversarial models are sketched in Section 4.4.

Protocol execution in the real-life model. We sketch the "mechanics" of executing a given protocol π (run by parties $P_1, ..., P_n$) with some adversary $\mathcal{A}$ and an environment machine $\mathcal{Z}$ with input z. All parties have a security parameter $k \in \mathbf{N}$ and are polynomial in k. The execution consists of a sequence of *activations,* where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$, or some P_i) is activated. The activated participant reads information from its input and incoming communication tapes, executes its code, and possibly writes information on its outgoing communication tapes and output tapes. In addition, the environment can write information on the input tapes of the parties, and read their output tapes. The adversary can read messages off the outgoing message tapes of the parties and *deliver* them by copying them to the incoming message tapes of the recipient parties. The adversary can also corrupt parties, with the usual consequences that it learns the internal information known to the corrupted party and that from now on it controls that party.

The environment is activated first; once activated, it may choose to activate either one of the parties (with some input value) or to activate the adversary. Whenever the adversary delivers a message to some party P, this party is activated next. Once P's activation is complete, the environment is activated. Throughout, the environment and the adversary may exchange information freely using their input and output tapes. The output of the protocol execution is the output of the environment.

Discussion. Several remarks are in order at this point. First, it is stressed that the environment has access only to the input and output tapes of the parties. It does not have direct access to the communication among the parties. In contrast, the adversary only sees the communication among the parties and has no access to their input and output tapes. In other words, the environment only sees the *functionality* (or, the input/output behavior) of the given protocol. The communication is treated as being "internal" to the protocol and not part of its functionality. (Of course, the environment may learn all the communication, and more, by interacting with the adversary. In a way, the definition of security will imply that in a secure protocol this extra information is "useless", since it is "simulatable" by the ideal process adversary.)

Second, recall that the model allows $\mathcal{A}$ and $\mathcal{Z}$ to exchange information freely throughout the interaction, via $\mathcal{A}$'s input and output tapes. 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. We stress that this is not the case. In fact, the status of $\mathcal{A}$ and $\mathcal{Z}$ is not symmetric: while no generality is lost by assuming that $\mathcal{A}$ reveals its entire internal state to $\mathcal{Z}$, the interesting cases will 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. These assertions become evident in the proof of the composition theorem and the proofs of Claims 9 and 10 in Section 7.

adversarially modified en-route. Nonetheless, we find it more useful to assume ideally authenticated channels. For justification and further discussion see Section 4.4 (and also [BCK98, CHH00]).

Another point to be highlighted is that the only external input to this process is the input of $\mathcal{Z}$. This input represents an initial state of the system and in particular includes the inputs of all parties. (In a non-uniform complexity setting this input may be included in the "non-uniform advice" that $\mathcal{Z}$ has anyway access to; we choose to state it explicitly for clarity. In a uniform complexity setting the external input is essential for modeling arbitrary external events.)

The ideal process. Security of protocols is defined via comparing the protocol execution in the real-life model to an *ideal process* for carrying out the task at hand. A key ingredient in the ideal process is the ideal functionality that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM that interacts with the environment and the adversary via a process described below. (Ideal functionalities generalize the multi-party functions of [C00] to functionalities where inputs are being taken throughout the life of the computation, and where inputs may depend on outputs that were generated earlier in the computation.)

More specifically, the ideal process involves an ideal functionality $\mathcal{F}$, an ideal process adversary $\mathcal{S}$, an environment $\mathcal{Z}$ on input z and a set of dummy parties $\tilde{P}_1, ..., \tilde{P}_n$. The dummy parties are fixed and simple ITMS: Whenever a dummy party is activated with input x, it forwards x to $\mathcal{F}$, say by copying x to its outgoing communication tape; whenever it is activated with incoming message from $\mathcal{F}$ it copies this message to its output. $\mathcal{F}$ receives information from the (dummy) parties by reading it off their outgoing communication tapes. It hands information back to the parties either by sending this information to them and letting the adversary deliver this information at a later time, or alternatively by writing this information directly on the incoming communication tapes of the parties. (These two methods of communicating information to the dummy parties represent two different classes of ideal functionalities. See more details in subsequent sections.) The ideal-process adversary proceeds as in the real-life model, except that it does *not* see the contents of the messages sent between $\mathcal{F}$ and the parties. Nonetheless, it can corrupt dummy parties, learn the information they know, and control their future activities.

The order of events in the ideal process is as follows. As in the real-life model, the environment is activated first. As there, parties are activated when they receive new information (here this information comes either from the environment or from $\mathcal{F}$). In addition, whenever a dummy party P sends information to $\mathcal{F}$, then $\mathcal{F}$ is activated. Once $\mathcal{F}$ completes its activation, P is activated again. Also, $\mathcal{F}$ may exchange messages directly with the adversary. It is stressed that in the ideal process there is no communication among the parties. The only "communication" is in fact idealized transfer of information between the parties and the ideal functionality.

Discussion. It may seem at first that specifying the ideal process is not necessary. Instead, define an adversary that does not corrupt some special party, say P_1, and does not eavesdrop to P_1's communication. Now, P_1 can play the role of the ideal functionality: All parties can send their inputs to P_1 and receive their outputs from P_1. This approach is, in principle, sound. Nonetheless, we prefer to define a special-purpose ideal process, for several reasons. First, explicitly defining ideal functionalities and an ideal process for realizing them seems to be a conceptually clearer way to capture the desired functionalities of tasks. (Indeed, in principle there is nothing in common between the ideal process and the real-life computational model.) Furthermore, defining a special-purpose process allows making this process slightly different from the real-life model. As seen in Section 7, these subtle differences (specifically, in the way data is communicated between the ideal functionality and the parties and the direct communication between the ideal functionality and the adversary) allow for simpler and more concise specification of ideal functionalities. More-

over, some of these differences allow defining a class of ideal functionalities (specifically, *immediate* functionalities, see within) that cannot be captured in the real-life process.

Securely realizing an ideal functionality. Let $\mathcal{C}$ be some class of real-life adversaries. (For instance, $\mathcal{C}$ may be the class of adversaries that corrupt only a certain number of parties.) We say that a protocol ρ securely realizes an ideal functionality $\mathcal{F}$ with respect to $\mathcal{C}$ if for any real-life adversary $\mathcal{A} \in \mathcal{C}$ there exists an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running ρ in the real-life process, or it is interaction with $\mathcal{A}$ and $\mathcal{F}$ in the ideal process. This means that, from the point of view of the environment, running protocol ρ is 'just as good' as interacting with an ideal process for $\mathcal{F}$. (In a way, $\mathcal{Z}$ serves as an "interactive distinguisher" between the two processes. Here it is important that $\mathcal{Z}$ can provide the process in question with *adaptively chosen* inputs throughout the computation.)

As discussed in the Introduction, here the order of quantifiers is different than in [C00, Definition 9]. Specifically, there it is only required that for any environment and any real-life adversary there exists an ideal-process adversary that satisfies the desired condition. As noted there, in the case of computationally unbounded adversaries the two formalizations are equivalent. However, the argument used there fails (and we are not aware of any other argument) in the case of polynomial-time adversaries.

The stronger formalization used here is somewhat more satisfying. More importantly, our proof of the concurrent composition theorem (discussed below) uses the stronger formalization in an essential way.

2.2 The composition theorem

The hybrid model. In order to state the composition theorem, and in particular in order to formalize the notion of a protocol that has additional access to an ideal functionality, we first formulate the hybrid model of computation with access to an ideal functionality $\mathcal{F}$ (or, in short, the $\mathcal{F}$-hybrid model). This model is identical to the real-life model, with the following exceptions. In addition to sending messages to each other, the parties may send messages to and receive messages from an unbounded number of copies of $\mathcal{F}$. Each copy of $\mathcal{F}$ is identified via a unique session identifier (SID); all messages addressed to this copy and all message sent by this copy carry the corresponding SID. (The SIDs are chosen by the protocol run by the parties.)

The communication between the parties and each one of the copies of $\mathcal{F}$ mimics the ideal process. That is, once a party sends a message to some copy of $\mathcal{F}$, that copy is immediately activated and reads that message off the party's tape. Furthermore, although the adversary in the hybrid model is responsible for delivering the messages from the copies of $\mathcal{F}$ to the parties, it does not have access to the contents of these messages.

Replacing a call to $\mathcal{F}$ with a protocol invocation. Let π be a protocol in the $\mathcal{F}$-hybrid model, and let ρ be a protocol that securely realizes $\mathcal{F}$ (with respect to some class of adversaries). The composed protocol π^ρ is constructed by modifying the code of each ITM in π so that the first message sent to each copy of $\mathcal{F}$ is replaced with an invocation of a new copy of π with fresh random input, and with the contents of that message as input. Each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of ρ, with the contents of that message given to ρ as new input. Each output value generated by a copy of ρ is treated as a message received from the corresponding copy of $\mathcal{F}$.

If protocol ρ is a protocol in the real-life model then so is π^ρ. If ρ is a protocol in some $\mathcal{G}$-hybrid model (i.e., ρ uses ideal evaluation calls to some functionality $\mathcal{G}$) then so is π^ρ.

Theorem statement. In its general form, the composition theorem basically says that if ρ securely realizes $\mathcal{F}$ w.r.t. some adversary class $\mathcal{C}$ then an execution of the composed protocol π^ρ "emulates" an execution of protocol π in the $\mathcal{F}$-hybrid model. That is, for any real-life adversary $\mathcal{A} \in \mathcal{C}$ there exists an adversary $\mathcal{H}$ in the $\mathcal{F}$-hybrid model such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and π^ρ in the real-life model or it is interacting with $\mathcal{H}$ and π in the $\mathcal{F}$-hybrid model.

A more specific corollary of the theorem states that if π securely realizes some functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model w.r.t. some class $\mathcal{C}$ of adversaries, and ρ securely realizes $\mathcal{F}$ in the real-life model w.r.t. the same class $\mathcal{C}$, then π^ρ securely realizes $\mathcal{G}$ in the real-life model w.r.t. $\mathcal{C}$. (Here one has to define what it means to securely realize functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model. This is done in the natural way.)

Discussion. It may seem sufficient at first to formulate a simplified version of the hybrid model and the composition theorem, where only a single copy of the ideal functionality $\mathcal{F}$ is available (instead of multiple copies, as defined above). One could then handle protocols that use multiple copies of $\mathcal{F}$ by repeated applications of the composition theorem until all copies of $\mathcal{F}$ are replaced by calls to a protocol ρ that securely realizes $\mathcal{F}$. This would work even in the concurrent case, since even the simplified hybrid model would allow the evaluation of $\mathcal{F}$ to run concurrently with the calling protocol, π.

However, in our setting the composition theorem can be safely repeated only a *constant* number of times. This is so, since the complexity of the hybrid model adversary $\mathcal{H}$ may depend on (in fact, it can be any polynomial in) the complexity of $\mathcal{A}$. Thus, if the theorem is applied a non-constant number of times, the resulting $\mathcal{H}$ may not be polynomial in the security parameter. Consequently, the above simplified hybrid model would result in a composition theorem that applies only to the limited case where at most a constant number of concurrent executions of a protocol take place. In contrast, the composition theorem here handles polynomially many concurrent executions, and even cases where the number of concurrent executions is determined by the adversary in an adaptive manner.

3 Preliminaries and protocol syntax

Probability ensembles and indistinguishability. We review the definitions of probability ensembles and indistinguishability. 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 distribution ensembles we consider in the sequel are binary (i.e., over $\{0,1\}$). Furthermore, they are outputs of computations where the parameter a represents input, and the parameter k is taken to be the security parameter. All complexity characteristics of our constructs are measured in terms of k.

Definition 1 *Two binary distribution ensembles X and Y are* indistinguishable *(written $X \approx Y$) if for any $c \in \mathbf{N}$ there exists $k_0 \in \mathbf{N}$ such that for all $k > k_0$ and for all a we have*

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

Interactive Turing machines. We adhere to the standard formalization of algorithms (or, rather, computer programs) in a communication network as interactive Turing machines [GMRa89]. A detailed exposition of interactive Turing machines, geared towards formalizing *pairs* of interacting machines, appears in [G95, Chapter 6.2.1]. For self-containment we recall the definition of [G95], adding some syntax that facilitates handling of the multi-party setting. The definition contains details that are somewhat arbitrary, and indeed do not add much insight on the nature of the problem. Nonetheless, fixing these details is essential for clarity and unambiguity of the treatment.

Definition 2 (Based on [GMRa89, G95]) *An* interactive Turing machine (ITM) *is a Turing machine that has the following tapes:*

- *A (read only)* identity *tape.*
- *A (read only)* security parameter *tape.*
- *A (read only)* random *tape.*
- *A (read only)* input *tape and a (write once*[3]*)* output *tape.*
- *A (read only)* incoming communication *tape and a (write once)* outgoing communication *tape.*
- *A (read and write) work tape.*
- *A (read and write) one-bit* activation *tape.*

The contents of the identity tape of an ITM M is called the identity *of M. The contents of the incoming communication tape is interpreted (using some standard encoding) to consist of a sequence of strings 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 outgoing communication tape is interpreted in the same way, where the first field of each message is called the* recipient *field.*

We say that M is activated *if its activation tape is set to 1. Once activated, M proceeds according to its program (i.e., its transition function) and the contents of its tapes. Once M completes its processing, it resets its activation tape to 0 and enters one of two states: either a* waiting *state, or a* halt *state. If M enters the waiting state then we say that the activation is complete and that M is waiting for the next activation. If M enters the halt state then we say that it has halted; in this case, it does nothing in all future activations. The contents of the output tape of M at the completion of an activation is called the* output *of M from this activation. The messages that appear on the outgoing communication tape of M are called the* outgoing messages *of M at this activation.*

Throughout this work we concentrate on ITMs that operate in (strict) probabilistic polynomial time. That is, we assume that with each ITM M there is an associated constant $c \in \mathbf{N}$ such that in each activation M enters either the halt state or the waiting state within n^c computational steps, where n is the value written on the security parameter tape. Furthermore, M enters the halt state after at most n^c activations. (We use a special tape to bound the running time of an ITM since the standard convention of using the input length would not do in our setting where the input length varies from activation to activation.)

[3] In a write-once tape each cell can be written into only once.

Message-driven protocols. Let $n \in \mathbf{N}$. An n-party message-driven protocol is a collection of n ITMs as in Definition 2, where each ITM has unique identity. (That is, no two ITMs have the same value written on their identity tapes.) We envision that each ITM represents a program to be run on a different physical computer, or "host", in a communication network. (We often use the term party to refer to a host.) The identity of each ITM represents the identity of the host in the network, plus possibly some additional information that differentiates between different programs that run on the same host. The input and output tapes model intra-host communication with other programs that are run on the host, as well as interaction with human users. The incoming and outgoing communication tapes model messages from and to other parties in the network, respectively.

3.1 Discussion

We motivate some of our definitional choices regarding the syntax of ITMs and protocols, and expand on how this syntax is used in subsequent sections.

Motivating the use of ITMs. Interactive Turing machines are only one of many standard mathematical models aimed at capturing computers in a network (or, more abstractly, interacting computing agents). Other models include the π-calculus of Milner, Parrow and Walker [MPW92] (that is based on the λ-calculus as its basic model of computation), the I/O automata of Lynch [L96] (that is based on automata theory) and more. Several reasons motivate our choice to use ITMs as a basis for our framework. First and foremost, ITMs seem to best represent the subtle relationships between communication (often with adversarial scheduling) and the complexity of local computations (which may be randomized). These relationships are at the essence of cryptographic protocols. Furthermore, ITMs seem to best mesh with standard models used in complexity theory (such as standard Turing machines, oracle machines, and circuit families). Lastly, ITMs seem to most faithfully represent the way in which existing computers operate in a network. This faithful representation includes the separation between local inputs/outputs and communication, the identities of parties, the use of a small number of physical communication channels to interact with a large number of other parties, and more.

On the role of the communication tapes. In the case of ITMs defined for the purpose of interactive proof-systems [GMRa89, G95], the main motivation for distinguishing between the communication tapes and the input/output tapes is that the input/output tapes are written only once throughout the execution, whereas the communication tapes keep being overwritten. Here we will not assume that the input remains the same across all activations of the machine. In fact, it will be expected that the contents of the input and output tapes will be different in different activations, and that the output tape will be read (by other ITMs) after each activation. In light of this difference, it may appear that the distinction between communication tapes and input/output tapes is no longer necessary. Nonetheless, keeping the distinction seems to be quite convenient. As seen below, the different tapes represent different types of communication of the ITM with the "outside world". Furthermore, our model allows different adversarial entities to have access to different tapes. Specifically, the environment machine will have access to the input and output tapes (which represent the local inputs and outputs of a program), while the adversary will have access to the communication tapes (which represent messages sent to and received from the network).

ITMs as circuit families. 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 represent (in some predefined way) the internal state of the ITM at the beginning of an activation, plus the contents of the incoming communication tape. The output lines of the circuit represent the internal state of the ITM at the end of the activation, plus the contents of the outgoing communication tape. We usually think of ITMs as uniform-complexity ones, thus restricting attention to uniformly generated circuit families. However, non-uniform circuit families are possible as well (and are often used to model adversarial entities).

Running several sub-protocols simultaneously. In subsequent sections we discuss protocols that run other protocols as "subroutines". That is, we say that some protocol π "invokes protocol ρ". This natural operation implicitly assumes that parties are running several protocols (or *sub-protocols*) simultaneously. There are a number of ways to realize this high-level description of protocols; let us specify one such way for sake of concreteness. The code of the ITM M run by each party is assumed to be structured as follows. There is a "protocol manager" code (representing the "operating system" of that party) and in addition code for each protocol to be run by the party. The protocol manager keeps track of all the protocol invocations (or, *protocol-copies*) that are currently running. Initially there is a single protocol-copy of a "main" protocol that is active. Whenever an existing protocol-copy invokes a new protocol-copy, the protocol manager adds the new protocol-copy to the list of currently active copies. Each active protocol-copy is assumed to have a unique identifier associated with it. (The identifier is decided upon by the protocol manager and given to the protocol-copy upon invocation.)

Each input and each incoming message to M is assumed to include the name of some protocol and an identifier of some protocol-copy. When M is activated with input or incoming message x that specifies some protocol π and protocol-copy i, the protocol manager first checks if there is an active copy of protocol π with identifier i. If such a protocol-copy exists then the protocol manager activates this protocol-copy with input or incoming message x. If no such protocol-copy exists then the protocol manager invokes a fresh copy of protocol π, gives it identifier i, and immediately activates it with input or incoming message x. (It is assumed that the protocol-copies send their identifiers to the protocol-copies they interact with within other parties.)

Protocols with variable number of parties. The above formalization of message-driven protocols has the disadvantage that some bound, n, on the number of participants in the protocol needs to be fixed in advance. An alternative formalization would allow several copies of a given ITM, with different identities, to interact with each other; here the number of copies may grow with time (say, in an adversarially controlled way). We chose the above formalization because of its relative simplicity, and since the difference between the two formalizations seems immaterial to the analysis of protocols. (In particular, note that the programs run by the parties in an n-party protocol are not necessarily aware of n, the number of parties.)

Protocols without unique identities. Our definition of message-driven protocols assumes that the participating ITMs have unique identities. Arguably, this fundamental assumption reasonably captures most existing communication networks. Nonetheless, protocols where the participants do not have unique identities merit separate study.

Adversaries and Ideal Functionalities as ITMs. Although the main goal in defining ITMs is to represent the programs run by the actual computers in a network, it will be convenient to use ITMs also to model adversaries and ideal functionalities. These ITMs will be somewhat different in that

they will have read and write accesses to tapes that are parts of other ITMs.[4] Also, these ITMs will not always use all their tapes. More importantly, adversarial entities are typically modeled via *non-uniform* ITMs (or, equivalently, circuit families). See more details in Section 4.

4 Defining security of protocols

This section presents a definition of protocols that securely realize a given ideal functionality, in a specific computational model (presented in Section 2). First we present the "real-life" model of computation (Section 4.1). Next the ideal process for carrying out a given functionality is presented (Section 4.2), followed by the notion of protocol emulation and the definition of security. Section 4.4 sketches some extensions of the definition, aimed at capturing some other prominent models.

4.1 The "real-life" model of computation

Let $n \in \mathbf{N}$, and let π be an n-party message-driven protocol. Let P_i denote the identity of the ith ITM in π. (This modeling represents a set of n parties, $P_1 ... P_n$, where P_i runs the ith ITM in π. We will often fail to distinguish between the party and the ITM, or program, run by this party.) In addition, the model includes two adversarial entities, called the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$. Both $\mathcal{A}$ and $\mathcal{Z}$ are programs, or ITMs, with interfaces described below. All the participants have (infinitely long) random values written on their random input tapes. In addition, the environment $\mathcal{Z}$ starts with some input value z written on its input tape. ($\mathcal{Z}$'s input represents some initial state of the environment in which the protocol execution takes place. We assume that $\mathcal{Z}$'s input represents all the external inputs to the system, including the local inputs of all parties.) We restrict attention to environments that output a single bit. (As discussed in Section 4.3, no generality is lost by this restriction.) The order of events in an execution of an n-party protocol is described in Figure 1.

We use the following notation. Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \vec{r})$ denote the output of environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and parties running protocol π on security parameter k, input z and random input $\vec{r} = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1 \dots r_n$ as described above (z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$, $r_{\mathcal{A}}$ for $\mathcal{A}$; r_i for party P_i). Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ denote the random variable describing $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \vec{r})$ when $\vec{r}$ is uniformly chosen. Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

4.2 The ideal process

We describe the ideal process for evaluating an ideal functionality. First, however, let us define ideal functionalities.

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 limitations imposed on the acceptable leakage information to the adversary. Technically, an ideal functionality $\mathcal{F}$ is modeled as an ITM (see Definition 2), with the following slight modifications. First, the input and output tapes are not used. Next, in addition to incoming messages whose sender is a party out of $P_1, ..., P_n$, $\mathcal{F}$ takes

[4] Specifically, to write to (resp., read from) a tape of an ITM with identity id, such an ITM will write $(\texttt{write}, id, t, v)$ (resp., $(\texttt{read}, id, t)$ on a special portion of its tape, where t is a value that identifies a specific portion of a tape within the machine M with identity id. Then the value v will be written on the appropriate tape of M (resp., the contents of the appropriate portion of M's tapes will be copied to special portion of the local tape.

Protocol execution in the real-life model

Participants: Parties $P_1, ..., P_n$ running protocol π with adversary $\mathcal{A}$ and environment $\mathcal{Z}$ on input z. All participants have the security parameter k.

1. While $\mathcal{Z}$ has not halted do:

   (a) $\mathcal{Z}$ is activated (i.e., its activation tape is set to 1). In addition to its own readable tapes, $\mathcal{Z}$ has read access to the output tapes of all the parties and of $\mathcal{A}$. Furthermore, $\mathcal{Z}$ may write an arbitrary value on the input tapes of either $\mathcal{A}$ or of *one* party out of $P_1, ..., P_n$. The activation ends when $\mathcal{Z}$ enters either the halting state of the waiting state. If $\mathcal{Z}$ enters the waiting state and it has written a value on the input tape of P_i (or $\mathcal{A}$) then P_i (or $\mathcal{A}$) is activated.

   (b) Once $\mathcal{A}$ is activated, it proceeds according to its program and possibly writes new information on its output tape. In addition to reading its own tapes, $\mathcal{A}$ can read the outgoing message tapes of all the parties. Furthermore, $\mathcal{A}$ can choose to perform one of the following two additional activities:

      i. $\mathcal{A}$ can deliver a message m from party P_i to party P_j. This activity is possible only if the outgoing message tape of P_i contains message m and specifies P_j as the recipient of m, and if m was not delivered before. Delivering m means copying the contents of m from the outgoing message tape of P_i to the incoming message tape of P_j, and adding the identity of P_i as the sender of this message.

      ii. $\mathcal{A}$ can corrupt a party P_i. Upon corruption $\mathcal{A}$ learns the sequence of all past states of P_i, and $\mathcal{Z}$ learns that P_i was corrupted. (Say, all the past states of P_i are written on $\mathcal{A}$'s input tape, and a note that P_i is corrupted is written on $\mathcal{Z}$'s input tape.)[a] Also, from this point on, P_i may no longer be activated. In addition, from this point on $\mathcal{A}$ gets read and write access to all the tapes of P_i and can deliver any message to any (uncorrupted) party, as long as P_i is specified as the sender of that message.

   The next participant to be activated once $\mathcal{A}$ enters its waiting state is determined as follows. If activity 1(b)i was performed then party P_j is activated once $\mathcal{A}$ enters the waiting state. Otherwise, $\mathcal{Z}$ is activated as in Step 1a.

   (c) Once an uncorrupted party P_i is activated (either due to a new incoming message, delivered by $\mathcal{A}$, or due to a new input, generated by $\mathcal{Z}$), it proceeds according to its program and possibly writes new information on its output tape and its outgoing message tape. Once P_i enters the waiting or the halt states, $\mathcal{Z}$ is activated as in Step 1a.

2. The output of the execution is the (one bit) output of $\mathcal{Z}$.

[a] By postulating that the adversary learns not only the current state of the corrupted party but also all the past states, we model a system where parties are not trusted to effectively erase local data. In a setting where erasures are trusted to be effective one may let the adversary see only the *current* state of the corrupted party. See more discussion in [C00].

Figure 1: The order of events in a protocol execution in the real-life model

incoming messages whose sender is the adversary. Similarly, in addition to sending messages to $P_1, ..., P_n$ by writing these messages on its outgoing communication tape, $\mathcal{F}$ may write messages directly on the incoming communication tapes of the parties. Furthermore, $\mathcal{F}$ may write outgoing messages whose recipient is the adversary. (Typically, useful ideal functionalities will have some more structure. To avoid cluttering the basic definition with unnecessary details, further restrictions

on ideal functionalities are postponed to subsequent sections.)

The ideal process. Let $n \in \mathbf{N}$. The ideal process involves an ideal functionality $\mathcal{F}$, an environment $\mathcal{Z}$ and an ideal-process adversary $\mathcal{S}$. In addition, there are n dummy parties $\tilde{P}_1, ..., \tilde{P}_n$; these are (very simple) fixed ITMs, that only forward their inputs to $\mathcal{F}$, and output any value received from $\mathcal{F}$. As in the real-life model, we restrict attention to environments that output a single bit. The order of events in the ideal process is presented in Figure 2.

We use the following notation. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z, \vec{r})$ denote the output of environment $\mathcal{Z}$ after interacting in the ideal process with adversary $\mathcal{S}$ and ideal functionality $\mathcal{F}$, on security parameter k, input z, and random input $\vec{r} = r_{\mathcal{Z}}, r_{\mathcal{S}}, r_{\mathcal{F}}$ as described above (z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$, $r_{\mathcal{S}}$ for $\mathcal{S}$; $r_{\mathcal{F}}$ for $\mathcal{F}$). Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)$ denote the random variable describing $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z, \vec{r})$ when $\vec{r}$ is uniformly chosen. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Remark: On the two methods for generating output. We highlight and motivate the following aspect of the ideal process. The ideal process specifies two methods for the ideal functionality $\mathcal{F}$ to provide the parties with output. The first method is to *send* a value to some party. (This is done by writing a message addressed to that party on the outgoing communication of $\mathcal{F}$.) This way, the ideal-process adversary may deliver this message (without reading it) at any time during the computation; it may also choose not to deliver this message at all, with the effect that the said party will not generate any output. This method of providing parties with output values reflects the "inevitable weaknesses" of the model of computation we work in (asynchronous communication without guaranteed delivery). Indeed, in this model we cannot guarantee that parties will generate output and we cannot in general control the order in which parties generate output.

The second method of providing a party P with an output value is to write this value directly on the incoming communication tape of P. This guarantees that as soon as P is activated, it will see the corresponding output value. Arguably, in our model of communication this method makes sense only if P is also the party that was last activated before $\mathcal{F}$ (i.e., the current activation of $\mathcal{F}$ is caused by a message from P). In this case, this method of providing P with "immediate" output represents subroutines that are executed locally within P, without need for outside communication. We call such ideal functionalities immediate. We remark that, while this method is useful in capturing the functionality of some standard primitives (e.g., signatures, see Section 7.4), if mis-used it may result in functionalities that are not realizable (see Section 6).

4.3 Definition of security

Roughly speaking, we say that a protocol π securely realizes an ideal functionality $\mathcal{F}$ if π emulates the ideal process for $\mathcal{F}$. More specifically, we require that for any real-life adversary $\mathcal{A}$ (from given class of adversaries) there should exist an ideal-process adversary $\mathcal{S}$ such that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ will hold *for any environment* $\mathcal{Z}$. Note that the environment is the same in the real-life model and the ideal process. This may be interpreted as saying that "For any real-life adversary $\mathcal{A}$ in the given class, there should exist an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running π in the real-life model, or with $\mathcal{S}$ and the dummy parties in the ideal process for $\mathcal{F}$." For sake of concreteness, we restrict attention to classes of adversaries that specify the allowed sets of parties to be corrupted. That is, a *class* $\mathcal{C}$ is the set of adversaries that corrupt parties subject to the restriction that the identities of the parties corrupted in each execution belongs a given access structure. Security with respect to adversaries in $\mathcal{C}$ is defined as follows.

The ideal process

Participants: Environment $\mathcal{Z}$ and ideal-process adversary $\mathcal{S}$, interacting with ideal functionality $\mathcal{F}$ and dummy parties $\tilde{P}_1, ..., \tilde{P}_n$. All participants have the security parameter k; $\mathcal{Z}$ has also input z.

1. While $\mathcal{Z}$ has not halted do:

   (a) $\mathcal{Z}$ is activated (i.e., its activation tape is set to 1). In addition to its own readable tapes, $\mathcal{Z}$ has read access to the output tapes of all the (dummy) parties and of $\mathcal{S}$. In addition, $\mathcal{Z}$ may write an arbitrary value on the input tapes of either $\mathcal{S}$ or of *one* dummy party out of $\tilde{P}_1, ..., \tilde{P}_n$. Once this value is written, $\mathcal{Z}$ either halts or enters the waiting state. If $\mathcal{Z}$ enters the waiting state then the corresponding dummy party (or $\mathcal{S}$) is activated.

   (b) Once $\mathcal{S}$ is activated due to a new input generated by $\mathcal{Z}$, it follows its program and possibly writes new information on its output tape. In addition, $\mathcal{S}$ can perform one of the following activities.

      i. $\mathcal{S}$ may deliver a message m from $\mathcal{F}$ to some dummy party $\tilde{P}_i$. Here $\mathcal{S}$ does not have access to the contents of m; it only sees the destination of m. (That is, once $\mathcal{S}$ decides to deliver a message to $\tilde{P}_i$, the contents of m is copied to the incoming communication tape of $\tilde{P}_i$ without further intervention of $\mathcal{S}$.)

      ii. $\mathcal{S}$ may write a message, m, on the incoming communication tape of $\mathcal{F}$. This message appears with sender $\mathcal{S}$.

      iii. $\mathcal{S}$ can corrupt a dummy party $\tilde{P}_i$. Upon corruption, $\mathcal{S}$ learns the history of all past states of $\tilde{P}_i$, and $\mathcal{Z}$ learns that $\tilde{P}_i$ was corrupted. (As seen below, this history is straightforward: it consists of the information received from $\mathcal{Z}$ and from $\mathcal{F}$ throughout the computation.) From this point on, P_i may no longer be activated; also, $\mathcal{S}$ gets read and write access to all the tapes of $\tilde{P}_i$ and may deliver messages whose sender identity is $\tilde{P}_i$ to $\mathcal{F}$.

      If some message was delivered to $\tilde{P}_i$ (or $\mathcal{F}$) in this activation then $\tilde{P}_i$ (or $\mathcal{F}$) is activated once $\mathcal{S}$ enters either the waiting state. Otherwise, $\mathcal{Z}$ is activated.

   (c) Once an (uncorrupted) dummy party $\tilde{P}_i$ is activated it proceeds as follows. First, it copies any new input value x (generated by $\mathcal{Z}$) to its outgoing communication tape (as a message for $\mathcal{F}$). Next, it copies to its output tape any messages it may have received from the ideal functionality $\mathcal{F}$ since its last activation, and enters the waiting state.

      If in this activation $\tilde{P}_i$ wrote some value on its outgoing communication tape then as soon as $\tilde{P}_i$ enters the waiting state $\mathcal{F}$ is activated. Otherwise, $\mathcal{Z}$ is activated next.

   (d) Once $\mathcal{F}$ is activated it follows its program until it enters either the waiting state or the halt state. In addition to its own tapes, $\mathcal{F}$ may read from and write to the communication tapes of all the parties. (That means that $\mathcal{F}$ can either send messages to some $\tilde{P}_i$ by writing the message on its own outgoing message tape, or write the message directly on $\tilde{P}_i$'s incoming communication tape.) In addition, $\mathcal{F}$ may write information on the incoming communication tape of $\mathcal{S}$. In that case, $\mathcal{S}$ is activated once $\mathcal{F}$ enters the waiting state. Otherwise, the party that was last activated before $\mathcal{F}$ is activated again.

   (e) Once $\mathcal{S}$ is activated due to an incoming message from $\mathcal{F}$, it follows its program and possibly writes some information on the incoming communication tape of $\mathcal{F}$. It is stressed that in this type of activation $\mathcal{S}$ cannot deliver messages to or corrupt parties. Once $\mathcal{S}$ enters the waiting state, $\mathcal{F}$ is activated again.

2. The global output is the (one bit) output of $\mathcal{Z}$.

Figure 2: The order of events in the ideal process for a given ideal functionality, $\mathcal{F}$.

Definition 3 *Let $n \in \mathbf{N}$. Let $\mathcal{F}$ be an ideal functionality and let π be an n-party protocol. Let $\mathcal{C}$ be some class of real-life adversaries. We say that π* securely realizes *$\mathcal{F}$* with respect to *$\mathcal{C}$ if for any adversary $\mathcal{A} \in \mathcal{C}$ there exists an ideal-process adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have:*

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}. \tag{1}$$

Remark: On protocols that don't generate outputs. Recall that the ideal process does not require the ideal-process adversary to deliver messages that are sent by the ideal functionality to the dummy parties. This means that, unless the ideal functionality is immediate (i.e., unless it writes output values directly on the tapes of the dummy parties), the ideal process does not guarantee that the dummy parties generate output at all. Consequently, as long as the functionality is not immediate, Definition 3 provides no guarantee that a protocol will ever generate output or "return" to the calling protocol. (In fact, a protocol that "hangs" and never generates output securely realizes any non-immediate functionality.)

Let us motivate this definitional decision. In an asynchronous setting where the adversary has full control over the communication channels there can be no guarantee that any communication goes through. Thus, there can be no general guarantee that a protocol will generate output. In fact, protocols may have different conditions for generating output, and different settings may have different requirements from protocols regarding the conditions under which a protocol is guaranteed not to "hang". Furthermore, these properties of protocols can be analyzed independently of their security properties. Therefore, we choose not to make any requirements regarding the conditions where a protocol is guaranteed to generate output, and concentrate on the security requirements *in the case that the protocol generates output.*

In cases where message delivery is guaranteed (such as in synchronous networks) the ideal process can be modified so as to exclude protocols that do not generate output. See more details in Section 4.4.

Remark: On environments with non-binary outputs. The models of computation and Definition 3 deal only with 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 $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ (that would no longer be binary) be *computationally indistinguishable,* as defined by Yao [Y82A]. It is easy to see, however, that this extra generality results in a definition that is equivalent to Definition 3. We leave the proof as an exercise.

Taking a slightly more abstract point of view, we note that Definition 3 in fact regards the environment itself as a distinguisher of sorts. More specifically, the environment plays the role of an "interactive distinguisher" between two stochastic processes, thus generalizing the notion of Yao to the interactive case.

4.4 Extensions and variants

Definition 3 captures one specific model of computation, as described in Section 2. This section sketches methods for adapting Definition 3 to capture some other prominent models.

Non-blocking networks. A popular variant of the basic asynchronous model of the previous section is the model where it is guaranteed that any message that was sent is eventually delivered.

Say that this is the asynchronous model with non-blocking adversaries. An interesting feature of this variant is that here it is reasonable to require that protocols eventually generate outputs. That is, protocols that "do nothing" should no longer be considered secure. (This stands in contrast to the basic asynchronous model; see Remark 1 above.)

Our way of forcing protocols to generate outputs is to modify the ideal process, so that the ideal-process adversary is obliged to eventually deliver any message sent by the ideal functionality. That is, say that a ideal-process adversary is non-blocking if any message sent by the ideal functionality to some dummy party is eventually delivered. Say that a protocol securely realizes an ideal functionality $\mathcal{F}$ w.r.t. class $\mathcal{C}$ of adversaries in non-blocking networks if for any *non-blocking* adversary $\mathcal{A} \in \mathcal{C}$ there exists a *non-blocking* ideal-process adversary $\mathcal{S}$, such that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ holds for any environment $\mathcal{Z}$.

Synchronous networks. A popular and convenient abstraction of communication networks is that of *synchronous* communication. Roughly speaking, here the computation proceeds in *rounds,* where in each round each party receives all the messages that were sent to it in the previous round, and generates outgoing messages for the next round. The order of activation of parties within a round is typically 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.* (This model is often called the "rushing" model for synchronous networks.) Definition 3 can be modified as follows in order to capture synchronous networks:

1. Restrict attention to synchronous environment machines. These are environment machines that start the computation by activating each one of the parties with some input value. (The order of activation is up to the environment.) Furthermore, a synchronous environment machine does not halt until the adversary writes "`halt`" on its output tape.

2. Restrict attention to synchronous real-life adversaries. These are adversaries that deliver messages in rounds, as follows. The messages generated in the first activation of each party are called the round 0 messages. For each round number $\ell = 1, 2, ...$ the adversary activates the parties, in an arbitrary order, to deliver all the round $\ell - 1$ messages. The messages generated in these activations are called the round ℓ messages. Once there are no more messages to deliver, the adversary writes "`halt`" on its output tape.

3. Require the ideal-process adversary to be *non-blocking,* as defined in the case of non-blocking networks. As there, the purpose of this requirement is to disallow protocols that do not generate output.

That is, say that a protocol securely realizes an ideal functionality $\mathcal{F}$ in synchronous networks (w.r.t. class $\mathcal{C}$ of adversaries) if for any *synchronous* adversary $\mathcal{A} \in \mathcal{C}$ there exists a *non-blocking* ideal-process adversary $\mathcal{S}$, such that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ holds for any *synchronous* environment $\mathcal{Z}$.

Remark: Secure Function evaluation. In the case of synchronous networks it becomes feasible to realize a richer set of cryptographic tasks (namely, ideal functionalities). One prominent class of functionalities that are realizable only in synchronous networks is secure function evaluation (see for instance [GL90, MR91, B91, DM00, C00]). Here there is a "trusted party" that obtains the input values of all parties, evaluates the given function on these inputs, and then hands the designated output values simultaneously to all parties. See more details in Section 7.9.

Broadcast channel. Another abstraction that is convenient and popular when designing synchronous protocols is that all parties have access to a broadcast channel. Every message sent on the broadcast channel is received by all parties. In our formalization, a broadcast channel is modeled by dedicating a special segment of the incoming and outgoing communication tapes of the parties to broadcast messages. The real-life computation is modified by requiring the adversary to deliver any outgoing message that was written on the special segment to all uncorrupted parties, and write these messages on the special segment of their incoming communication tapes. The ideal process remains unchanged.

Unauthenticated communication. The real-life model of Section 4.1 allows the adversary to read messages sent among parties, and also to delay, reorder and delete messages. Nonetheless, the communication is "ideally authenticated", in the sense that the adversary is not allowed to duplicate, modify, or inject messages.

In contrast, in most realistic networks the communication is not guaranteed to be authentic: messages may be maliciously duplicated, modified and injected. Using our formalization, this may be captured by modifying the real-life model of computation to reflect these additional capabilities of the adversary. Specifically, modify item 1(b)i in Figure 1 so that $\mathcal{A}$ is allowed to write *any* message (specifying any sender) on the incoming communication tape of the chosen recipient, regardless of whether this message was ever generated by the sender. Following [BCK98], we call this model the unauthenticated-links model. As there, it may be useful to augment the unauthenticated-links model with an idealized initial stage where the parties are dealt some initial secret information. (This information typically represents the set-up of long-term keys for either symmetric or asymmetric cryptosystems.) Say that a protocol securely realizes an ideal functionality $\mathcal{F}$ with unauthenticated links if it satisfies Definition 3 where the real-life model is replaced by the unauthenticated-links model. (The ideal process remains unchanged.)

Remark: In light of the fact that realistic networks are typically unauthenticated, it may appear that Definition 3 should have been stated in the unauthenticated-links model, real-life model of computation, rather than in a model that postulates ideally authenticated links. Our choice to use ideally authenticated communication in the main model of computation is motivated by the fact that assuming authenticated links is a common abstraction that facilitates the design and analysis of complex multi-party protocols. Indeed, separating the part that guarantees authenticity from the rest of the protocol seems to be a useful and convenient design methodology. (This methodology is formalized and justified in [BCK98] in a context that is very similar to the one here, and also in [CHH00] in a different context. In particular, general and simple "compilers" are shown in [BCK98] that transform any protocol that was designed for the model of Section 4.1 into an "equivalent" protocol that runs in the unauthenticated-links model. See more details in Section 7.1.[5])

Ideally secure communication. Another popular and useful abstraction of communication networks is that of ideally secure communication channels. Here it is assumed that uncorrupted parties can communicate "secretly" without allowing the adversary to have access to the communicated information. We consider two variants of this abstraction:

The first variant postulates that the adversary has no access whatsoever to the communication among parties. Here a "technical" problem arises: if the adversary has no knowledge of messages

[5]The [BCK98] models of computation do not include the environment, thus the proofs there do not immediately apply to the models here. Nonetheless, it is straightforward to extend the proof there to the models here.

sent between uncorrupted parties, then how do messages get delivered and how are parties being activated. Luckily, in the case of *synchronous* networks this technicality has a simple and natural solution: It is assumed that each party sends a (possibly empty) message to each other party in each round; consequently, in each round each party is activated to receive the previous-round messages from all other parties. As usual, the order of activation of parties within a round is determined by the adversary.

The second variant guarantees a lesser degree of security of the channels, and is consequently somewhat more realistic. (In particular, it makes sense even in an asynchronous setting.) Here the adversary learns the identities of the sender and the recipient of messages sent by parties, and also a bound on the length of messages. Delivery of messages proceeds as in the real-life model of Section 4.1, with the exception that the adversary's decisions are made without access to the contents of messages sent among uncorrupted parties.

One may strengthen the security provided by either variant by considering also *computationally unbounded* real-life adversary and environment. In this case it makes sense to require that the complexity of the ideal-model adversary be polynomial in the complexity of the real-life adversary. Other standard strong variants require the two sides of (1) to be statistically indistinguishable, or even identically distributed. See more details and discussion on these issues in [C00].

The common random string (reference string) model. A popular and useful methodology for designing cryptographic protocols is to make sure that all the participants have access to a common string that is drawn from some specified distribution. (This string is chosen ahead of time and is given to each party in an "out of band" manner, before it starts any interaction with the other parties.) It is stressed that the security of the protocol depends on the fact that the common string (often called the reference string) is generated using a pre-specified randomized procedure. This in turn implies full trust in the entity that generates the reference string. In our formalization, this methodology is modeled as follows:

- The real-life model of computation is modified so that all participants have access to a common string that is chosen in advance according to some distribution (specified by the protocol run by the parties) and is written in a special location on the input tape of each party.
- The ideal model is modified as follows. In a preliminary step, the ideal-model adversary chooses a string in some arbitrary way and writes this string on the input tape of the environment machine. After this initial step the computation proceeds as before (i.e., as in Figure 2). Note that the dummy parties and the ideal functionality do not have access to the reference string.

Allowing the ideal-model adversary to choose the reference string represents the fact that security is guaranteed only if the choice of the reference string was carried out independently of the actions of the adversary and the environment. Furthermore, it is assumed that the adversary and environment do not have any additional information on the choice of the reference string. (For instance, a common distribution of reference strings is that of two random generators g, h of some large group of prime order. Here it is important that the discrete logarithm $\log_g h$ is not known.)

Non-adaptive (static) adversaries. Definition 3 postulates adaptive adversaries, namely adversaries that may corrupt parties as the computation proceeds, based on the information gathered 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].

Security against non-adaptive adversaries is captured by considering only environments and real-life adversaries where all the party corruption activities are performed before any party is activated. (Recall that in the case of non-concurrent composition and non-adaptive adversaries, treated in [C00], there was no need to include the environment machine in the model. In the case of security that is closed under concurrent composition the environment machine seems to be needed even with non-adaptive adversaries.)

Passive (eavesdropping) adversaries. Definition 3 postulates active adversaries, namely adversaries that have 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.

The definition can be adapted in a straightforward way to deal with the case of passive adversaries. Specifically:

- Modify the real-life model as follows: Even after being corrupted by the adversary (Step 1(b)ii in Figure 1), the ITM that corresponds to the corrupted party keeps being activated for receiving inputs and incoming messages, and keeps generating outputs and outgoing messages, with the exception that now the adversary has read access to the memory contents of the corrupted parties.

- Modify the ideal process as follows: Even after being corrupted by the adversary (Step 1(b)iii in Figure 2), the corrupted dummy party $\tilde{P}_i$ can keep being activated for receiving inputs from the environment, and keeps forwarding any received input to the ideal functionality. The adversary cannot send $\mathcal{F}$ messages with source identity $\tilde{P}_i$.

The rest of the definition remains unchanged.

Protection against session-wise corruption. Many popular cryptographic tasks consist of a simpler sub-task that is repeatedly carried out by different parties and with different parameters. Requests for carrying out the sub-task are generated by other protocols or external events, and are usually assumed to be arbitrary. Also, different requests may have substantially different security requirements and are subject to different security threats. Examples include key-exchange and securing session-oriented communication, as well as electronic commerce transactions, pairwise interactive proofs in a larger network, and more. Consequently, protocols that carry out these tasks often consist of a single (or a small number of) subroutines that are invoked, by the involved parties and with the appropriate parameters, whenever a request to carry out the sub-task is made. We generically refer to each invocation of such a subroutine within a party as a session. (See more details on the notion of a session in the next section.) A security concern that often arises in the case of session-oriented tasks is how to make sure that the security of each session remains intact (as much as possible), regardless of whether other sessions *run by the same party* are compromised.

In a setting where attacks on specific sessions are based only on adversarial control over the communication channels and over the invocations of sessions, this concern can be captured by formulating appropriate ideal functionalities and no further modification to Definition 3 is needed.

However, one may sometimes be concerned about adversaries that can obtain (via corruption) the internal state of some sessions more easily than obtaining the entire internal state of a party. To capture this concern, Definition 3 can be modified by allowing both the real-life adversary and the ideal-process adversary to corrupt also individual sessions (i.e., subroutine invocations), in addition to corrupting parties. All other model characteristics remain unchanged. More precisely, modify the definition as follows.

- Allow a protocol to designate some of its subroutines as corruptible. (We do not formally define how to specify subroutines within the code of a program, or an ITM. This can be done in a number of standard ways and is left to the reader.)
- Modify the real-life model by allowing the adversary an additional activity: session corruption. Here the adversary names a session within a party, and obtains the local state of this session. The environment learns that the session has been corrupted. The session has to be an invocation of a *corruptible* subroutine. We do not specify nomenclature mechanism for sessions. The specific mechanism by which the adversary refers to a specific session is left to be determined by the individual task or protocol. (In reality there is often a natural method for naming sessions, for instance session-id's that are attached to messages.)
- Modify the ideal process by allowing the adversary to perform session corruption. Here it is the ideal functionality that notifies the adversary (in some predefined manner) of the identities of the sessions that are available for corruption. The adversary names a session within a dummy party, and obtains (from the ideal functionality) some information that relates to that session. In addition, the environment learns that the session has been corrupted.

The rest of Definition 3 remains unchanged.

5 Concurrent composition of secure protocols

This section states and proves the composition theorem, as sketched in Section 2. Sections 5.1 and 5.2 define the hybrid model and state the composition theorem. Section 5.3 discusses and motivates some aspects of the hybrid model and the theorem. Section 5.4 presents the proof.

5.1 Preparing the ground

In order to be able to state the composition theorem, we first "prepare the ground", in three steps. First we make a technical restriction on the format of ideal functionalities to which the composition theorem applies. (Without this restriction it is not clear how to even state the composition theorem.) Next, we define the hybrid model where a protocol π has access to (many independent copies of) a given ideal functionality $\mathcal{F}$. Next, we define the "mechanics" of replacing π's calls to the ideal functionality with calls to some protocol ρ that securely realizes $\mathcal{F}$.

Session ID's. In order to be able to formalize the hybrid model and state the composition theorem, we make the following syntactic restriction on the ideal functionality. Each incoming message for $\mathcal{F}$ should have a special field, called the Session ID (SID) field. The contents of this field, the SID, should be the same in all incoming messages to a given invocation of $\mathcal{F}$. That is, any incoming message whose SID field is different than that of the first message in an invocation

is ignored. Furthermore, all outgoing messages generated by $\mathcal{F}$ have an SID field whose contents is identical to the SID field of the first incoming message. (We envision that the SID field of the first message is written on the ID tape of $\mathcal{F}$.)

This restriction allows us to distinguish between different copies (i.e., invocations) of the ideal functionality that are running at the same time. It also guarantees that protocols that realize these functionalities will have similar behavior: they will expect an SID in each input, and will include the SID in each output.

The hybrid model. We specify the real-life model of computation with the assistance of some ideal functionality $\mathcal{F}$. This model, called the hybrid model with ideal access to $\mathcal{F}$ (or in short the $\mathcal{F}$-hybrid model), is obtained as follows. We start with the real-life model of Section 4.1. To this model we add an unlimited number of copies of the ideal functionality $\mathcal{F}$. In addition to the activities specified in the real-life model, in each activation the parties and adversary may send messages to and receive messages from the various copies of $\mathcal{F}$. The communication between the parties and each copy of $\mathcal{F}$ mimics the ideal process. That is, messages addressed to some copy of $\mathcal{F}$ are written on a distinguished part of the sending party's outgoing communication tape. $\mathcal{F}$ reads these messages right off the sending party's tape, and can write value directly on the party's incoming communication tape. In addition, messages that $\mathcal{F}$ writes on its own outgoing communication tape are delivered by the adversary $\mathcal{H}$, without allowing $\mathcal{H}$ access to the contents of the delivered messages. (as in the ideal process, $\mathcal{H}$ may choose to deliver $\mathcal{F}$'s messages at any time during the computation, or even to not deliver these messages at all.)

Distinguishing between the different copies of $\mathcal{F}$ is done via the Session IDs. That is, whenever a party writes a message on the distinguished segment of its outgoing communication tape, the message is copied to the incoming communication tape of the copy of $\mathcal{F}$ that corresponds to the SID field of that message. If no existing copy of $\mathcal{F}$ corresponds to the SID field then a new copy of $\mathcal{F}$ is invoked and given that message. For a string $id \in \{0,1\}^*$ We let $\mathcal{F}_{(id)}$ denote the copy of $\mathcal{F}$ whose SID is id.

The order of events in a protocol execution in the hybrid model is described in Figure 3. It is stressed that the copies of $\mathcal{F}$ are separate ITMs with independent random inputs; furthermore, they do not send messages to each other. (Indeed, $\mathcal{F}$ originally operates in a stand-alone ideal process where no other copies exist.)

Let $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},\mathcal{Z}}(k,z)$ denote the random variable describing the output of environment machine $\mathcal{Z}$ on input z, after interacting in the $\mathcal{F}$-hybrid model with protocol π, adversary $\mathcal{H}$, analogously to the definition of $\text{REAL}_{\pi,\mathcal{H},\mathcal{Z}}(k,z)$ in Section 4. (We stress that here π is a hybrid of a real-life protocol with ideal evaluation calls to $\mathcal{F}$.) Let $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},Z}$ denote the distribution ensemble $\{\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},Z}\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

Replacing a call to an ideal functionality with a subroutine call. Next we describe the 'mechanics' of replacing a call, generated by a protocol π in the $\mathcal{F}$-hybrid model to some copy of the ideal functionality $\mathcal{F}$, with an activation of a message-driven protocol ρ (that presumably realizes $\mathcal{F}$). This is done in a straightforward way. That is, the description of π is modified as follows, to obtain a protocol π^ρ. (It is stressed that the following are protocol instructions, to be carried out by the uncorrupted parties.)

1. Whenever π instructs party P_i to send a message x to $\mathcal{F}_{(id)}$, protocol π^ρ instructs P_i to proceed as follows.

Protocol execution in the $\mathcal{F}$-hybrid model

Participants: Parties $P_1, ..., P_n$ running protocol π with multiple copies of an ideal functionality $\mathcal{F}$, with adversary $\mathcal{H}$, and with environment $\mathcal{Z}$ on input z. All participants have the security parameter k.

1. While $\mathcal{Z}$ has not halted do:
   (a) $\mathcal{Z}$ is activated (i.e., its activation tape is set to 1). $\mathcal{Z}$'s activity remains unchanged from the real-life model (Figure 1). In particular, $\mathcal{Z}$ cannot access any copy of $\mathcal{F}$.
   (b) Once the adversary $\mathcal{H}$ is activated with a new value input coming from $\mathcal{Z}$, it proceeds according to its program and possibly writes new information on its output tape. In addition to reading its own tapes, $\mathcal{H}$ can read the outgoing message tapes of all the parties (except for the distinguished segments, where the parties communicate with the copies of $\mathcal{F}$). Furthermore, $\mathcal{H}$ can choose to perform one of the following three additional activities:
      i. $\mathcal{H}$ can deliver a message m from party P_i to party P_j. This activity remains unchanged from the real-life model (Figure 1).
      ii. $\mathcal{H}$ can deliver a message m from some copy of $\mathcal{F}$ to party P_j. This activity remains unchanged from the ideal process (Figure 2).
      iii. $\mathcal{H}$ can corrupt a party P_i. This activity remains unchanged from the real-life model (Figure 1).
      iv. $\mathcal{H}$ may write a message, m, on the incoming communication tape of some copy $\mathcal{F}_{(id)}$ of $\mathcal{F}$. This message appears with sender $\mathcal{H}$.

      If some message was delivered to P_i (or some $\mathcal{F}_{(id)}$) in this activation then P_i (or $\mathcal{F}_{(id)}$) is activated once $\mathcal{H}$ enters either the waiting state. Otherwise, $\mathcal{Z}$ is activated.
   (c) Once an uncorrupted party P_i is activated (either due to a new input generated by $\mathcal{Z}$, or due to a message delivered by $\mathcal{H}$), it proceeds according to its program and possibly writes new information on its output tape and new outgoing messages on its outgoing message tape. In each activation, P_i may write a message to at most one copy of $\mathcal{F}$. If such message is written, then that copy of $\mathcal{F}$ is activated next. Otherwise, $\mathcal{Z}$ is activated next.
   (d) Once some copy $\mathcal{F}_{(id)}$ of $\mathcal{F}$ is activated it follows its program until it enters either the waiting state or the halt state. As in the ideal process, $\mathcal{F}_{(id)}$ has read and write access to the communication tapes of the parties, and in addition can send messages to any party. In addition, $\mathcal{F}_{(id)}$ may write information on the incoming communication tape of $\mathcal{H}$. In that case, $\mathcal{H}$ is the next party to be activated once $\mathcal{F}_{(id)}$ enters the waiting state. Otherwise, the party that was last activated before $\mathcal{F}_{(id)}$ is activated again.
   (e) Once $\mathcal{H}$ is activated due to an incoming message from some $\mathcal{F}_{(id)}$, it follows its program and possibly writes some information on the incoming communication tape of $\mathcal{F}_{(id)}$. As in the ideal process, in this type of activation $\mathcal{H}$ may not deliver messages to or corrupt parties, nor can it hand information to other copies of $\mathcal{F}$. Once $\mathcal{H}$ enters the waiting state, $\mathcal{F}_{(id)}$ is activated again.
2. The output of the execution is the (one bit) output of $\mathcal{Z}$.

Figure 3: The order of events in a protocol execution in the hybrid model

(a) If this is the first message sent to $\mathcal{F}_{(id)}$ then invoke a new copy of protocol ρ and immediately activate this copy with incoming message x and uniformly distributed string for its random input. That is, a new copy of the ith ITM in ρ is invoked as a subroutine of P_i, and is given identity (P_i, id). (See Section 3.1 for more low-level details on how a new protocol invocation runs side-by-side with a an existing invocation.) We say that the new copy of ρ, denoted $\rho_{i,(id)}$, is associated with $\mathcal{F}_{(id)}$.

(b) Otherwise, hand x to the already-existing copy $\rho_{i,(id)}$.

2. Whenever the copy $\rho_{i,(id)}$ wishes to send a message m to some party $P_{i'}$, write the message $(m, \rho, (id))$ on the outgoing communication tape. Messages that are generated by $\rho_{i,(id)}$ are called $\rho_{i,(id)}$-messages.

3. Whenever activated due to delivery of a $\rho_{i',(id)}$-message $(m, \rho, (id))$ from $P_{i'}$, check whether there is a running copy of ρ that is associated with $\mathcal{F}_{(id)}$. If such a copy, $\rho_{i,(id)}$, exists, then activate this copy with incoming message m. Otherwise, invoke a new copy of ρ and activate it with incoming message m.

4. Whenever a copy $\rho_{i,(id)}$ generates an output value y, proceed just as π proceeds with incoming message y from $\mathcal{F}_{(id)}$.

5.2 The composition theorem

We are now ready to state the composition theorem. Following [C00], this is done in two steps. First we state a more general theorem that makes roughly the following statement: Let π be an arbitrary protocol in the $\mathcal{F}$-hybrid model, and let ρ be a protocol that securely realizes $\mathcal{F}$ as in Definition 3. Then the protocol π^ρ, running in the real-life model, has "essentially the same behavior" as protocol π. That is, for any real-life adversary $\mathcal{A}$ there exists a hybrid-model adversary $\mathcal{H}$ such that no environment machine $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and π^ρ in the real-life model or with $\mathcal{H}$ and π in the $\mathcal{F}$-hybrid model. The statement of the theorem follows.

Theorem 4 (Concurrent composition: General statement) *Let $\mathcal{C}$ be a class of real-life adversaries, and let $\mathcal{F}$ be an ideal functionality. Let π be an n-party protocol in the $\mathcal{F}$-hybrid model, and let ρ be an n-party protocol that securely realizes $\mathcal{F}$ with respect to $\mathcal{C}$. Then for any real-life adversary $\mathcal{A} \in \mathcal{C}$ there exists a hybrid-model adversary $\mathcal{H} \in \mathcal{C}$ such that for any environment machine $\mathcal{Z}$ we have:*

$$\text{REAL}_{\pi^\rho,\mathcal{A},\mathcal{Z}} \approx \text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},\mathcal{Z}}. \tag{2}$$

The second step, which is a corollary from the first one, concentrates on protocols π that securely realize some given functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model. The corollary essentially states that if protocol π securely realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model and if ρ securely realizes $\mathcal{F}$, then π^ρ securely realizes $\mathcal{G}$. To formalize this corollary, we first define protocols for securely realizing a functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model. This is done via the usual comparison to the ideal process for $\mathcal{G}$:

Definition 5 *Let $\mathcal{C}$ be a class of real-life adversaries, let π be an n-party protocol, and let $\mathcal{F}, \mathcal{G}$ be ideal functionalities. We say that π* securely realizes *$\mathcal{G}$* in the *$\mathcal{F}$*-hybrid model with respect to *$\mathcal{C}$ if for any adversary $\mathcal{H} \in \mathcal{C}$ (in the $\mathcal{F}$-hybrid model) there exists an ideal-process adversary $\mathcal{S}$ such that for any environment machine $\mathcal{Z}$ we have:*

$$\text{IDEAL}_{\mathcal{G},\mathcal{S},Z} \approx \text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},Z}. \tag{3}$$

Corollary 6 (Concurrent composition: Realizing functionalities) *Let $\mathcal{C}$ be a class of real-life adversaries, and let $\mathcal{F}, \mathcal{G}$ be ideal functionalities. Let π be an n-party protocol that realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model with respect to $\mathcal{C}$, and let ρ be an n-party protocol that securely realizes $\mathcal{F}$ with respect to $\mathcal{C}$. Then protocol π^ρ securely realizes $\mathcal{G}$ with respect to $\mathcal{C}$.*

Proof: Let $\mathcal{A} \in \mathcal{C}$ be a real-life adversary that interacts with parties running π^ρ. Theorem 4 guarantees that there exists an adversary $\mathcal{H} \in \mathcal{C}$ in the $\mathcal{F}$-hybrid model such that $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},\mathcal{Z}} \approx \text{REAL}_{\pi^\rho,\mathcal{A},\mathcal{Z}}$ for any environment machine $\mathcal{Z}$. The security of π in the $\mathcal{F}$-hybrid model guarantees that there exists an ideal model adversary (a "simulator") $\mathcal{S}$ such that $\text{IDEAL}_{g,\mathcal{S},\mathcal{Z}} \approx \text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},\mathcal{Z}}$ for any $\mathcal{Z}$. The corollary follows by combining the two equalities. □

5.3 Discussion

One aspect of the hybrid model (the explicit incorporation of multiple copies of $\mathcal{F}$ in the hybrid model) was motivated in Section 2. This section highlights and motivates some additional aspects.

Handling multiple functionalities. The hybrid model is defined with respect to (multiple copies of) a *single* functionality, $\mathcal{F}$. One can easily generalize the definition to allow ideal calls to several different ideal functionalities. We chose not to do so for sake of simplicity, and since the more general formalization hardly adds any power to the model: One can always define a single functionality that mimics several different ones, say via a "universal functionality".

The role of Session IDs. The $\mathcal{F}$-hybrid model requires that each message addressed to a copy of $\mathcal{F}$ has an SID, and specifies that the messages is directed to the copy of $\mathcal{F}$ that are associated with that SID. This guarantees that at most one copy of $\mathcal{F}$ with a given SID exists. The model does not specify how the SIDs are chosen, or how several parties learn the SID of a given copy of $\mathcal{F}$. These tasks are left to the protocol in the $\mathcal{F}$-hybrid model. This convention seems to simplify the task of formulating ideal functionalities (and designing protocols that securely realize them) by freeing the functionality from the need to choose the SIDs and guarantee their uniqueness. In addition, it seems to reflect common practices of protocol design in existing networks.

Invoking several subroutines in a single activation. The hybrid model specifies that, whenever some party P_i activates a copy $\mathcal{F}_{(id)}$ of $\mathcal{F}$, then P_i is immediately re-activated once $\mathcal{F}_{(id)}$ completes its activation. This provision allows P_i to activate a number of copies of $\mathcal{F}$, one after the other, without interference of either $\mathcal{H}$ or $\mathcal{Z}$. This complicates the model somewhat; Nonetheless, we chose this formalization since it seems to better represent realistic protocols, where a number of subroutines can be activated in a single activation of the calling protocol.

Nesting of protocol invocations. Theorem 4 is stated for the case where the "inner" protocol (i.e., the protocol ρ that securely realizes $\mathcal{F}$) is a protocol in the real-life model. The theorem can be easily extended to deal also with the case where ρ runs in a hybrid model with ideal access to some functionality $\mathcal{F}'$. Here the composed protocol, π^ρ, is also a protocol in the $\mathcal{F}'$-hybrid model. This extension allows us to apply the theorem iteratively to protocols with multiple "nesting". (For instance, if π is a protocol in the $\mathcal{F}$-hybrid model, protocol ρ securely realizes $\mathcal{F}$ in the $\mathcal{F}'$-hybrid model, and protocol ρ' securely realizes $\mathcal{F}'$ in the real-life model, then we obtain a protocol $\pi^{\rho^{\rho'}}$ that runs in the real-life models and "emulates" π.)

We stress, however, that this process can be safely repeated only a constant number of times; otherwise the complexity of the adversaries guaranteed by the theorem may no longer be bounded by a polynomial in the security parameter.

Extensions of the hybrid model and the composition theorems. The hybrid model can be adapted as in Section 4.4 to capture the models of computation described there. Specifically, to obtain the adapted hybrid model that captures some extension, start with the corresponding adapted version of the real-life model and modify it as in Section 5.1 to accommodate the copies of the ideal functionality. As a result, Definition 5 (Security in the hybrid model) is extended to security in the hybrid version of the corresponding extension of the real-life model.

Furthermore, it can be verified that the composition theorems (Theorems 4 and 6) hold also in each one of the extended models. (That is, if the given real-life adversary belongs to the class of adversaries defined by some adaptation of the original real-life model, then the resulting hybrid-model adversary belongs to the class of adversaries that correspond to the adapted version of the hybrid model. Specifically, this holds for the cases of synchronous networks, broadcast channel, unauthenticated communication, ideally secure communication, non-adaptive adversaries, session-wise corruption, and the common random string model (discussed below).

Protocol composition in the common random string (CRS) model. Some words of clarification are in order with respect to the composition theorem in the CRS model. Specifically, the hybrid CRS model is defined so that each copy of the ideal functionality has its own separate instance (or, portion) of the CRS. Similarly, the composed protocol π^ρ is assumed to provide each copy of the subroutine protocol ρ with a separate portion of the CRS. (Indeed, it is easy to see that the proof of Theorem 4 fails if two copies of ρ use the same instance of the CRS.) We remark that the behavior of protocols where several copies of the protocol use the same instance of the CRS can be captured using ideal functionalities that capture all copies of the protocol within a single copy of the functionality.

5.4 Proof of the composition theorem

Section 5.4.1 contains an outline of the proof. A detailed proof appears is in Section 5.4.2.

5.4.1 Proof outline

Let $\mathcal{C}$ be a class of adversaries. (For concreteness, we consider classes that only restrict the allowable sets of parties to be corrupted.) Let $\mathcal{F}$ be an ideal functionality, let π be an n-party protocol in the $\mathcal{F}$-hybrid model, let ρ be a protocol that securely realizes f w.r.t. $\mathcal{C}$, and let π^ρ be the composed protocol. Let $\mathcal{A} \in \mathcal{C}$ be a real-life adversary, geared towards interacting with parties running π^ρ. We wish to construct an adversary $\mathcal{H} \in \mathcal{C}$ in the $\mathcal{F}$-hybrid model such that no $\mathcal{Z}$ will be able to tell whether it is interacting with π^ρ and $\mathcal{A}$ in the real-life model or with π and $\mathcal{H}$ in the $\mathcal{F}$-hybrid model. That is, for any $\mathcal{Z}$, $\mathcal{H}$ should satisfy

$$\text{REAL}_{\pi^\rho,\mathcal{A},\mathcal{Z}} \approx \text{HYB}^{\mathcal{F}}_{\pi,\mathcal{H},\mathcal{Z}}. \tag{4}$$

The general outline of the proof proceeds as follows. (Note that this outline is somewhat different than that used in [C00] for the case non-concurrent composition of protocols for secure function evaluation. This is mainly because the order of quantifiers here is different.)

1. Construct a real-life adversary, denoted $\mathcal{A}_\rho$, that interacts with some environment and parties running a single copy of protocol ρ as a stand-alone protocol.

2. The security of ρ guarantees that there exists a simulator (i.e., an ideal-process adversary), $\mathcal{S}$, such that for any environment $\mathcal{Z}_\rho$ we have:

$$\text{REAL}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho} \approx \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}_\rho}. \tag{5}$$

3. Construct out of $\mathcal{A}$ and $\mathcal{S}$ an adversary, $\mathcal{H}$, that interacts with some environment $\mathcal{Z}$ and parties running protocol π in the $\mathcal{F}$-hybrid model.

4. Demonstrate that $\mathcal{H}$ satisfies (4). This is done by reduction: Given an environment $\mathcal{Z}$ that violates (4), construct an environment $\mathcal{Z}_\rho$ that violates (5).

We sketch the above steps. Adversary $\mathcal{A}_\rho$ is very simple; in fact, it does not depend on $\mathcal{A}$ at all. Once activated, it follows the instructions written on its input tape. These instructions, written by the environment, may be either to report the new outgoing messages that were generated by the parties, or to deliver given a message to some party, or to corrupt a party and report the party's internal state. Once the instructions are fulfilled, $\mathcal{A}_\rho$ writes the gathered information on its output tape and enters the waiting state.

Adversary $\mathcal{H}$ operates as follows. In a nutshell, $\mathcal{H}$ follows the instructions of $\mathcal{A}$ with respect to the execution of π, where the interaction of $\mathcal{A}$ with the copies of ρ is handled using several copies of the simulator $\mathcal{S}$. More specifically, $\mathcal{H}$ runs a copy of $\mathcal{A}$, plus copies of the simulator $\mathcal{S}$ that correspond to the current copies of $\mathcal{F}$. In each activation, $\mathcal{H}$ first updates the state of $\mathcal{A}$ to reflect the new activity that occurred in the network. (The updates regarding protocol π are taken from the actual network that $\mathcal{H}$ interacts with. The updates regarding the copies of ρ are generated by instructing the corresponding copies of $\mathcal{S}$ to report the new outgoing messages.) Once the sate of $\mathcal{A}$ is updated, $\mathcal{H}$ runs $\mathcal{A}$ and follows its instructions regarding delivery of π-messages. The instructions of $\mathcal{A}$ regarding delivery of ρ-messages are forwarded to the corresponding copies of $\mathcal{S}$. Party corruptions are handled analogously. Whenever some copy $\mathcal{S}_{(id)}$ of $\mathcal{S}$ sends some message m to its ideal functionality, $\mathcal{H}$ forwards m to the corresponding copy, $\mathcal{F}_{(id)}$, of $\mathcal{F}$. Similarly, messages from $\mathcal{F}_{(id)}$ are forwarded to $\mathcal{S}_{(id)}$. $\mathcal{H}$ halts when $\mathcal{A}$ halts.

The validity of $\mathcal{H}$ 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 complex setting requires some work. We sketch this argument. Let m be an upper bound on the number of copies of ρ that are invoked in this interaction. Informally, we let the $\mathcal{F}^{(l)}$-hybrid model denote the model where the interaction with the first l copies of $\mathcal{F}$ works as in the $\mathcal{F}$-hybrid model, whereas the rest of the copies of $\mathcal{F}$ are replaced with an invocation of ρ. In particular, an interaction of $\mathcal{Z}$ with $\mathcal{H}$ in the $\mathcal{F}^{(m)}$-hybrid model is essentially identical to an interaction of $\mathcal{Z}$ with $\mathcal{H}$ in the $\mathcal{F}$-hybrid model; similarly, an interaction of $\mathcal{Z}$ with $\mathcal{H}$ in the $\mathcal{F}^{(0)}$-hybrid model is essentially identical to an interaction of $\mathcal{Z}$ with $\mathcal{A}$ in the real-life model of computation.

Now, assume that there exists an environment $\mathcal{Z}$ that distinguishes with probability ϵ between an interaction with $\mathcal{H}$ in the $\mathcal{F}$-hybrid model and an interaction with $\mathcal{A}$ in the real-life model. Then there is an $0 < l \leq m$ such that $\mathcal{Z}$ distinguishes between an interaction with $\mathcal{H}$ in the $\mathcal{F}^{(l)}$-hybrid model and an interaction with $\mathcal{H}$ in the $\mathcal{F}^{(l-1)}$-hybrid model. We then construct an environment $\mathcal{Z}_\rho$, that interacts with parties running a single copy of ρ, and can distinguish with probability ϵ/m between an interaction with $\mathcal{A}_\rho$ and parties running ρ in the real-life model, and an interaction with $\mathcal{S}$ in the ideal process for $\mathcal{F}$.

Essentially, $\mathcal{Z}_\rho$ runs a simulated execution of $\mathcal{Z}$, $\mathcal{H}$, and parties running π^ρ in the $\mathcal{F}^{(l)}$-hybrid model, but with the following exception. $\mathcal{Z}_\rho$ uses its actual interaction (which takes place either in the ideal process for $\mathcal{F}$ or in the real-life model) to replace the parts of the simulated execution that have to do with the interaction with the lth copy of $\mathcal{F}$, denoted $\mathcal{F}_{(id_l)}$. (That is, id_l is the SID of the lth copy of $\mathcal{F}$.) A bit more specifically, whenever some simulated party P_i sends a message x to $\mathcal{F}_{(id_l)}$, $\mathcal{Z}_\rho$ activates the corresponding actual party with input x. Outputs generated by an actual party P_i are treated like message from $\mathcal{F}_{(id_l)}$ to the simulated P_i. Whenever the simulated $\mathcal{H}$ needs to be activated, $\mathcal{Z}_\rho$ proceeds in two steps, as follows: First $\mathcal{Z}_\rho$ activates the actual adversary it interacts with and expects to obtain from it the information that $\mathcal{H}$ expects to obtain from $\mathcal{S}_{(id_l)}$, Once this information is obtained, $\mathcal{Z}_\rho$ simulates an activation of $\mathcal{H}$; if in this activation $\mathcal{H}$ instructs $\mathcal{S}_{(id_l)}$ to deliver some messages or to corrupt some P_i then $\mathcal{Z}_\rho$ hands the same instruction to the adversary it interacts with. Once the simulated $\mathcal{Z}$ halts, $\mathcal{Z}_\rho$ halts and outputs whatever $\mathcal{Z}$ outputs.

The proof is completed by demonstrating that, if $\mathcal{Z}_\rho$ interacts with $\mathcal{S}$ in the ideal process for $\mathcal{F}$, 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 $\mathcal{H}$ in the $\mathcal{F}^{(l)}$-hybrid model. Similarly, if $\mathcal{Z}_\rho$ interacts with $\mathcal{A}$ and parties running ρ in the real-life model, 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 $\mathcal{H}$ in the $\mathcal{F}^{(l-1)}$-hybrid model.

5.4.2 A detailed proof

Terminology. We use the following terminology. The internal state of an ITM M includes the contents and head locations of all the tapes readable by M and the control state. In cases where the model allows M to read tapes that are parts of other ITMs then these tapes become part of the internal state of M. Also, for convenience we include in the internal state the contents of the entire random tape of M, including locations that were not yet read by M.

The global state of a system is the concatenation of the internal states of the environment, the adversary, and all parties at some point during the protocol execution. In the hybrid model the global state contains also the internal state of all the copies of the ideal functionality. Note that exactly one ITM in a global state has its activation bit set. This ITM is the active entity in the global state.

We assume an encoding convention of internal states into strings. A string $s \in \{0,1\}^*$ is said to be an internal state of machine M if s encodes some internal state of M. (Without loss of generality we can assume that any string s encodes *some* internal state.) In the sequel we often do not distinguish between internal states and their encodings.

An execution of a protocol (either in the real-life or in the $\mathcal{F}$-hybrid model) is the process of running the protocol with a given environment and adversary on given inputs for the environment and random inputs for the all participants. Similarly, "running an activation of machine M from internal state s" means simulating an activation of M starting at the internal state described in s until M enters the halting or the waiting states. Note that s contains all the information needed for the simulation; in particular, it contains all the necessary randomness. The view of M in a given execution is the sequence of all internal states of M in that execution until it halts.

Constructions of $\mathcal{A}_\rho$, $\mathcal{H}$. Let $\mathcal{C}$ be a class of adversaries (that specifies the allowable sets of parties to be corrupted). Let $\mathcal{F}$ be an ideal functionality, let π be an n-party protocol in the $\mathcal{F}$-hybrid model, let ρ be a protocol that securely realizes f, and let π^ρ be the composed protocol. Let $\mathcal{A} \in \mathcal{C}$ be an an adversary (that interacts with parties running π^ρ). Adversary $\mathcal{A}_\rho$, sketched above, is presented in Figure 4. The security of ρ guarantees that there exists an ideal-process adversary

Adversary $\mathcal{A}_\rho$

Adversary $\mathcal{A}_\rho$ proceeds as follows, given a value k for the security parameter, and interacting with parties $P_1, ..., P_n$ running protocol ρ and with an environment $\mathcal{Z}_\rho$.

Once activated, proceed as follows:

1. If the input (written by $\mathcal{Z}_\rho$) is `'Report Status'` then copy the contents of the outgoing communication tapes of all uncorrupted parties to the output tape.
2. If the input is `'Deliver` m `to` P_j`'`, and m was either sent to P_i by some uncorrupted party and not yet delivered, or the sender of m is corrupted, then write m on the incoming communication tape of P_j. Otherwise, ignore the instruction.
3. If the input is `'Corrupt` P_j`'` and less than t parties have been corrupted so far then corrupt party P_j, and copy the internal state of P_j to the output tape. Otherwise, enter the waiting state.

Enter the waiting state.

Figure 4: The real-life adversary for a single copy of ρ.

$\mathcal{S}$ such that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}_\rho} \approx \text{REAL}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}$ holds for any environment $\mathcal{Z}_\rho$. Adversary $\mathcal{H}$ uses $\mathcal{S}$ and is presented in Figure 5.

Validity of $\mathcal{H}$. Assume that there exists an environment machine $\mathcal{Z}$ that violates the validity of $\mathcal{H}$ (that is, $\mathcal{Z}$ violates Equation (4)). We construct an environment machine $\mathcal{Z}_\rho$ that violates the validity of $\mathcal{S}$ with respect to a single run of ρ. Let us first set some conventions that will simplify the presentation and analysis of $\mathcal{Z}_\rho$. Assume that, even in the real-life model of computation, the parties have access to an ideal functionality, denoted $\mathcal{N}$, that, when activated by P_i, responds with $\perp$ and does nothing else. (I.e., $\mathcal{N}$ writes $\perp$ on P_i's incoming tape and enters the waiting state.) Clearly, this convention does not change any substantial aspect of the real-life model.

Next, define the following slight variation of protocol π^ρ. Let $\hat{\pi}^\rho$ be the protocol, running in a hybrid model, where the parties run π^ρ with the following exception. Whenever π^ρ instructs some party P_i to activate a copy $\rho_{i,(id)}$ of ρ with input x, $\hat{\pi}^\rho$ instructs P_i to first activate the corresponding copy of the ideal functionality (i.e., the copy whose SID is id) with input (x, id). If the ideal functionality responds with $\perp$, then P_i proceeds to activate $\rho_{i,(id)}$ as in π^ρ. Otherwise, $\hat{\pi}^\rho$ proceeds as protocol π would with access to the ideal functionality $\mathcal{F}_{(id)}$.

Let $m = m(k)$ be an upper bound on the number of invocations of ρ within π, given security parameter k. (The bound m is used in the analysis only. The parties need not be aware of m.) Let the $\mathcal{F}^{(l)}$-hybrid model denote the $\mathcal{F}$-hybrid model where all the copies of $\mathcal{F}$, except for the first l copies to be invoked, are replaced with copies of the null functionality $\mathcal{N}$. That is, in the $\mathcal{F}^{(l)}$-hybrid model all the calls to the copies of $\mathcal{F}$ up to the lth copy are answered as in the $\mathcal{F}$-hybrid model; the calls to the remaining copies of $\mathcal{F}$ are answered with $\perp$ (thus forcing $\hat{\pi}^\rho$ to run copies of ρ instead). Then, the $\mathcal{F}^{(m)}$-hybrid model is the same as the $\mathcal{F}$-hybrid model. Similarly, the $\mathcal{F}^{(0)}$-hybrid mode is the same as the real-life model of computation.

Notice that, when $\mathcal{H}$ interacts with parties running $\hat{\pi}^\rho$ in the $\mathcal{F}^{(l)}$-hybrid model, $\mathcal{H}$ invokes only l copies of the simulator $\mathcal{S}$. (These are the copies that correspond to the first l copies of the ideal functionality $\mathcal{F}$.) The rest of the invocations of the ideal functionality are answered with $\perp$ and

Adversary $\mathcal{H}$

Adversary $\mathcal{H}$ proceeds as follows, given a value k for the security parameter, and interacting with parties $P_1, ..., P_n$ running protocol π in the $\mathcal{F}$-hybrid model, with an environment $\mathcal{Z}$, and with multiple copies of an ideal functionality $\mathcal{F}$.

When activated for the first time, initialize a simulated interaction of adversary $\mathcal{A}$ with parties $P'_1, ..., P'_n$ running π^ρ and some environment. Save the initial internal state of $\mathcal{A}$ in variable a and proceed to Step 1. In each activation other than the first start in Step 1.

1. Update the state in a:
   (a) If there is a new input value v (written by $\mathcal{Z}$) on the input tape, then update a by writing v on the input tape of the simulated $\mathcal{A}$.
   (b) Inspect the outgoing communication tapes of $P_1, ..., P_n$. For any π-message m, generated by P_i, update a to include m in the outgoing communication tape of P'_i.
   (c) For each new incoming message m from some copy $\mathcal{F}_{(id)}$ of $\mathcal{F}$ do:
      i. If there is no copy $\mathcal{S}_{(id)}$ of the ideal-model adversary $\mathcal{S}$ that is associated with $\mathcal{F}_{(id)}$ then invoke such a copy. Let $P_1^{(id)}, ..., P_n^{(id)}$ and $\mathcal{F}'_{(id)}$ denote the parties and ideal functionality that $\mathcal{S}_{(id)}$ expects to interact with.
      ii. Write m on the incoming communication tape of $\mathcal{S}_{(id)}$ as a message from $\mathcal{F}'_{(id)}$.
   (d) Activate each one of the existing copies $\mathcal{S}_{(id)}$ of $\mathcal{S}$ with input `'Report Status'`. When $\mathcal{S}_{(id)}$ completes its activation, interpret $\mathcal{S}_{(id)}$'s output as the new $\rho_{i,(id)}$-messages generated by each uncorrupted party P_i, and update a accordingly.[a]
2. Run $\mathcal{A}$ from the updated internal state a, until $\mathcal{A}$ enters the waiting state. Furthermore:
   (a) If the simulated $\mathcal{A}$ delivers a message m to some party P'_j then do:
      i. If m is a $\rho_{i,(id)}$-message m from some sender P'_i, then activate $\mathcal{S}_{(id)}$ with input `'Deliver` m `to` P_j`'`. If in this activation $\mathcal{S}_{(id)}$ delivers a message m' from $\mathcal{F}'_{(id)}$ to $P_i^{(id)}$ then deliver the message m' from $\mathcal{F}_{(id)}$ to the actual party P_i.[a]
      ii. Otherwise (i.e., m is a π-message), deliver m to the actual party P_j.
   (b) If the simulated $\mathcal{A}$ corrupts party P'_j then do:
      i. Corrupt party P_j and obtain the internal state of P_j with respect to protocol π in the $\mathcal{F}$-hybrid model.
      ii. Activate each one of the existing copies $\mathcal{S}_{(id)}$ of $\mathcal{S}$ with input `'Corrupt` $P_j^{(id)}$`'`. When $\mathcal{S}_{(id)}$ corrupts $P_j^{(id)}$, play $\mathcal{F}'_{(id)}$ for $\mathcal{S}_{(id)}$ by providing it with the sequence of inputs and outputs of the copy $\rho_{j,(id)}$ within P_j.[a]
      iii. Interpret the output of each $\mathcal{S}_{(id)}$ as the state of P'_j with respect to $\rho_{j,(id)}$. Combine this information with the state of P_j for protocol π to obtain the state of P'_j in protocol π^ρ. Update a accordingly and continue running $\mathcal{A}$.
   (c) If the simulated $\mathcal{A}$ writes value v on its output tape then write v on the output tape.

[a] Also, if during its activation $\mathcal{S}_{(id)}$ sends a message m to $\mathcal{F}'_{(id)}$ then hand m to $\mathcal{F}_{(id)}$. Once $\mathcal{F}_{(id)}$ completes its activation, forward any new messages from $\mathcal{F}_{(id)}$ to $\mathcal{S}_{(id)}$ and resume the execution of $\mathcal{S}_{(id)}$.

Figure 5: The adversary for protocol π in the $\mathcal{F}$-hybrid model.

Environment $\mathcal{Z}_\rho$

Environment $\mathcal{Z}_\rho$ proceeds as follows, given a value k for the security parameter, input z_ρ, and interacting with parties $P_1, ..., P_n$ running protocol ρ and with an adversary $\mathcal{A}_\rho$.

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 $l \in \mathbf{N}$. Let variable s hold the initial global state of a system of parties $P'_1, ..., P'_n$ running protocol π^ρ in the $\mathcal{F}^{(l)}$-hybrid model, with adversary $\mathcal{H}$ and environment $\mathcal{Z}$ on input z. (The active entity in state s is the environment $\mathcal{Z}$.) Proceed to Step 2.
In each activation other than the first, start in Step 1.

1. Update the state s:
   (a) If the active entity in s is party P'_i and the output tape of P_i contains a new output value y then update the internal state of the simulated P'_i by including a message y received from the lth copy of $\mathcal{F}$, denoted $\mathcal{F}_{(id_l)}$.
   (b) If the active entity in s is $\mathcal{H}$ and the output of $\mathcal{A}_\rho$ includes new information regarding the state of $\mathcal{H}$ then update s to include this information. (As seen below, this information will be either new outgoing $\rho_{i,(id_l)}$-messages of some party P_i, or the internal state of some newly corrupted party.)
2. Simulate an execution of the system in the $\mathcal{F}^{(l)}$-hybrid model, from state s, until one of the following events occurs:
   (a) Party P'_i sends a message x to $\mathcal{F}_{(id_l)}$. In this case, save the current state of $\mathcal{Z}$ in s, write x on the input tape of P_i, and enter the waiting state. *(P_i will be activated next. In the next activation of $\mathcal{Z}_\rho$, which will occur immediately after the activation of P_i is complete, P'_i will be the active entity in s and Instruction 1a will be carried out.)*
   (b) The simulated adversary $\mathcal{H}$ is activated. In this case, save the current state of $\mathcal{Z}$ in variable s, write a special input value `'Report Status'` on the input tape of $\mathcal{A}_\rho$, and enter the waiting state. *($\mathcal{A}_\rho$ will be activated next. In the next activation of $\mathcal{Z}_\rho$, $\mathcal{H}$ will be the active entity in s, and Instruction 1b will be carried out.)*
   (c) The simulated $\mathcal{H}$ instructs $\mathcal{S}_{(id_l)}$ to deliver a $\rho_{i,(id_l)}$-message m from $P_i^{(id)}$ to $P_j^{(id)}$. In this case write `'Deliver` m `to` P_j`'` on A_ρ's input tape, and enter the waiting state. *(In the next activation of $\mathcal{Z}_\rho$, P'_j will be the active entity in s and Instruction 1a will be carried out.)*
   (d) The simulated $\mathcal{H}$ instructs $\mathcal{S}_l$ to corrupt party $P_j^{(id_l)}$. In this case write `'corrupt` P_j`'` on A_ρ's input tape and enter the waiting state. *(In the next activation of $\mathcal{Z}_\rho$, $\mathcal{H}$ will be the active entity in s, and Instruction 1b will be carried out.)*
   (e) Environment $\mathcal{Z}$ halts. In this case, output whatever $\mathcal{Z}$ outputs and halt.

Figure 6: The environment for a single copy of ρ.

protocol ρ is invoked instead. In particular, in the $\mathcal{F}^{(m)}$-hybrid model an execution of protocol $\hat{\pi}^\rho$ with adversary $\mathcal{H}$ and environment $\mathcal{Z}$ is identical to an execution of protocol π with $\mathcal{H}$ and $\mathcal{Z}$. Similarly, in the $\mathcal{F}^{(0)}$-hybrid model, an execution of protocol $\hat{\pi}^\rho$ with $\mathcal{H}$ and $\mathcal{Z}$ is in essence identical to an execution of protocol π^ρ with adversary $\mathcal{A}$ and environment $\mathcal{Z}$. (This holds since, in the $\mathcal{F}^{(0)}$-hybrid model, $\mathcal{H}$ will not invoke any copy of $\mathcal{S}$ and will end up running the code of $\mathcal{A}$

without any deviations.) Consequently, Equation (4) can be rewritten as:

$$\text{HYB}^{\mathcal{F}^{(0)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}} \approx \text{HYB}^{\mathcal{F}^{(m)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}. \tag{6}$$

Environment $\mathcal{Z}_\rho$ is described in Figure 6. The rest of the proof analyzes the performance of $\mathcal{Z}_\rho$. Let $k \in \mathbf{N}$ be some value of the security parameter, and let $z \in \{0,1\}^*$ be some input value for $\mathcal{Z}$. Assume that, given (k,z), $\mathcal{Z}$ distinguishes with probability ϵ between an interaction in the $\mathcal{F}^{(m)}$-hybrid model and an interaction in the $\mathcal{F}^{(0)}$-hybrid model. (Both interactions are with $\mathcal{H}$ and parties running $\hat{\pi}^\rho$.) That is, assume that:

$$|\text{HYB}^{\mathcal{F}^{(0)}}_{\hat{\pi}^\rho,\mathcal{A},\mathcal{Z}}(k,z) - \text{HYB}^{\mathcal{F}^{(m)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}(k,z)| \geq \epsilon. \tag{7}$$

Consequently, there exists a value $l \in \{1,...,m\}$ such that

$$|\text{HYB}^{\mathcal{F}^{(l)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}(k,z) - \text{HYB}^{\mathcal{F}^{(l-1)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}(k,z)| \geq \epsilon/m. \tag{8}$$

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

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}_\rho}(k,(z,l)) = \text{HYB}^{\mathcal{F}^{(l)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}(k,z) \tag{9}$$

and

$$\text{REAL}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}(k,(z,l)) = \text{HYB}^{\mathcal{F}^{(l-1)}}_{\hat{\pi}^\rho,\mathcal{H},\mathcal{Z}}(k,z). \tag{10}$$

Equations (9) and (10) follow from inspecting the code of $\mathcal{Z}_\rho$ and $\mathcal{H}$. In particular, if $\mathcal{Z}_\rho$ interacts with the ideal process for $\mathcal{F}$ then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ is distributed identically to the view of $\mathcal{Z}$ when interacting with $\hat{\pi}^\rho$ and $\mathcal{H}$ in the $\mathcal{F}^{(l)}$-hybrid model. Similarly, if $\mathcal{Z}_\rho$ interacts with parties running ρ in the real-life model then the view of the simulated $\mathcal{Z}$ within $\mathcal{Z}_\rho$ is distributed identically to the view of $\mathcal{Z}$ when interacting with $\hat{\pi}^\rho$ and $\mathcal{H}$ in the $\mathcal{F}^{(l-1)}$-hybrid model.

From Equations (8), (9) and (10) it follows that:

$$|\text{REAL}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}(k,(z,l)) - \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}_\rho}(k,(z,l))| \geq \epsilon/m. \tag{11}$$

This contradicts the validity of $\mathcal{S}$, thus completing the analysis of $\mathcal{Z}_\rho$ and the proof of Theorem 4.

6 Realizing general ideal functionalities

This section argues that realizing ideal functionalities in the present formalization is, in general, feasible. That is, we claim (with only a sketch of proof) that a large class of ideal functionalities can be securely realized in "any standard setting", using known techniques, as long as the protocol involves a set of parties such that only a certain fraction of these parties are corrupted.

As pointed out the remark following Definition 3, in models where message delivery is not guaranteed (and the ideal-process adversary is not obliged to deliver the messages sent by the ideal functionality), a protocol that generates no outputs securely realizes any ideal functionality. Consequently, a general feasibility theorem is of no interest in such cases. In contrast, a general feasibility theorem is indeed of interest in settings where message delivery is guaranteed, and protocols are required to generate output. Two important such settings are the cases of *synchronous* networks and *non-blocking, asynchronous* networks (see Section 4.4). The rest of this section concentrates on synchronous networks. Similar results hold with respect to non-blocking asynchronous networks.

More specifically, assume that there exists a set A of parties such that less than $|A|/3$ of the parties in A may be corrupted by the adversary throughout the computation. Then we claim that any ideal functionality, that does not write information directly on the input tapes of uncorrupted parties, can be securely realized using standard techniques. A crucial ingredient in the proposed protocol is to have the protocol include "assistant programs" to be run by the parties in A. These programs, or interactive Turing machines, will not have local inputs or outputs; instead, they will be invoked by incoming messages from other parties, and will generate messages to other parties. (Note that the formalization of Sections 3 and 4 allows for such protocols.) Specifically, we use the construction of [BGW88], with the simplification of [GRR98], and in addition encrypt each message using an appropriate encryption protocol.

The above discussion is formalized in the following theorem. Recall that an ideal functionality is immediate if it contains instructions to write information directly on the input tape of some uncorrupted party (see Section 4.2). Say that a real-life adversary $\mathcal{A}$ is t-limited if it never corrupts more than t parties in an interaction.

Theorem 7 *Let $n \in \mathbf{N}$ and let $t < n/3$. Let $\mathcal{F}$ be an ideal functionality that is not immediate. Then, under the assumption that trapdoor permutations exist, there exists a protocol that t-securely realizes $\mathcal{F}$ in a synchronous setting.*

Remarks: **1.** Variants of the above theorem hold in most other standard models of computation, including those described in Section 4.4. For instance, the theorem holds even in the case of unauthenticated networks (with an initialization phase), using the *authenticators* of [BCK98]. In the case of ideally secure communication, encrypting messages (and the assumption that trapdoor permutations exist) is not necessary. In the case of synchronous networks with a broadcast channel, $t < n/2$ is sufficient using the techniques of [RB89, CDDHR99]. In the case of non-blocking asynchronous networks a similar result holds using the techniques of [BCG93, BKR94].

2. Recall that *immediate* ideal functionalities can be used to model cases where parties hand inputs to the functionality and receive outputs *within the same activation.* We do not make any claims regarding such ideal functionalities. Indeed, while some immediate ideal functionalities can be realized (see Section 7.4), others cannot. For instance, the *Random Oracle Model* (see, e.g., [BR93A, CGH98]) can be cast in our framework as a hybrid model with access to an immediate ideal functionality $\mathcal{F}_{\text{RO}}$.[6]

Proof (sketch): We provide a very high level sketch of proof of Theorem 7. Let $t < n/3$ and let $\mathcal{F}$ be an ideal functionality that is not immediate. The first step in constructing a protocol that t-securely realizes $\mathcal{F}$ is to represent $\mathcal{F}$ via a circuit-family $C_{\mathcal{F}}$ as described in Remark 2, Section 4.2. That is, the kth circuit in the family $C_{\mathcal{F}}$ describes an activation of $\mathcal{F}$ with security parameter k. The inputs to the circuit represent the initial internal state of $\mathcal{F}$ in this activation plus the contents of the incoming message from some party. The outputs of the circuit represent the final internal state of $\mathcal{F}$ in this activation, plus the outgoing messages. Specifically, we use arithmetic circuits over Z_p^* where $p > n$ is a prime and the operations are modular addition and multiplication.

Sketch of the protocol. The protocol for realizing $\mathcal{F}$ proceeds as follows, given security parameter k. Whenever a party P_i is activated with a new input value x, it sends a request to the parties

[6] When receiving a value x from P_i, $\mathcal{F}_{\text{RO}}$ will proceed as follows. If x was never before given to $\mathcal{F}_{\text{RO}}$, then $\mathcal{F}_{\text{RO}}$ sets a variable y to a random value from an appropriate domain. If x was previously given to $\mathcal{F}_{\text{RO}}$ (by any party) then set y to the value previously answered for that x. Finally, write y on the incoming communication tape of P_i.

in A to securely evaluate the kth circuit in $C_{\mathcal{F}}$. The evaluation is done in a gate-by-gate manner, as in [BGW88, GRR98]. The values associated with the input lines to the circuit are determined as follows. Party P_i provides the parties in A with the values associated with the lines that represent the message received from P_i. (This is done using a Verifiable Secret Sharing protocol, as there.) The other input lines to the circuit represent the initial state of $\mathcal{F}$ in this activation. This state equals the final state of $\mathcal{F}$ in the previous activation; thus the parties in A already have the values associated with these lines. (In the first activation, the values of these lines are fixed and known, and the evaluated circuit is degenerated accordingly.)

Once the output lines of the circuit are evaluated, the parties in A proceed as follows. The values associated with each line that represents a message to be sent to party P_j are revealed to P_j. Values addressed to the adversary are revealed to all parties. The values associated with the output lines that represent the state of $\mathcal{F}$ at the end of the activation are kept by the parties in A, to be used in the next evaluation of the circuit. Whenever P_j receives the values associated with an output line assigned to itself, it reconstructs the corresponding output value and writes it on its output tape.

In oder to hide which party receives input at which round, the parties can provide (possibly null) input to $C_{\mathcal{F}}$ at each round. Similarly, $C_{\mathcal{F}}$ is modified so that if the input is null then the activation is aborted. In order to hide which party generates output at which round, $C_{\mathcal{F}}$ can be modified so that each party receives a (possibly null) output at each round. Only non-null values will be copied to the party's output tape. Activations of $\mathcal{F}$ with messages coming from the adversary can be realized by modifying $C_{\mathcal{F}}$ to treat input values (coming from any party) with some special format as messages coming from the adversary.

Ordering the activations. One important missing issue, among the many missing in the above sketch, is how the parties "synchronize" the various evaluations of $\mathcal{F}$. That is, the parties in A need to agree on an ordering of the activations of the circuit. This is done as follows: Evaluations of the circuit are carried out sequentially, one by one. At the end of each evaluation, the parties in A run a protocol for agreement on which evaluation request to fulfill next. This request should be one of those that were made in the earliest communication round and is not yet fulfilled. (There is no required order among the requests made in the same round.)

Adding encryption. In the ideally-secure communication model the above construction suffices. In our model, where the adversary sees all messages sent by the parties, we encrypt each message using an appropriate encryption protocol: In the case of non-adaptive adversaries, semantically secure encryption [GM84] is sufficient (if a different set of keys is used for each pair or parties). In the case of adaptive adversaries, if one trusts data erasures then simple extensions of semantically secure encryption suffice [BH92]. If data erasures are not trusted then non-committing encryption [CFGN96, B97, DN00] can be used.[7]

Sketch of analysis. Let $\mathcal{F}$ be a non-immediate functionality. Essentially, the proof that the above construction securely realizes $\mathcal{F}$ is a natural extension of the proof that the [BGW88, GRR98] construction securely evaluates any given function, according to known definitions of secure function evaluation. Unfortunately, such a proof does not appear in the literature; nonetheless, the general structure of such a proof (and many of the details) are very similar to the proof of the main theorem

[7] We remark that the known protocols for non-committing encryption based on general complexity assumption [CFGN96, DN00] use *oblivious transfer* in an essential way. As discussed in Section 7.6, we do not know if oblivious transfer is possible in our setting without some set-up assumptions. However, these works present also non-committing encryption protocols that are secure in our setting based on the RSA assumption or alternatively on the Diffie-Hellman assumption.

in [CKOR00]. There are three main differences between our setting and the setting considered in [CKOR00] (secure function evaluation in the *secure channels* setting of [C00]). Let us highlight these differences:

1. Here the parties evaluate the circuit many times in a single execution of the protocol, where the circuit evaluations are executed sequentially, and the inputs to an evaluation may depend on the outputs of previous evaluations. This can be dealt with in a straightforward way by running the ideal-process adversary (i.e., the simulator) of [CKOR00] on each of the circuit evaluations, where at the beginning of each evaluation the simulator is given the current internal state of the adversary.

2. The model here includes the environment machine, which interacts with the parties and the adversary (both in the real-life model and in the ideal process). However, we claim that the simulator constructed in [CKOR00] can be modified in a straightforward way to accommodate the addition of the environment machine. Recall that the [CKOR00] simulator $\mathcal{S}$ (like many other simulators in the literature) proceeds via "one pass black-box simulation" of the real-life adversary $\mathcal{A}$. That is, $\mathcal{S}$ runs $\mathcal{A}$ on a simulated interaction with some network, and interprets the instructions of $\mathcal{A}$ in some ways. When $\mathcal{A}$ generates output, $\mathcal{S}$ generates the same output. Such a simulation technique is readily amenable to adding the environment: whenever the simulated $\mathcal{A}$ sends a message to the environment, $\mathcal{S}$ (treats this message as an output of $\mathcal{A}$ and) forwards the same message to its environment. Whenever $\mathcal{S}$ receives an input value from the environment, it forwards this value to $\mathcal{A}$. Other than that, $\mathcal{S}$ proceeds as in [CKOR00].

   We stress that it is crucial that $\mathcal{S}$ never "rewinds" $\mathcal{A}$. As soon as $\mathcal{S}$ rewinds $\mathcal{A}$, the interaction between $\mathcal{S}$ and the environment may be distinguishable from the interaction between $\mathcal{A}$ and the environment.[8]

3. Here the secrecy of the communication is protected via encryption, whereas in the [CKOR00] model the communication is ideally secure. This difference can be dealt with as follows, using the composition theorem. First cast the secure-channels model as a standard (open channels) hybrid model with access to the "secret message transmission" ideal functionality, that receives a message from the sender and privately sends it to the receiver.[9] Next cast the encryption protocol in use as a protocol for securely realizing the "secret message transmission" functionality. Finally, compose the two protocols to obtain a protocol that securely realizes $\mathcal{F}$ in the (open channels) synchronous model.

Remark: We stress that the above construction and analysis do not work if the ideal functionality $\mathcal{F}$ is immediate. In particular, if $\mathcal{F}$ is immediate then the environment may activate some party with input value, and then expect the party to generate output *in the same activation.* This cannot be realized using the above construction. □

[8] Indeed, we chose the construction of [BGW88, GRR98] mainly because it can be proven secure via one-pass black-box simulation. We do not know whether other general constructions for secure function evaluation, such as that of [GMW87, G98] (whose only known proof of security does not proceed via one-pass, black-box simulation), are secure in the present framework.

[9] More specifically, this functionality proceeds as follow. Whenever it receives a message (m, P_j) from party P_i, it sends the message (m, P_i) to P_j, notifies the adversary that P_i sent a message to P_j (and possibly also the length of that message), and halts. A similar formalization, cast as an instance of secure function evaluation, was used in [CFGN96]. An alternative way of formalizing secret communication is presented in Section 7.3.

7 Some ideal functionalities

This section demonstrates the general applicability of the framework by using it to provide alternative definitions of a number of known cryptographic tasks. To do that, we formulate ideal functionalities that intend to capture the security requirements of these tasks; a protocol is said to securely carry out a task if it securely realizes the corresponding ideal functionality within the present framework. Recall that an ideal functionality interacts in the ideal process with the environment (via the *dummy parties*) and with the adversary. See more details in Sections 2.1 and 4.2.

We first formulate ideal functionalities that represent some more basic tasks of cryptographic protocols, namely Message Authentication, Key-Exchange, Encryption (or, rather, secret communication) in both the public-key and shared-key models, and Digital Signatures. We then proceed to formulate functionalities representing two-party primitives such as Commitment, Zero-Knowledge, and Oblivious Transfer. These primitives are treated as two-party protocols in a multi-party setting. Finally, we formulate ideal functionalities that capture Verifiable Secret Sharing and Secure Function Evaluation in *synchronous* networks.

Each one of the ideal functionalities presented below corresponds to only a single invocation 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 copies are running concurrently, by different subsets of the parties, and in an adversarially scheduled way. Furthermore, these protocols are guaranteed to "faithfully mimic" the corresponding functionality for *any application* that makes use of that functionality.

All the ideal functionalities in this section, with the exception of the one capturing the security requirements of digital signatures, are non-immediate. It thus follows from Section 6 that these ideal functionalities can be realized in settings where "assistant parties" are available. In addition, we briefly discuss the feasibility of realizing these ideal functionalities by more efficient protocols (and in particular by two-party protocols, in the relevant cases). In the case of secret communication and digital signatures we demonstrate how protocols that securely realize the proposed ideal functionalities can be constructed from schemes that satisfy known security requirements.

7.1 Authenticated Communication

Throughout most of this work we assume ideally authenticated communication links. However, as pointed out in Section 4.4 (see the paragraphs on unauthenticated communication), most existing communication networks are best represented by a model where the adversary may modify or inject messages on the communication links, and in particular "impersonate" uncorrupted parties. We call this model (which was described there) the unauthenticated-links model (UM).

As mentioned there, general "compilers" were developed (in [BCK98] in a setting similar to ours and in [CHH00] in a different setting) that transform any protocol that works in the authenticated-link model (AM) into a protocol with essentially the same functionality in the UM. We provide the following alternative specification for these compilers (called authenticators). In the present framework, the AM can be cast as a *hybrid* UM with ideal access to a functionality $\mathcal{F}_{\text{AUTH}}$ that provides ideally authenticated communication. Applying an authenticator to a protocol π (that was designed for the AM) now amounts to composing π with a protocol ρ that securely realizes $\mathcal{F}_{\text{AUTH}}$.

Figure 7 below presents functionality $\mathcal{F}_{\text{AUTH}}$. This functionality represents authentication proto-

cols that authenticate each message separately and independently of all other messages. Authentication of messages in *sessions* can be represented by a somewhat different functionality (formulated along the lines of functionality $\mathcal{F}_{\text{SCH}}$ in Figure 10 but without the secrecy guarantee).

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

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

1. Upon receiving a message $(\texttt{send}, id, P_j, m)$ from party P_i, send (id, P_i, P_j, m) to P_j and to the adversary, and halt.

Figure 7: The Message Authentication functionality, $\mathcal{F}_{\text{AUTH}}$

Claim 8 *The encryption-based and the signature-based* MT*-authenticators of* [BCK98] *securely realize functionality $\mathcal{F}_{\text{AUTH}}$ in the* UM.

Proof: The proof is a simple extension of the proofs in [BCK98]. (The main difference is that here one has to incorporate the environment.) We leave it as an exercise. □

7.2 Key Exchange

Key Exchange (KE) is a functionality where two parties interact in order to generate a common random value (a key) that remains unknown to everyone except the two parties. Typically, the key is then used as a "session key" for symmetric encryption and authentication of messages. A prominent setting for studying KE protocols is an asynchronous multi-party network with *unauthenticated* communication channels, and where many copies of the protocol (i.e., many sessions) between different pairs of parties are carried out concurrently, in an adversarially controlled way. A crucial concern in this setting is to verify (or, authenticate) the identity of the participant for an exchange. KE is one of the most widely used cryptographic primitives in today's networks; indeed, there exists a large body of work on the security requirements from KE protocols. Here let us mention only [BAN90, DOW92, BR93, BCK98, Sh99]. See more details there.

In the present framework, a KE protocol is a protocol that securely realizes functionality $\mathcal{F}_{\text{KE}}$, presented in Figure 8. The functionality provides pairs of parties with a randomly chosen shared key. It also captures the fact that there is no requirement on the distribution of keys in sessions where one of the participants is corrupted. (This is done by allowing parties to fix the generated key to any desired value, via the α field in the `(exchange...)` message. An uncorrupted user is expected to set $\alpha = \perp$.)

As usual, functionality $\mathcal{F}_{\text{KE}}$ is defined for a single invocation of the protocol (i.e., for a single session). The composition theorem guarantees that security is maintained when many sessions are running concurrently. Furthermore, it guarantees that the generated key is as secure as an "ideally random key" for any application protocol that uses these keys. In contrast, other definitions of KE incorporate multiple sessions of the protocol *in the actual definition,* and do not rigorously argue on the sufficiency of the definition for guaranteeing the security of protocols that use the shared key. (Exceptions are the work of Goldreich and Lindell [GL00] that defines KE protocols for a single session and then shows secure *non-concurrent* composition, and the work of Canetti and Krawczyk [CK00] that demonstrate the security of the generated key for symmetric encryption and authentication.)

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

$\mathcal{F}_{\text{KE}}$ proceeds as follows, running on security parameter k, with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Wait to receive a value $(\texttt{exchange}, id, P_i, P_j, \alpha)$ from some party P_i.
2. Wait to receive a value $(\texttt{exchange}, id, P_i, P_j, \alpha')$ from P_j. Then, do:
   (a) If $\alpha = \alpha' = \perp$ then choose $\kappa \stackrel{R}{\leftarrow} \{0,1\}^k$. Else, if $\alpha \neq \perp$ then set $\kappa = \alpha$; else set $\kappa = \alpha'$.
   (b) Send $(\texttt{key}, id, \kappa)$ to P_i and P_j, send $(\texttt{key}, id, P_i, P_j)$ to the adversary, and halt.

Figure 8: The Key Exchange functionality, $\mathcal{F}_{\text{KE}}$

On the feasibility of 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, BPRR95, CK00]. Nonetheless, several natural and widely used protocols securely realize $\mathcal{F}_{\text{KE}}$. We omit further details from this write-up.

7.3 Secret Communication (Encryption)

Secret communication is an abstraction of the functionality that is usually provided by encryption. There are two main classes of protocols for providing secret communication, which provide somewhat different functionalities. The first is *asymmetric* (or, public-key) encryption, where there is a specified receiver and the functionality is that all parties should be able to send messages to the receiver without allowing the adversary to learn the contents of these messages, nor to send related messages. The second class is that of *symmetric* (or, shared-key) encryption, where two parties wish to set up a "secure channel" where they can send secret messages to each other. This functionality is typically combined with the requirement that messages sent on the "secure channel" be authenticated. (This requirement is of course moot when the communication is assumed to be ideally authenticated to begin with.)

Here we mostly concentrate on modeling asymmetric encryption, and only remark on the symmetric case at the end of this section. We first propose (in Figure 9) an ideal functionality, $\mathcal{F}_{\text{ENC}}$, that captures the security requirements of asymmetric encryption. Next, after discussing some variants of this functionality, we concentrate on the relationship between protocols that securely realize $\mathcal{F}_{\text{ENC}}$ and the notion of resilience to adaptive chosen ciphertext attacks (or, CCA-security) of encryption schemes [DDN91, RS91, BDPR98]. Specifically we show that, while CCA-security is by itself not sufficient for securely realizing $\mathcal{F}_{\text{ENC}}$, a simple additional security measure allows transforming any CCA-secure encryption scheme into a protocol that securely realizes $\mathcal{F}_{\text{ENC}}$ with respect to *non-adaptive* adversaries. Securely realizing $\mathcal{F}_{\text{ENC}}$ with respect to adaptive adversaries is left as an interesting open problem. (We also note that CCA-security is not *necessary* for securely realizing $\mathcal{F}_{\text{ENC}}$; there may be protocols that securely realize $\mathcal{F}_{\text{ENC}}$ and do not have a CCA-secure counterpart.)

On secret transmission of a single message. Functionality $\mathcal{F}_{\text{ENC}}$ allows many messages to be sent to the receiver within a single session (i.e., in a single invocation of $\mathcal{F}_{\text{ENC}}$). Furthermore, messages may be sent by all parties (some of which may be corrupted). This is different from the "secret message transmission" functionality of Section 6 (see Footnote 10): There, each copy of the functionality

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

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

1. In the first activation, expect to receive a value $(\texttt{receiver}, id)$ from some party P_i. Then, send $(\texttt{receiver}, id, P_i)$ to all parties and the adversary. From now on, ignore all $(\texttt{receiver}, id)$ values.
2. Upon receiving a value $(\texttt{send}, id, m)$ from some party P_j, send (id, P_j, m) to P_i and (id, P_j) to the adversary.

Figure 9: The Secret Communication functionality, $\mathcal{F}_{\text{ENC}}$

allows sending only a single message. While the composition theorem guarantees that functionality of Section 6 suffices in principle for guaranteeing general secret communication, $\mathcal{F}_{\text{ENC}}$ better captures the functionality expected from public-key cryptosystems. Indeed, public-key cryptosystems allow parties to encrypt many messages using a single public key. In contrast, when using the "secret message transmission" functionality of Section 6 to implement secret communication a different copy of the functionality is needed for each message. Consequently, when replacing this functionality with an encryption protocol π that securely realizes it, one has to invoke a new copy of π for each message. If π is based on public-key encryption then this in particular implies that new public and private keys need to be generated for each message.

On hiding the length and traffic analysis. Whenever some party P_j sends a message to the receiver, $\mathcal{F}_{\text{ENC}}$ notifies the adversary that P_j sent a message to the receiver. This reflects the common view that encryption does not hide the fact that a message was sent. (Using common terminology, there is not protection against traffic analysis.) On the other hand, $\mathcal{F}_{\text{ENC}}$ hides all information on the message, including its length, from the adversary. (Of course, the length of the message can only be polynomial in the security parameter.) Encryption schemes that reveal more information on the length of messages, or schemes that hide the fact that a message was sent, can be modeled by appropriate modifications to $\mathcal{F}_{\text{ENC}}$.

On threshold decryption. Functionality $\mathcal{F}_{\text{ENC}}$ can be extended in a natural way to represent the security requirements of threshold decryption schemes. See more details in [CG99].

Securely realizing $\mathcal{F}_{\text{ENC}}$ versus CCA-security. Let $S = (gen, enc, dec)$ be an encryption scheme. (Here *gen* is the key generation algorithm, *enc* is the encryption algorithm and *dec* is the decryption algorithm.) Very loosely, S 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 e and a decryption key d. F is given e and access to a decryption oracle $dec(d, \cdot)$. When F generates a pair m_0, m_1 of messages, a bit $b \stackrel{R}{\leftarrow} \{0,1\}$ is chosen and F is given $c = enc(e, m_b)$. From this point on, F may continue querying its 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 [DDN91, RS91, BDPR98].

CCA-security is a standard security measure for encryption protocols. It is thus tempting to see whether CCA-security is sufficient for constructing protocols that securely realize $\mathcal{F}_{\text{ENC}}$. As

a first attempt, consider the following natural transformation from an encryption scheme S to a protocol $\tilde{\pi}_S$ for realizing $\mathcal{F}_{\text{ENC}}$. When invoked with input `(receiver,`id`)` within party P_i, protocol $\tilde{\pi}_S$ runs algorithm gen, keeps the private decryption key d, and sends the public encryption key e to all parties, together with the identifier id. When invoked with incoming message (id, e) from P_i, the protocol outputs `(receiver,`id, P_i`)`, and keeps the triple id, P_i, e. When invoked with input `(send,`id, m`)`, the protocol sends `(ciphertext,`$id, enc(e, m)$`)` to P_i. Finally, when invoked within P_i with incoming message `(ciphertext,`id, c`)`, the protocol outputs $dec(d, c)$.

It may seem at first that if S is CCA-secure then $\tilde{\pi}_S$ securely realizes $\mathcal{F}_{\text{ENC}}$. However, two problems arise here. First, if the adversary is adaptive then we may run into problems similar to the ones discussed in [CFGN96]. Second, even if the adversary is non-adaptive then $\tilde{\pi}_S$ allows corrupted parties to "copy" messages sent by other parties. That is, assume that an uncorrupted party P sends a ciphertext c to the receiver. Then a corrupted party may send the same c, thus copying the message sent by P. This adversarial behavior is not allowed by $\mathcal{F}_{\text{ENC}}$, thus protocol $\tilde{\pi}_S$ will cannot securely realize it.

A simple solution to the second problem mentioned above is to have each sender incorporate its identity within the encrypted message, and have the receiver accept decrypted messages only if the identity of the sender agrees with the identity specified in the message. That is, given an encryption scheme S we now construct a protocol π_S as follows. When invoked with input `(receiver,`id`)` within party P_i, or with incoming message (id, e) from P_i, protocol π_S proceeds as protocol $\tilde{\pi}_S$ does. When invoked with input `(send,`id, m`)` within party P_j, the protocol sends `(ciphertext,`$id, enc(e, P_j|m)$`)` to P_i (where '|' denotes concatenation). When invoked within P_i with incoming message `(ciphertext,`id, c`)` from P_j, π_S first computes $m' = dec(d, c)$ and checks whether $m' = P_j|m$ for some m. If so, then π_S outputs (id, P_j, m). Otherwise, π_S outputs nothing. We show:

Claim 9 *Let S be an encryption scheme that is CCA-secure. Then π_S securely realizes $\mathcal{F}_{\text{ENC}}$ with respect to non-adaptive adversaries.*

Proof (sketch): Let $S = (gen, enc, dec)$ be a CCA-secure encryption scheme, and let π_S be the protocol constructed from S as described above. Assume that $\pi_{\mathcal{S}}$ does not securely realize $\mathcal{F}_{\text{ENC}}$. That is, assume that there exists a (non-adaptive) real-life adversary $\mathcal{A}$ such that for any ideal-process adversary $\mathcal{S}$ there exists an environment $\mathcal{Z}$ that can tell whether it is interacting with $\mathcal{F}_{\text{ENC}}$ and $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{ENC}}$, or with π_S and $\mathcal{A}$ in the real-life model. We construct a CCA attacker F that succeeds against S in the experiment sketched above. (For simplicity we assume that all messages sent are of length at most k bits, and that the identity of a party is described using at most k bits. The more general case can be easily inferred.)

Before constructing F, consider the following ideal-process adversary, $\mathcal{S}$. $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$ and follows the instructions of $\mathcal{A}$, with the following exceptions:

1. When $\mathcal{S}$ receives an output `(receiver,`id, P_i`)` from $\mathcal{F}_{\text{ENC}}$, $\mathcal{S}$ runs the key generation algorithm gen, obtains a pair (e, d) of keys, and simulates for $\mathcal{A}$ a message that is sent from P_i to all parties and contains the encryption key e, together with the session identifier id.

2. Whenever $\mathcal{S}$ receives (id, P_j) from $\mathcal{F}_{\text{ENC}}$ (indicating that P_j sent a secret message to P_i), it proceeds as follows. If P_j is corrupted then $\mathcal{S}$ does nothing. Else, $\mathcal{S}$ computes $c = enc(e, 0^{2k})$ and simulates for $\mathcal{A}$ the event where P_j sends `(ciphertext,`id, c`)` to P_i.

3. Whenever the simulated $\mathcal{A}$ delivers a message `(ciphertext,`id, c`)` from a corrupted party P_j to P_i, $\mathcal{S}$ computes $m' = dec(d, c)$. If $m' = P_j|m$ for some m then $\mathcal{S}$ sends a message

(send,id, m) to $\mathcal{F}_{\text{ENC}}$, in the name of P_j. Otherwise (m' is of the wrong format) then $\mathcal{S}$ does nothing.

Note: In the construction of $\mathcal{S}$ we assumed that the receiver P_i is not corrupted (or, rather, that the environment activates an uncorrupted party to be the receiver). This is justified as follows. If P_i is corrupted then $\mathcal{S}$ learns all the messages sent to P_i, thus Step 2 can be modified so that c is an encryption of the real message sent to P_i. Consequently, in this case the environments view in the ideal process is identical to its view in the real-life model. It is thus sufficient to concentrate on the case where the receiver is not corrupted.

Now, assume that there exist a real-life adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ (with input z) such that $\mathcal{Z}$ can tell whether it is interacting with π_S and $\mathcal{A}$ in the real-life model or with $\mathcal{F}_{\text{ENC}}$ and $\mathcal{S}$ in the ideal process. Without loss of generality we assume that $\mathcal{Z}$ outputs either `ideal` or `real`. We construct an attacker F that breaks the CCA-security of S. Attacker F operates as $\mathcal{S}$ does, with the following exceptions. Like $\mathcal{S}$, F interacts with copies of $\mathcal{A}$ and $\mathcal{Z}$. Here, however, both $\mathcal{Z}$ and $\mathcal{A}$ are simulated within F. Furthermore:

1. Instead of running *gen* (in Step 1 above) to obtain the keys (e, d), F will hand $\mathcal{A}$ the public encryption key e in its input.

2. When $\mathcal{Z}$ asks an uncorrupted party P_j to send a message m to P_i, F outputs the two messages $(P_j|m, 0^{2k})$, and receives a ciphertext c^*. It then simulates for $\mathcal{A}$ the event where P_j sends (ciphertext,id,c^*) to P_i.

3. Whenever $\mathcal{S}$ runs the decryption algorithm (in step 3 above), F asks its oracle to decrypt the corresponding ciphertext. (The uniqueness of the identities guarantee that F does not ask its oracle to decrypt c^*.)

Finally, If $\mathcal{Z}$ outputs `real` then F outputs 0. If $\mathcal{Z}$ outputs `ideal` then F outputs 1.

It can be seen that if $b = 0$ (i.e., c^* is an encryption of $P_j|m$) then the view seen by $\mathcal{Z}$ is distributed as in a real-life interaction with π_S and $\mathcal{A}$. Similarly, if $b = 1$ (i.e., c^* is an encryption of 0^{2k}) then the view seen by $\mathcal{Z}$ is distributed as in an interaction in the ideal process for $\mathcal{F}_{\text{ENC}}$ and $\mathcal{S}$.

Note: For sake of simplicity, the above description assumes that $\mathcal{Z}$ has *exactly one* message encrypted by an uncorrupted party. It is easy to see that in order to distinguish between real and ideal $\mathcal{Z}$ has to ask some uncorrupted party to encrypt at least one message. The case where many messages are encrypted by uncorrupted parties is handled analogously to what is done above, using a standard hybrids argument. Specifically, if $\mathcal{Z}$ asks to encrypt up to l messages then define $l + 1$ hybrid interactions where in the ith interaction the adversary receives real encryptions of the first i messages, and encryptions of 0 in the other ciphertexts. $\mathcal{Z}$ is guaranteed to distinguish between the $(i-1)$th and the ith interaction for some $i = 1...l$. Thus F can choose i at random and proceed as above for the ith message. □

Remark: On the necessity of CCA security. It is interesting to note that π_S may securely realize $\mathcal{F}_{\text{ENC}}$ (and, thus, S may be regarded as a good encryption scheme), and still S may not be CCA-secure. For instance, let S be a CCA-secure encryption scheme, and construct S' from S as follows. The key generation algorithm remains unchanged. The encryption algorithm of S' first runs the encryption algorithm of S and then appends 1 to the ciphertext. The decryption algorithm of

S' first deletes all trailing 0s and then the trailing 1 from the received ciphertext, and then runs the decryption algorithm of S. It is easy to see that S' is no longer CCA-secure; nonetheless, $\pi_{S'}$ securely emulates $\mathcal{F}_{\text{ENC}}$ all the same.

Modeling symmetric encryption. Symmetric encryption seem to be best modeled via a secure channel functionality, that first waits to receive the identities of two parties, P_i and P_j, and then simply forwards messages between these two parties while notifying the adversary whenever a message was sent. This functionality, $\mathcal{F}_{\text{SCH}}$, is described in Figure 10.

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

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

1. Upon receiving a value $(\texttt{set-up}, id, P_j)$ from some party, P_i, wait to receive a value $(\texttt{set-up}, id, P_i)$ from P_j. Once this value is received, proceed to Step 2.
2. Upon receiving a value $(\texttt{send}, id, m)$ from P_i, send (id, m) to P_i and (id, P_i) to the adversary. Upon receiving a value $(\texttt{send}, id, m)$ from P_j, send (id, m) to P_i and (id, P_j) to the adversary.

Figure 10: The Secure Channel functionality, $\mathcal{F}_{\text{SCH}}$

Notice that $\mathcal{F}_{\text{SCH}}$ guarantees both secrecy and authenticity of the communication between the two parties. Indeed, a typical methodology for securely realizing $\mathcal{F}_{\text{SCH}}$ starts with a Key-Exchange protocol, and then encrypts and authenticates each message via symmetric encryption and message authentication codes using the shared key. (An alternative treatment of this methodology, not based on the current framework, is given in [CK00].)

The modeling of Pfitzmann and Waidner [PW00a]. Another modeling of the security provided by public-key encryption is given in [PW00a]. Their modeling is similar to that of functionality $\mathcal{F}_{\text{ENC}}$ (Figure 9), with the exceptions that they allow the adversary to "replay" old ciphertexts and that they deal with many receivers within the same copy of their functionality. (In contrast, functionality $\mathcal{F}_{\text{ENC}}$ does not allow replay, and handles only a single receiver. The case of many receivers is taken care of by the composition theorem.)

They also propose a protocol for securely realizing their functionality, for the case of non-adaptive adversaries in a model that is roughly equivalent to the unauthenticated-links model (the UM) here. Their protocol calls for signing each message together with the identity of the sender and receiver, and then encrypting it using a CCA-secure encryption scheme. (In the present framework, a protocol for securely realizing $\mathcal{F}_{\text{ENC}}$ in the UM can be obtained by composing a protocol that securely realizes $\mathcal{F}_{\text{ENC}}$ in authenticated networks with a protocol that securely realizes $\mathcal{F}_{\text{AUTH}}$, see Figure 7.)

7.4 Digital Signatures

Digital signature schemes allow one 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 by Diffie and Hellman

in [DH76]; their basic security requirements were formulated by Goldwasser, Micali and Rivest in [GMRi88].

Here we treat digital signatures as a cryptographic task to be run in a larger environment, thus providing another view of the security of such schemes. We present two alternative ideal functionalities that are aimed at capturing the security requirements of digital signatures. Both functionalities guarantee the strongest security requirement in [GMRi88] (impossibility of existential forgery in the presence of chosen message attacks). However, while the first functionality, $\mathcal{F}'_{\text{SIG}}$, seems a bit more "natural" at first, it is overly strong in the sense that it imposes some extra implicit restrictions on protocols that realize it. In contrast, the second functionality, $\mathcal{F}_{\text{SIG}}$, is easier to realize. Specifically, securely realizing $\mathcal{F}_{\text{SIG}}$ in the presence of *non-adaptive* adversaries is shown to be *equivalent* to resilience against existential forgery in the presence of chosen message attacks. Functionalities $\mathcal{F}'_{\text{SIG}}$ and $\mathcal{F}_{\text{SIG}}$ are presented in Figure 11 below. (We choose to present both functionalities in order to highlight the effect of relatively small changes in the formalization of ideal functionalities.)

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

$\mathcal{F}'_{\text{SIG}}$ proceeds as follows, running on security parameter k, with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. In the first activation, expect to receive a value $(\texttt{signer}, id)$ from some party P_i. Then, send $(\texttt{signer}, id, P_i)$ to all parties and the adversary. ¿From now on, ignore all $(\texttt{signer}, id)$ values.
2. Upon receiving a value $(\texttt{sign}, id, m)$ from P_i, choose a value $s_m \stackrel{R}{\leftarrow} \{0,1\}^k$ and write $(\texttt{signature}, id, m, s_m)$ on the incoming communication tape of P_i.
3. Upon receiving a value $(\texttt{verify}, id, m, \sigma)$ from some party P_j, let $f = 1$ if $\sigma = s_m$ and $f = 0$ otherwise. Write $(\texttt{verified}, id, m, f))$ on the incoming communication tape of P_j.

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

$\mathcal{F}_{\text{SIG}}$ proceeds as follows, running on security parameter k, with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. In the first activation, expect to receive a value $(\texttt{signer}, id)$ from some party P_i. Then, send $(\texttt{signer}, id, P_i)$ to all parties and the adversary. ¿From now on, ignore all $(\texttt{signer}, id)$ values.
2. Upon receiving a value $(\texttt{sign}, id, m)$ from P_i, hand $(\texttt{sign}, id, m)$ to the adversary. Upon receiving $(\texttt{signature}, id, m, \sigma)$ from the adversary, set $s_m = \sigma$, and write $(\texttt{signature}, id, m, \sigma)$ on the incoming communication tape of P_i
3. Upon receiving a value $(\texttt{verify}, id, m, \sigma)$ from some party P_j, proceed as follows. If m was signed before (i.e., s_m is defined) and $\sigma = s_m$ then let $f = 1$. If m was never before signed then let $f = 0$. If m was signed but $s_m \neq \sigma$ then hand $(\texttt{verify}, id, m, \sigma)$ to the adversary. Upon receiving $(\texttt{verified}, id, m, \phi))$ from the adversary let $f = \phi$. Once the value of f is set, write $(\texttt{verified}, id, m, f))$ on the incoming communication tape of P_j.

Figure 11: The two signature functionalities, $\mathcal{F}'_{\text{SIG}}$ and $\mathcal{F}_{\text{SIG}}$.

Remarks: **1.** Notice that, in contrast with all other functionalities presented in this paper, functionalities $\mathcal{F}'_{\text{SIG}}$ and $\mathcal{F}_{\text{SIG}}$ write the signature value and the verification answer directly on the input tapes of the signer and verifier, respectively (namely, these functionalities are *immediate.*) This represents signature and verification algorithms that are run locally and provide output in the same activation of the calling protocol.

2. The two functionalities differ in Steps 2 and 3, namely in the signing and the verification operations. The difference in the signing operation is that, while $\mathcal{F}'_{\text{SIG}}$ generates the signature value at random from some distribution, $\mathcal{F}_{\text{SIG}}$ lets the adversary choose the signature value. This extra leniency of $\mathcal{F}_{\text{SIG}}$ represents the fact that the the security requirements of signature schemes impose no requirement on the distribution of the signature string itself. In particular, it can depend on all the signed message (or, in fact, on all the messages signed in the past by the system) in some obvious way.

The difference in the verification process is in the case where the message m was signed in the past but the signature provided by the verifier is not generated by the signer. Functionality $\mathcal{F}'_{\text{SIG}}$ rejects that message. In contrast, $\mathcal{F}_{\text{SIG}}$ lets the adversary decide whether to accept or reject that signature. This extra leniency of $\mathcal{F}_{\text{SIG}}$ represents the fact that, according to the [GMRi88] notion of security, we do not care what the outcome of the verification algorithm is in such a case.

3. Functionality $\mathcal{F}_{\text{SIG}}$ may be modified somewhat to capture the functionality of threshold signature schemes. See more details in [CHH00].

Equivalence with the [GMRi88] notion of security. We show that securely realizing $\mathcal{F}_{\text{SIG}}$ is equivalent to resilience against existential forgery by chosen message attacks a la [GMRi88]. First, we describe how to translate a signature scheme $S = (gen, sig, ver)$ as in [GMRi88] into a protocol π_S in the present setting. This is done as follows: When P_i, running π_S, receives an input `(signer,`id`)`, it executes algorithm gen, keeps the signing key s and sends the verification key v to all parties. When the signer receives an input `(sign,`id, m`)`, it sets $\sigma = sig(s, m)$ and outputs `signature,`id, m, σ`)`. When any party gets an input `(verify,`id, m, σ`)`, it outputs `(verified,`$id, m, ver(v, m, \sigma)$.

Recall that *non-adaptive* adversaries are adversaries that choose the set of corrupted parties before the computation begins.

Claim 10 *Let $S = (gen, sig, ver)$ be a signature scheme as in* [GMRi88]. *Then π_S securely realizes $\mathcal{F}_{\text{SIG}}$ with respect to non-adaptive adversaries if and only if S is existentially unforgeable against chosen message attacks.*

Proof (sketch): Let $S = (gen, sig, ver)$ be a signature scheme, and assume that π_S securely realizes F_{SIG}. We show that S is existentially unforgeable against chosen message attacks. That is, assume that there is a [GMRi88] forger G against S. We construct an environment $\mathcal{Z}$ and a (non-adaptive) real-life adversary $\mathcal{A}$ such that, for any ideal-process adversary $\mathcal{S}$, environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{F}_{\text{SIG}}$ and $\mathcal{S}$ in the ideal process, or with π_S and $\mathcal{A}$ in the real-life model.

Environment $\mathcal{Z}$ proceeds as follows. It first activates some party P_i with input `(signer,`id`)` for some value of id (say, $id = 0$). ¿From now on, whenever $\mathcal{A}$ asks $\mathcal{Z}$ to sign a message m, $\mathcal{Z}$ will activate the signer with input `(sign,`id, m`)`, and will report the output to $\mathcal{A}$. When $\mathcal{A}$ asks $\mathcal{Z}$ to verify a pair (m, σ), $\mathcal{Z}$ will first verify that m was never signed before. (if m was signed before then $\mathcal{Z}$ outputs 0 and halts.) Next, $\mathcal{Z}$ activates some uncorrupted party with input `(verify,`id, m, σ`)` and outputs whatever that party outputs.

The real-life adversary $\mathcal{A}$ will first wait to hear some party P_i send a public verification key v. (This will happen when $\mathcal{Z}$ activates P_i to be the signer.) Then, $\mathcal{A}$ will run G on input v. Whenever G generates a message m to be signed, $\mathcal{A}$ asks $\mathcal{Z}$ to sign m. When $\mathcal{A}$ hears the signature σ from $\mathcal{Z}$, it hands σ to G. When G outputs a pair (m^*, σ^*) (supposedly a new message and its forged signature), $\mathcal{A}$ asks $\mathcal{Z}$ to verify (m^*, s^*).

It can be seen that whenever G succeeds (i.e., whenever $ver(m^*, \sigma^*) = 1$ and m^* was not previously signed by P_i), $\mathcal{Z}$ outputs 1. Thus, under the assumption that G succeeds with non-negligible probability, in the real-life model $\mathcal{Z}$ outputs 1 with non-negligible probability. However, in the ideal process with $\mathcal{F}_{\text{SIG}}$ $\mathcal{Z}$ never outputs 1, regardless of what the ideal-process adversary does.

For the other direction, assume that there is a (non-adaptive) real-life adversary $\mathcal{A}$ such 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 $\mathcal{S}$ in the ideal process, or with π_S and $\mathcal{A}$ in the real-life model. We construct a [GMRi88] forger G against S.

But first consider the following ideal-process adversary $\mathcal{S}$: $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$ and follows the instructions of $\mathcal{A}$, with the following exceptions. First, when $\mathcal{S}$ receives an output $\texttt{(signer,}id, P_i\texttt{)}$ from $\mathcal{F}_{\text{SIG}}$, it runs the key generation algorithm gen, obtains a pair (s, v) of keys, and simulates for $\mathcal{A}$ a message that is sent from P_i to all parties and contains the verification key v. Next, whenever $\mathcal{S}$ receives $\texttt{(sign,}id, m\texttt{)}$ from $\mathcal{F}_{\text{SIG}}$, it computes $\sigma = sig(s, m)$ and hands $\texttt{(signature,}id, m, \sigma\texttt{)}$ back to $\mathcal{F}_{\text{SIG}}$. Whenever $\mathcal{S}$ receives $\texttt{(verify,}id, m, \sigma\texttt{)}$ from $\mathcal{F}_{\text{SIG}}$, it hands $\texttt{(verified,}id, m, ver(v, m, \sigma)\texttt{)}$ back to $\mathcal{F}_{\text{SIG}}$.[10]

Let B denote the event that, in a run of π_S, $ver(v, m, \sigma) = 1$ for some message m and signature σ, but during the signer never signed m the execution of the protocol. We are guaranteed that there exist a real-life adversary $\mathcal{A}$ and environment $\mathcal{Z}$ that distinguishes between its run with $\mathcal{A}$ in the real life model and its run with $\mathcal{S}$ in the ideal process. However, as long as event B does not occur, the two interactions is identical from the point of view of $\mathcal{Z}$. Thus, we are guaranteed that in the real-life model event B occurs with non-negligible probability.

We turn to constructing the [GMRi88] forger G. G operates as $\mathcal{S}$ does, with the following exceptions. Like $\mathcal{S}$, G interacts with simulated copies of $\mathcal{A}$ and $\mathcal{Z}$. However, instead of running gen to obtain the keys (s, v), G will hand $\mathcal{A}$ the public verification key v in its input. Instead of running the signing algorithm to obtain $\sigma = sig(s, m)$, G will ask its oracle to sign m and will thus obtain the signature σ. Whenever the simulated $\mathcal{Z}$ activates some uncorrupted party with input $\texttt{(verify,}id, m, \sigma\texttt{)}$, G verifies that m was never signed before and $ver(v, m, \sigma) = 1$. Once such a pair (m, σ) is found, G outputs that pair and halts. (Here it is important that $\mathcal{A}$ is non-adaptive. In particular, G may not be able to mimic the behavior of $\mathcal{S}$ in the case where $\mathcal{A}$ corrupts the signer.)

Notice that, from the point of view of $\mathcal{A}$ and $\mathcal{Z}$, the interaction with G looks the same as an interaction in the real-life model with π_S. Thus, we are guaranteed that event B (and, thus, successful forgery by G) will occur with non-negligible probability. □

[10] In the construction of $\mathcal{S}$ we assumed that the signer P_i is not corrupted (or, rather, that the environment activates an uncorrupted party to be the signer). If the signer is corrupted then $\mathcal{S}$ simply follows the instructions of $\mathcal{A}$. In this case, the environment's view in the ideal process is identical to its view in the real-life model.

7.5 Commitment[11]

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).

In the present formalization, commitment protocols are protocols that securely realize an "ideal commitment" functionality. More specifically the proposed functionality, denoted $\mathcal{F}_{\text{COM}}$, proceeds as follows. The commitment phase is modeled by having $\mathcal{F}_{\text{COM}}$ receive a value $(\texttt{Commit}, id, P_i, P_j, x)$, from some party P_i (the committer). Here id is an Session ID (SID), P_j is the identity of another party (the receiver), and x is the value committed to. In response, $\mathcal{F}_{\text{COM}}$ lets the receiver P_j *and the adversary* $\mathcal{S}$ know that P_i has committed to some value, and that this value is associated with SID id. This is done by sending the message $(\texttt{Receipt}, id, P_i, P_j)$ to P_j and $\mathcal{S}$. The opening phase is initiated by the committer sending a value $(\texttt{Open}, id, P_i, P_j)$ to $\mathcal{F}_{\text{COM}}$. In response, $\mathcal{F}_{\text{COM}}$ hands the value $(\texttt{Open}, id, P_i, P_j, x)$ to P_j and $\mathcal{S}$. Functionality $\mathcal{F}_{\text{COM}}$ is presented in Figure 12 below. We say that a protocol is a composable commitment scheme if it securely realizes this functionality.

We stress that functionality $\mathcal{F}_{\text{COM}}$ corresponds to only a single invocation 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 [DDN91] and security for selective decommitment [G89, DNRS99], are met.

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

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

1. Upon receiving a value $(\texttt{Commit}, id, P_i, P_j, x)$ from P_i, send the value $(\texttt{Receipt}, id, P_i, P_j)$ to P_j and $\mathcal{S}$. (Ignore any subsequent Commit messages.)
2. Upon receiving a value $(\texttt{Open}, id, P_i, P_j)$ from P_i, proceed as follows: If a value $(\texttt{Commit}, id, P_i, P_j, x)$ was previously received from P_i then send the value $(\texttt{Open}, id, P_i, P_j, x)$ to P_j and $\mathcal{S}$. Otherwise, do nothing.

Figure 12: The Ideal Commitment functionality, $\mathcal{F}_{\text{COM}}$

Some variants of the Ideal Commitment Functionality. Functionality $\mathcal{F}_{\text{COM}}$ attempts to capture one standard variant of commitment schemes. Other variants are possible, providing different security properties. We sketch a few:

1. The functionality can be modified so that the adversary does not receive the opened value x. This captures the concern that the opening of the commitment should be available only to the receiver.

[11] This section is taken from [CF00] and is included here for self-containment only.

2. The functionality can be modified so that the receiver of the commitment provides $\mathcal{F}_{\text{COM}}$ with acknowledgments for obtaining the commitment and the opening, and $\mathcal{F}_{\text{COM}}$ forwards these acknowledgments to the committer. This may be useful in cases where the committer has to make sure that the receiver accepted the commitment and/or the opening.

3. The functionality can be modified so that the adversary receives no messages whatsoever. This captures the concern that the adversary does not learn whether a commitment protocol took place at all. (This requirement has a flavor of protection against traffic analysis.)

4. Functionality $\mathcal{F}_{\text{COM}}$ disallows "copying commitments". (That is, assume that party A commits to some value x for party B, and that the commitment protocol in use allows B to commit to the same value x for some party C, before A decommitted to x. Once A decommits to x for B, B will decommit to x for C.) Then this protocol does not securely realize $\mathcal{F}_{\text{COM}}$. One may wish to relax this strong requirement by allowing parties to "copy" commitments made by other parties. This can be done as follows. Modify $\mathcal{F}_{\text{COM}}$ to allow several parties to send values of the type `(Commit...)`. When the first $(\texttt{Commit}, id, P_i, P_j, x)$ value is received, set $v \leftarrow x$. Later, whenever some $P_{i'}$ that has sent a value $(\texttt{Commit}, id, P_{i'}, P_{j'}, ...)$ sends $(\texttt{Open}, id, P_{i'}, P_{j'})$, the functionality sends $(\texttt{Open}, id, P_{i'}, P_{j'}, v)$ to $P_{j'}$.

On the feasibility of realizing $\mathcal{F}_{\text{COM}}$. Composable commitment schemes in the common-random-string model are constructed in [CF00] based on any trapdoor permutation. Composable commitment schemes in the plain model are constructed based on the strong assumption of Hada and Tanaka [HT98]. It is also shown that composable commitment schemes in the plain model (i.e., without a common random string), whose security is proven via black-box simulation, do not exist. (Notice that, using standard terminology, the above definition guarantees only computational binding and only computational secrecy. This is in contrast with other definitions that typically guarantee either unconditional binding or unconditional secrecy. This fact is essential in the constructions of [CF00].)

7.6 Oblivious Transfer

Oblivious Transfer (OT) 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 introduced in [R81, EGL85] and studied in many works since.

In the present formalization, a secure OT protocol is a protocol that realizes an "ideal OT" functionality. The proposed functionality, $\mathcal{F}_{\text{OT}}$, is presented in Figure 13. (Using standard terminology, $\mathcal{F}_{\text{OT}}$ captures 1-out-of-l OT.)

On the feasibility of realizing $\mathcal{F}_{\text{OT}}$. Using the techniques of [EGL85, G98], $\mathcal{F}_{\text{OT}}$ can be realized in the $\mathcal{F}_{\text{COM}}$-hybrid model (i.e., it can be realized given any protocol that securely realizes $\mathcal{F}_{\text{COM}}$), if trapdoor permutations exist. This means that $\mathcal{F}_{\text{OT}}$ can be realized in the setting of Theorem 7, or in the common random string model, or alternatively under the strong assumption of Hada and Tanaka [HT98]. We do not know whether $\mathcal{F}_{\text{OT}}$ can be realized by a two party protocol in the "plain" asynchronous setting under standard assumptions. An interesting candidate is the protocol of [GM00].

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

$\mathcal{F}_{\text{OT}}$ proceeds as follows, running with security parameter k, parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Wait to receive a value $(\texttt{sender}, id, P_i, P_j, x_1, ..., x_l)$ from some party P_i, where each $x_m \in \{0,1\}^*$. Once such a value is received, ignore all subsequent $(\texttt{sender}...)$ values.
2. Wait to receive a value $(\texttt{receiver}, id, P_i, P_j, m)$ from P_j, where $m \in \{1..l\}$. Then, send $(\texttt{output}, id, x_m)$ to P_j, send (id, P_i, P_j) to the adversary, and halt.

Figure 13: The Oblivious Transfer functionality, $\mathcal{F}_{\text{OT}}$

7.7 Zero Knowledge

Zero-Knowledge [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 gets an additional input, w. If $R(x,w) = 1$ then the verifier should output 1. If there exists no w such that $R(x,w) = 1$ then the verifier should output 0. There is no requirement in the case where there exists w s.t. $R(x,w) = 1$ but the prover does not have such w as input.[12] In any case, the verifier should "learn nothing" from the protocol except of whether there exists a w such that $R(x,w) = 1$. See [G95] for many more details on and variants of Zero-Knowledge.

Cast in the present formalization, Zero-Knowledge (ZK) is treated as a two-party task in a multi-party, asynchronous network. The ZK functionality $\mathcal{F}_{\text{ZK}}$, presented in Figure 14, is willing to receive a w s.t. $R(x,w) = 1$ from *any* party, not only the prover specified by the verifier. This reflects the "plain" variant of ZK, where the verifier is only interested in the existence of such w, not in whether the specified verifier "knows" such w.[13] Nonetheless, $\mathcal{F}_{\text{ZK}}$ guarantees that only the prover specified by the verifier can cause the verifier to reject the input (by providing $\mathcal{F}_{\text{ZK}}$ with a w such that $R(x,w) = 0$).

As usual, the composition theorem guarantees that, if a protocol securely realizes $\mathcal{F}_{\text{ZK}}$ for some relation R, then security of this protocol is guaranteed even when many copies of a ZK protocol are running concurrently, in an interleaved way and with related inputs. This in particular guarantees that protocols that securely realize $\mathcal{F}_{\text{ZK}}$ are Concurrent Zero Knowledge protocols as in [F91, DNS98].

Designated-prover ZK. A stronger variant of ZK requires also that the "designated prover", i.e., the party specified by verifier, actually "knows" the witness w. This variant can be captured by modifying $\mathcal{F}_{\text{ZK}}$ so that it ignores all $(\texttt{prover}, id, w)$ messages that were not sent by P_j, the party specified in the first $(\texttt{verifier},...)$ message.

We remark that this variant is reminiscent of the concept of "proofs of knowledge", in the sense that it makes sure that if the verifier accepts x then the witness is "extractable" from the prover (via the ideal-process adversary). In addition, this variant guarantees "non-malleability" properties in a sense reminiscent of the one discussed in [DDN91, Sec. 5]. An interesting candidate for securely

[12] 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).

[13] For instance, the following behavior is not considered a breach of security: Party P_1 holds w such that $R(x,w)$ holds, and runs a ZK protocol to prove to P_2 that $\exists w$ *s.t.* $R(x,w)$. At the same time, P_2 (who does not have w) manages to prove to P_3 that $\exists w$ *s.t.* $R(x,w)$, by simply forwarding the messages of P_3 to P_1 and forwarding the messages of P_1 to P_3.

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

$\mathcal{F}_{\text{ZK}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$. The functionality is parameterized by a binary relation R.

1. Wait to receive a value $(\texttt{verifier}, id, P_i, P_j, x)$ from some party P_i. Once such a value is received, ignore all subsequent $(\texttt{verifier}...)$ values.
2. Upon receipt of a value $(\texttt{prover}, id, w)$ from some party, P_l, do:
   (a) If $R(x, w) = 1$ then send $(id, \texttt{accept})$ to P_i and halt.
   (b) Else ($R(x, w) = 0$), if $P_l = P_j$ then send $(id, \texttt{reject})$ to P_i and halt. If $P_l \neq P_j$ then do nothing (and wait for the next activation).

Figure 14: The Zero-Knowledge functionality, $\mathcal{F}_{\text{ZK}}$

realizing this variant in the common random string model is the protocol of Sahai [sa99].

On the feasibility of realizing $\mathcal{F}_{\text{ZK}}$. It is easy to see that in the "plain" asynchronous model functionality $\mathcal{F}_{\text{ZK}}$ cannot be securely realized by a two-party protocol, as long as the ideal-process adversary operates via black-box simulation of the real-life adversary.[14] This rules out a large number of known protocols (e.g., [GMRa89, GMW91, GK88, RK99]), at least unless a non black-box proof is found.

Nonetheless, there exist a number of ways to securely realize $\mathcal{F}_{\text{ZK}}$. Theorem 7 implies that $\mathcal{F}_{\text{ZK}}$ can be realized by protocols that involve a set of "helpers," such that only a fraction of which may be corrupted. (Although Theorem 7 is stated in a synchronous model, it applies also to this model.) The three-round protocol of Hada and Tanaka [HT98] realizes $\mathcal{F}_{\text{ZK}}$, albeit under a strong and non-standard hardness assumption. (This hardness assumption is what allows the proof of security of this protocol to avoid the impossibility for proofs that operate via black-box simulation.) Finally, we conjecture that the protocol of Damgard [D00] securely realizes $\mathcal{F}_{\text{ZK}}$ in the common random string model.

7.8 Verifiable Secret Sharing

Verifiable Secret Sharing (VSS) is a multi-party primitive for synchronous networks, 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 dealers input. It is required that at the end of the sharing phase the dealers 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 dealers input. We stress that, in contrast with commitment, here the shared value should be reconstructible even without the cooperation of the dealer.

VSS was introduced in [CGMA85]. Several definitions of VSS exist (e.g., [FM97, BGW88, GM95]), but no definition guarantees secure composition with other protocols (or with other copies of the same protocol) in our setting. Also, since VSS is inherently a "two step process", it cannot be

[14] Very roughly, this is because in the present framework the simulator (i.e., the ideal process adversary) cannot "rewind" the environment.

naturally captured as secure evaluation of some function. (Indeed, it is possible to construct valid VSS protocols that do not evaluate any function.)

The ideal VSS functionality, denoted $\mathcal{F}_{\text{VSS}}$, is parameterized by m, the minimum number of parties required in order to open a shared secret. (It is easy to see that $m + t \leq n$ should hold for the functionality to be realizable, where t is the maximum number of faults.) Also here, all the messages received and sent by $\mathcal{F}_{\text{SFE}}$ include a Message ID, id. This allows the calling protocol to distinguish between different copies of the functionality. The functionality is presented in Figure 15.

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

$\mathcal{F}_{\text{VSS}}$ proceeds as follows, running with parties $D, P_1, ..., P_n$ and an adversary $\mathcal{S}$. Let m be the number of parties required to open a shared secret.

1. Upon receiving a value `(input,`id`,`v`)` from D, send `(shared,`id`)` to all parties and $\mathcal{S}$. Ignore all future `(input,`id`,...)` values.
2. Upon receiving a value `(open,`id`)` from m parties, and if a value `(input,`id`,`v`)` has been received from D, then send `(open,`id`,`v`)` to all parties and $\mathcal{S}$.

Figure 15: The Verifiable Secret Sharing functionality, $\mathcal{F}_{\text{VSS}}$.

Remark: As in the case of SFE, functionality $\mathcal{F}_{\text{VSS}}$ can be generalized to deal with cases where only a subset of the parties participates in a protocol invocation. In addition, it may be specified that in the opening phase the shared secret is to be given only to a subset of the participants. Also, the identity of the dealer may be determined dynamically via an appropriate message from one or more parties.

On the feasibility of realizing $\mathcal{F}_{\text{VSS}}$. As in the case of secure function evaluation, Theorem 7 implies that $\mathcal{F}_{\text{VSS}}$ can be t-securely realized in a synchronous setting whenever $t < n/3$. In fact, the VSS protocols of [BGW88, FM97] by themselves t-securely realize $\mathcal{F}_{\text{VSS}}$ if the communication is ideally secure. Furthermore, we conjecture that the AVSS (Asynchronous VSS) protocols of [BCG93] and [BKR94, CR93] t-securely realize $\mathcal{F}_{\text{VSS}}$ in a non-blocking asynchronous setting with ideally secure communication, respectively for $t < n/4$ and $t < n/3$.

7.9 Secure Function Evaluation

Secure Function Evaluation (SFE) is a multi-party primitive for synchronous networks, 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 $f : (\{0,1\}^*)^n \times R \rightarrow (\{0,1\}^*)^n$ is a given function and $r \stackrel{R}{\leftarrow} R$. Several definitions for SFE exist, e.g. [GL90, MR91, B91, C00].

A first attempt to cast the ideal process of secure function evaluation as an ideal functionality may proceed as follows. First the ideal functionality waits to receive input values from all parties. Once all inputs are received, the functionality evaluates the function on these inputs, and writes the output value of each party on the incoming message tape of that party (or, alternatively, sends each party a message containing its output value).

Indeed, this functionality captures the limitations on the information gathered by the adversary and the correctness requirements from the outputs of the uncorrupted parties, in the case where the uncorrupted parties generate outputs. However, it fails to guarantee that the uncorrupted parties generate any output at all — even in a synchronous system where message delivery is guaranteed. (In fact, the protocol where no party generates any output securely realizes this functionality for any function f to be evaluated: All the ideal-process adversary has to do is to have the corrupted parties not send anything to the ideal functionality.)

The following ideal functionality, denoted $\mathcal{F}_{\text{SFE}}$, provides the additional guarantee that all uncorrupted parties terminate the protocol with output values. It is parameterized by t, the maximum number of corrupted parties. Functionality $\mathcal{F}_{\text{SFE}}$ (presented in Figure 16) expects to receive two different messages from each party P_i: The first message includes the input value contributed by P_i. The second message is a `ready` value, whose interpretation is that P_i is ready to evaluate the function. As soon as $\mathcal{F}_{\text{SFE}}$ receives $t+1$ `ready` messages, it evaluates the function based on the input values it received so far, and sends the corresponding part of the function value to each party. (The "application protocol" that invokes a SFE protocol π may instruct the parties to activate π with the `ready` input at the round following the round where the inputs are given.) All the messages received and sent by $\mathcal{F}_{\text{SFE}}$ include a Message ID, id. This allows the calling protocol to distinguish between different copies of the functionality, that may be running at the same time.

Functionality $\mathcal{F}_{\text{SFE}}$ allows $\mathcal{Z}$ (which controls the input values of the uncorrupted parties) to make sure that in the synchronous ideal process the parties always output their function values, regardless of what $\mathcal{S}$ does. In particular, at a given round $\mathcal{Z}$ can activate each party P_i with an input value x_i. At subsequent rounds, activate each party with input value `ready`. It is now guaranteed that all uncorrupted parties will generate output.

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

$\mathcal{F}_{\text{SFE}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$. (Recall that $\mathcal{S}$ can be either an ideal-process adversary or an adversary in the hybrid model.) Let t be the maximum number of corrupted parties, and let $f : (\{0,1\}^*)^n \times R \to (\{0,1\}^*)^n$.

Upon activation, set variables $x_1, ..., x_n \leftarrow \perp$ and $\ell \leftarrow 0$. Next:

1. Upon receiving a value `Input`(id, v) from P_i, set $x_i = v$.
2. Upon receiving a value `ready`(id) for the first time from some party, increment ℓ. Once $\ell = t + 1$, choose $r \stackrel{R}{\leftarrow} R$ and let $(y_1, ..., y_n) \leftarrow f(x_1, ..., x_n, r)$. Send a message `Output`$(id, y_i)$ to each party P_i.

Figure 16: The Secure Function Evaluation functionality for evaluating an n-party function f

Remarks: **1.** Functionality $\mathcal{F}_{\text{SFE}}$ assumes by default that all parties participate in the evaluation of the function. A natural extension is to let F_{SFE} interact only with a subset of the parties. The identities of the parties in the subset can be given to the functionality either as a fixed parameter, or as input.

2. It follows from the general result of Section 6 that functionality $\mathcal{F}_{\text{SFE}}$ can be t-securely realized in a synchronous network, whenever $t < n/3$.

3. We conjecture that functionality $\mathcal{F}_{\text{SFE}}$ can be t-securely realized also in *non-blocking asyn-*

chronous networks, using the techniques of [BCG93, BKR94]. Here, however, the interpretation of this functionality is slightly weaker: the "application" that calls the SFE protocol cannot guarantee that the `ready` values are given only after *all* uncorrupted parties have contributed their inputs. Thus it cannot be guaranteed that the function is evaluated based on the input values of all uncorrupted parties. Instead, it can only be guaranteed that the input values of $n - t$ parties are taken into account in the evaluation of 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 mention two possible directions for future work towards this goal.

Formulating and realizing functionalities. The most immediate direction for future research is to come up with protocols that securely realize the ideal functionalities presented in Section 7 (or alternatively to show that these functionalities are realizable by existing protocols). In particular, this includes the Commitment, Oblivious Transfer, Zero-Knowledge (with and without the 'designated prover' property), Secret Communication, and Signatures (against adaptive adversaries).

Furthermore, ideal functionalities for capturing the security requirements of other tasks can be written and subsequently realized. Here examples include agreement tasks (such as broadcast and Byzantine agreement); variants of authentication (such as the "one-sided authentication" mentioned in [sh99]); identification; threshold cryptography tasks such as threshold signatures and threshold decryption; group signatures; electronic-commerce tasks such as auctions, contract signing (either in the *optimistic model* of [ASW97] or in the plain model), anonymous transactions, and more.

We remark that the approach (taken in this work) 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], Key Exchange protocols [BR93, BCK98] and more. Furthermore, the present framework is even more restrictive in that it requires the ideal-process adversary to mimic the real-life adversary *for any environment machine.* Thus demonstrating security of protocols within the present framework may not be trivial. Let us point to two ways in which the task of constructing and proving security of protocols may be simplified. First, ideal functionalities can be formulated in a way that relaxes the requirements as much as possible and simplifies proving security of protocols. (Examples of how this can be done include the treatment of digital signatures in Section 7.4.) Second, security of protocols may be proven based on different hardness assumptions. A promising direction for a new type of hardness assumptions was pointed out by Hada and Tanaka in [HT98]. This type of assumption is attractive in that it circumvents the difficulties encountered by traditional proofs of security, and seems to allow proving security of relatively simple protocols in the present framework. Further study of the soundness and implications of this type of hardness assumption seems warranted.

Exploring connections with formal-methods calculi and automated analysis. Traditionally, proofs of security of cryptographic protocols (say, based on known hardness assumptions) are hand-written and require considerable proficiency and thought. In fact, security analysis of even simple systems in the present framework is a considerable undertaking. It thus seems that the only way to apply the framework to more complex tasks and systems is to *automate* the proof process.

Tools for automating the process of asserting properties of programs and communication protocols exist, say using model-checking techniques (see, e.g., [CGP99]). As mentioned in the Introduction, some of these tools have been applied to verifying security properties of protocols, e.g. [M94, L96, So99]. However, these tools represent cryptographic primitives as symbolic operations and thus lack in soundness: There is no guarantee that a protocol that passes the analysis is indeed 'secure'.

Potentially, definitions of security in the present framework may serve as a computationally-sound basis for formal-methods based automated analysis of protocols. More specifically, ideal functionalities, as defined and used here, may be regarded as idealizations of cryptographic tasks that "hide the dirty details of cryptography" from applications, but at the same time are realizable by actual protocols. Consequently, it may be possible to develop calculi of concurrent protocols where cryptography is represented via ideal functionalities that are in fact securely realizable. Such calculi may lead to automated proofs of security of protocols that maintain cryptographic soundness.

We note that first steps in essentially the same direction have already been taken. In particular, Abadi and Rogaway [AR00] demonstrate the cryptographic soundness of some formal derivation rules for "symbolic encryption". This promising line of research can potentially be extended (say, as described above) to more general and complex cryptographic tasks.

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 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, Mihir Bellare, Ivan Damgard, Mark Fischlin, Shafi Goldwasser, Rosario Gennaro, Shai Halevi, Yuval Ishai, Eyal Kushilevitz, Tal Malkin, Cathy Meadows, Silvio Micali, Moni Naor, Rafi Ostrovsky, Tal Rabin, Charlie Rackoff, Phil Rogaway, Victor Shoup, Paul Syverson and Michael Waidner.

References

[AG97] M. Abadi and A. D. Gordon, A calculus for cryptographic protocols: The spi calculus. In *proceedings of the 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), *International Conference on Theoretical Computer Science IFIP TCS 2000,* LNCS, 2000. On-line version at http://pa.bell-labs.com/ abadi/.

[ASW97] N. Ashokan, M. Schunter and M. Waidner, Optimistic protocols for fair exchange, *4th ACM Conference on Computer and Communication Security,* 1997.

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

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

[BH92] D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Advances in Cryptology — Eurocrypt '92*, LNCS No. 658, Springer-Verlag, 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.

[BPRR95] M. Bellare, E. Petrank, C. Rackoff and P. Rogaway, "Authenticated key exchange in the public key model," manuscript 1995–96.

[BR93] M. Bellare and P. Rogaway, "Entity authentication and key distribution", *Advances in Cryptology, - CRYPTO'93*, Lecture Notes in Computer Science Vol. 773, D. Stinson ed, Springer-Verlag, 1994, pp. 232-249.

[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.

[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 *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, 1988, and *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, 1989.

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

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

[CF00] R. Canetti and M. Fischlin, "Concurrently Composable Commitments", in preparation.

[CK00] R. Canetti and H. Krawczyk, "Analysis of key exchange protocols and their use for building secure channels", manuscript.

[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.

[CGH98] R. Canetti, O. Goldreich and S. Halevi. The Random Oracle Methodology, Revisited. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, Dallas, TX, May 1998. ACM.

[CG99] R. Canetti and Shafi 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.

[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,

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

[CCD88] D. Chaum, C. Crepeau, and I. Damgard. Multi-party Unconditionally Secure Protocols. In *Proc. 20th Annual Symp. on the Theory of Computing (STOC)*, pages 11–19, ACM, 1988.

[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.

[CDDHR99] R. Cramer, I. Damgard, S. Dziembowski, M. Hirt and T. Rabin ,"Efficient miltiparty computations secure agaisnt an adaptive adversary", *Eurocrypt,* 1999, pp. 311-326.

[D00] I. Damgard. Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. *Eurocrypt 00*, LNCS, 2000.

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

[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.

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

[DDN91] D. Dolev, C. Dwork and M. Naor, Non-malleable cryptography, *SIAM. J. Computing,* to appear. 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. In *40th Annual Symposium on Foundations of Computer Science*, 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.

[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-maeelable commitment schemes", *CRYPTO '00, LNCS 1880*, 2000, pp. 413-428.

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

[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.

[G89] O. Goldreich, Private communication, 1989.

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

[G95] O. Goldreich, *"Foundations of Cryptography (Fragments of a book)"*, Weizmann Inst. of Science, 1995. (Avaliable at http://philby.ucsd.edu)

[G98] O. Goldreich. *"Secure Multi-Party Computation"*, 1998. (Avaliable at http://philby.ucsd.edu)

[GK88] O. Goldreich and H. Krawczyk, On the Composition of Zero-Knowledge Proof Systems, *SIAM. J. Computing,* Vol. 25, No. 1, 1996.

[GL00] O. Goldreich and Y. Lindell, Session-key generation using human passwords only, manuscript, 2000.

[GMW91] O. Goldreich, S. Micali and A. Wigderson, "Proofs that yield nothing but their validity or All Languages in NP Have Zero-Knowledge Proof Systems", *Journal of the ACM,* Vol 38, No. 1, ACM, 1991, pp. 691–729. Preliminary version in *27th Symp. on Foundations of Computer Science (FOCS),* IEEE, 1986, pp. 174-187.

[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, Springer-Verlag, 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,* Springer-Verlag, 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.

[HT98] S. Hada and T. Tanaka. On the Existence of 3-Round Zero-Knowledge Protocols. In *Crypto98*, 1998.

[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.

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

[LMMS98] P. Lincoln, J. Mitchell, M. Mitchell, A. Schedrov, "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. Schedrov, "Probabilistic Polynomial-time equivalence and security analysis", *Formal Methods Workshop,* 1999. Avaiable at ftp://theory.standford.edu/pub/jcm/papers/fm-99.ps.

[L96] G. Lowe, Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR, *2nd International Workshop on Tools and Algorithms for the construction and analysis of systems,* Springer-Verlag, 1996.

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

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

Asiacrypt '94, LNCS 917, 1995, pp. 133-150.

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

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

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

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

[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.

[PW00a] B. Pfitzmann and M. Waidner, "A model for asynchronous reactive systems and its application to secure message transmission", IBM Research Report RZ 3304 (#93350), IBM Research, Zurich, December 2000.

secure reactive systems", *7th ACM Conf. on Computer and Communication Security*, 2000, pp. 245-254.

[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*, Springer LNCS 1592, pages 415–413.

[sa99] A. Sahai, Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security, *in Proceedings of 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS),* 1999.

[sh99] V. Shoup, "On Formal Models for Secure Key Echange", 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.

[Y82] A. Yao, "Protocols for Secure Computation", In *Proc. 23rd Annual Symp. on Foundations of Computer Science (FOCS),* pages 160–164. IEEE, 1982.

[Y82A] A. Yao, Theory and applications of trapdoor functions, In *Proc. 23rd Annual Symp. on Foundations of Computer Science (FOCS),* pages 80–91. 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.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'