Security and Composition of Multi-party Cryptographic Protocols[*]

Ran Canetti[†]

September 9, 1999

Abstract

We present general definitions of security for multi-party cryptographic protocols, with focus on the task of evaluating a probabilistic function of the parties' inputs. We show that, with respect to these definitions, security is preserved under a natural composition operation.

The definitions follow the general paradigm of known definitions; yet some substantial modifications and simplifications are introduced. The composition operation is the natural 'subroutine substitution' operation, formalized by Micali and Rogaway.

We consider several standard settings for multi-party protocols, including the cases of *eavesdropping, Byzantine, non-adaptive* and *adaptive* adversaries, as well as the *information-theoretic* and the *computational* models. In particular, in the computational model we provide the first definition of security of protocols that is shown to be preserved under composition.

Keywords: multi-party cryptographic protocols, security of protocols, secure function evaluation, composition of protocols.

[*] To appear at Journal of Cryptology.

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

Contents

1 Introduction

Designing *secure protocols* is one of the central tasks of cryptography. Here security is generally understood as guaranteeing, in the presence of adversarial behavior of some parts of the system, a set of correctness properties of the output values of the parties together with a set of secrecy requirements regarding the local data of the parties.

A general study of secure protocols started with the pioneering works of Yao and Goldreich, Micali, and Wigderson [Y86, GMW87]. On top of introducing this fundamental notion, these works suggest a general methodology for solving "any cryptographic protocol problem" in a secure way. They were followed by a large body of work that describe general constructions for solving protocol problems in various settings (most notably, [BGW88, CCD88, RB89, GL90, OY91]), as well as protocols for more specific tasks (e.g., [DF89, GJKR96, R98]).

In contrast to the great advances in constructing secure protocols, our understanding of the notion of security of protocols progresses more slowly. The first works in this field (and in particular [Y82, Y86, GMW87]) contain only an intuitive exposition of this notion. Several general definitions of security of protocols were subsequently formulated, most notably by Goldwasser and Levin [GL90], Micali and Rogaway [MR91], and Beaver [B91], where the work of Micali and Rogaway is considerably more comprehensive than others. More recently, a definition based on [GL90, MR91, B91] was presented in [C95]. (The definition of [C95] is closest in its approach to [B91].) While the general approach of these definitions is roughly the same, the definitions differ from each other in several substantial ways. See more details below.

Indeed, while the notion of secure protocols seems intuitively obvious, capturing the security requirements of a "cryptographic protocol problem" in a way that is both precise and workable is not an easy task. In particular, a large number of constructions of secure protocols that appear in the literature, including most of the constructions mentioned above, have never been rigorously proven secure. (An exception is the detailed exposition and analysis of [GMW87] that was recently made available in [G98].)

This paper aims at improving our understanding of the nature of secure computation and our ability to prove cryptographic security of protocols. As a first step, we present definitions of security for protocols, with emphasis on *simplicity* and *minimality.* (Here minimality means that the definition is aimed at making minimal requirements from secure protocols, while not losing in rigor and in relevance to our intuitive notion of security.) We build on the formalization of [C95] that seems convenient and flexible. In particular, the approach underlying that formalization has been used in a number of quite varied settings, e.g. [BCG93, CFGN96, CG96, HM97, BCK98, CHH98, CKOR98].

Next, we consider *composition* of protocols. An important (almost obligatory) property of a definition of secure protocols is a guarantee that a protocol obtained by "properly" composing together secure protocols is secure. This is needed both for designing cryptographic protocols in a modular way, and for proving their security in a clear and understandable manner. In particular, such a property would greatly simplify the proofs of security of known constructions.

We show that our definition of security provides this guarantee, in several standard settings and with respect to a natural composition operation suggested in [MR91]. (Previously only the definition of [MR91] was known to preserve security under this composition operation, in some of these settings.) We hope that the results and techniques presented here will contribute to the writing of easy to follow proofs of security of known protocols, such as [GMW87, BGW88, CFGN96, BCK98].

As in [GL90, MR91, B91, C95], this work concentrates on the very general task of evaluating a probabilistic function of the parties' inputs. (This task is often known as secure function evaluation.)

In addition, the definitional approach presented here can be readily applied to capturing the security requirements of a variety of other tasks.

1.1 Previous definitional efforts

A common paradigm underlying all efforts to define secure protocols is to guarantee that running a secure protocol is "just as good" as carrying out an idealized computational process where security is guaranteed. In the context of secure function evaluation this ideal process consists of having all parties hand their inputs to a trusted party, who locally evaluates the function and hands the appropriate portion of the function value to each party. The definitional efforts differ in the method by which this basic paradigm is fleshed out. Let us sketch the approaches of [GL90, MR91, B91]. We elaborate on these definitions in Appendix A.

The definition of Goldwasser and Levin [GL90] does not make explicit comparison with the ideal process. Yet, this definition can be viewed as making a comparison with the ideal process as follows. They start with defining *legal behavior* of an adversary; this behavior captures the adversary's limited capabilities in the ideal process. Next they define a notion of *robustness* of protocols that essentially means that any adversary can be 'emulated' by a legal one. A protocol securely evaluates some function if it is robust and in addition it correctly evaluates the function whenever the adversary is limited to legal behavior.

The comparison with the ideal process serves as strong motivation behind the formulation of the Micali and Rogaway definition [MR91]. Yet also here it is not explicitly used in the actual definition, which contains some additional technicalities. These technicalities make the definition of [MR91] more restrictive. Micali and Rogaway also define a general and natural composition operation of protocols and state that their definition is preserved under this composition operation. The composition operation discussed in this work is essentially taken from there. It was previously believed that the extra restrictiveness of their definition is *necessary* for proving that composition preserves security. Here we show that this is not the case. (They consider only protocols that evaluate *deterministic* functions, in *the secure channels setting*. The secure channels setting is defined in the sequel.) Micali and Rogaway's manuscript is quite comprehensive and contains many enlightening observations, discussions, and examples regarding secure multi-party protocols. We have benefited a lot from reading this work, as well as from attending [M96].

Beaver makes the comparison of a protocol with the ideal process more explicit [B91]. That is, first a general notion of comparing security of protocols is formulated. Next, a protocol for evaluating a given function is considered secure if it is at least as secure as the ideal process for evaluating that function. This approach is very similar to the one taken here, with some technical differences that are explained in the sequel. In addition, it is stated that security according to this definition is preserved under 'sequential composition'. That is, if secure protocols are invoked one after the other, the inputs for each are the local outputs from the previous one, then the resulting protocol securely evaluates the composed function, as long as all intermediate results are part of the output. As seen below, this composition operation is a special case of the one considered here.

1.2 The definitional approach taken here

We first formalize the 'ideal process' mentioned above. This process is aimed at capturing the desired *functionality* of the task at hand, and in particular rules out any unwanted behavior. For the task of secure function evaluation, the ideal process is formulated as follows. There is no communication among the parties; instead, all parties hand their inputs to an incorruptible "trusted party", who locally computes the desired outputs and hands them back to the parties. Thus in the ideal process the adversary, controlling a set of *corrupted* parties, is very limited: Essentially, it only learns and perhaps modifies the inputs and outputs of the corrupted parties.

Next, we say that a protocol securely performs the task at hand if executing the protocol (in a given model of distributed computation) amounts to 'emulating' the ideal process for that task. Emulating the ideal process is interpreted as follows. First we formalize the "output of running a protocol with a given adversary", in the given distributed model, as well as the "output of running the ideal process with a given adversary". This formalization is a key ingredient of the definition. Now, running the protocol emulates the ideal process if for *any* adversary attacking the protocol in the given distributed model, there *exists* an 'ideal process adversary' that manages to induce essentially the same output distribution in the ideal process. This way, we are assured that the only adversarial effects that can occur when running the protocol in the given distributed model are those that are explicitly allowed in the ideal process.

In a way, this approach is a generalization of the 'simulation approach' used in [G93] (rephrasing [GM84]) to define security of encryption functions and in [GMR89] to define Zero-Knowledge protocols. Yet, the formulation here is more complex, as it applies to the more complex domain of many parties.

This approach can, of-course, be applied to a large variety of 'adversary models'. We concentrate on several salient models, characterized via the following parameters. Throughout, the network is assumed to be synchronous, and the communication channels are ideally authenticated. Next, we make the following distinctions.

A first distinction is between passive and active adversaries. Passive adversaries (often called 'eavesdropping' adversaries) only gather information and do not modify the behavior of the parties. Such adversaries often model attacks that take place only after the execution of the protocol has completed. Active adversaries (often called 'Byzantine') cause the corrupted parties to execute some arbitrary, malicious code.

Another distinction is between non-adaptive and adaptive adversaries. A non-adaptive (or 'static') adversary controls an arbitrary but fixed set of corrupted parties. An adaptive (or 'dynamic') adversary chooses the identities of the parties to be corrupted during the computation, based on the information gathered so far. Non-adaptive adversaries allow for simpler formalization and protocols. Yet, considering adaptive adversaries forces protocols to address security concerns that are important in many real-world situations and not addressed in the non-adaptive formalization. (See more discussion at the preamble to Section 5.)

Yet another distinction is between the computational setting where the adversary learns all the communication among the parties and is restricted to probabilistic polynomial time, and the secure channels setting where channels are absolutely secure and the adversary has unlimited computational power. Obtaining protocols that are secure in the secure channels setting is often regarded as a 'stepping stone' on the way to obtaining secure protocols in the (more realistic) computational setting.

Other variations of these settings may of course be interesting. For instance, many works assume an *authenticated broadcast channel,* where it is guaranteed that any message that is received by one party is received by all parties. Also, the setting where the adversary is probabilistic polynomial time *and* learns only messages sent to corrupted parties is often convenient for designing protocols (e.g., [F87, CH94, GJKR96, G98, R98]). The definitions can be easily adapted to these settings.

In all the above models, we concentrate on the case of honest majority, where strictly less than half of the parties are corrupted at any time. When half or more of the parties are corrupted the definition has to be weakened somewhat. (Essentially, now an active adversary cannot be prevented from interrupting the computation at any time. Yet, the general definitional approach will remain largely unchanged.) See [Y86, GMW87, BG89, GL90, G98] for definitions and protocols for the case of dishonest majority.

Differences from previous definitions. While being inspired by Micali and Rogaway [MR91], and following the approach of [B91, C95] quite closely, the formalization here differs in several aspects. Let us highlight two points of difference from [B91, C95]. One is the (no longer necessary) requirement that the 'ideal process adversary' operates via *one-pass, black-box* simulation of the 'real-life' adversary. That is, the 'ideal process adversary' was restricted to have only oracle access to the 'real-life' adversary. More importantly, it was required that the simulated adversary is run only once and is not 'rewound'. This requirement is quite restrictive; in particular, in the case of computationally bounded adversaries it essentially prohibits the use of Zero-Knowledge proofs within secure protocols. Removing this requirement seems essential for good treatment of the computational model. (The definition of [MR91] uses a similar notion of simulation as [B91, C95]. In fact, it is a bit more restrictive.)

Another modification, relevant to the case of adaptive adversaries, is the treatment of the "information flow" between a single protocol execution and the external environment. Good modeling of this "information flow" is essential for successful treatment of secure protocol composition. In the definition here this is modeled by introducing an additional algorithmic entity, representing the external environment, to the model. This seems to better represent the effect of the external environment on a single execution; in particular it allows us to deal with composition of protocols even for the case of computationally bounded adversaries. See more details in Sections 2.1 and 5.

1.3 Modular composition

When designing a protocol for some task, we want to be able to break the task into several partial (presumably simpler) sub-tasks, design secure protocols for these sub-tasks, and then use the already designed protocols as subroutines in the solution for the given task. In other words, we want to support the following design methodology for secure protocols:

(1) Design a 'high-level' protocol for the given task assuming that other, simpler sub-tasks can be carried out securely.

(2) Design protocols that securely carry out these simpler sub-tasks.

(3) Construct a full-fledged protocol for the given task by plugging the simpler protocols as subroutines in the 'high-level' protocol.

We call this technique of combining protocols modular composition. (Modular composition was first formalized in this context by Micali and Rogaway [MR91]. There it is called reducibility of protocols). We want the security of protocols to be preserved under modular composition. That is, the security of the full-fledged protocol should follow from the security of the high-level design and the security of the subroutine protocols for their specified sub-tasks. In other words, we would like to have:

General Goal: *Suppose that protocols $\rho_1 ... \rho_m$ securely evaluate functions $f_1 ... f_m$ respectively, and that a protocol π securely evaluates a function g while using subroutine calls for ideal evaluation of $f_1 ... f_m$. Then the protocol $\pi^{\rho_1 \cdots \rho_m}$, derived from protocol π by replacing every subroutine call for ideal evaluation of f_i with an invocation of protocol ρ_i, securely evaluates g.*

Several other composition operations on protocols are considered in the literature. For instance, 'sequential composition' usually means simply running several (secure) protocols one after the other, and 'parallel composition' means running them in parallel at the same time. We note that these composition operations can be regarded as special cases of modular composition with the appropriate "high level" protocol. Consequently we consider modular composition as the main general tool for modular protocol design.

We achieve this goal with respect to the definitions in this paper, in the non-concurrent case where only a single subroutine invocation is in execution at any given time. We consider the settings described above (i.e., non-adaptive, adaptive, passive, active adversaries in the secure channels and computational settings). In particular, in the computational setting this is the first time a composition theorem is stated with respect to *any* definition. (In fact, we demonstrate a slightly more general result: *Any* protocol π that uses ideal evaluation calls to $f_1 \ldots f_m$ maintains its "functionality" when the ideal evaluation calls are replaced by invocations of $\rho_1 \ldots \rho_m$, respectively.)

1.4 Other related work

Goldreich [G98] presents a detailed exposition and proof of the general construction of [GMW87], for both the two-party and the multi-party cases. He treats the computational setting, but only with non-adaptive adversaries. The definitions used there are essentially the same as the ones here for the non-adaptive case. Also, that work does not present general purpose composition theorems, but rather composes the constructed protocols in an ad-hoc manner.

A notion of security for the case of deterministic functions, non-adaptive, passive adversaries in the secure channels setting is studied by Chor and Kushilevitz [CK89, K89]. (This notion of security is somewhat weaker than the one here, as argued in Remark 1, Section 4.2.) Reducibility of protocols w.r.t. the notion of security of [CK89, K89] is discussed in [KKMO97]. The notion of reducibility of [KKMO97] is different than the one here in that there no communication is allowed in the high-level protocol except for invocations of the specified subroutines.

Finally, our proofs of the composition theorem in the various settings follow and adopt the general structure of the sequential composition theorems for Zero-Knowledge as proven by Goldreich and Oren [GO94], adapting their techniques to our setting.

Organization. In Section 2 we motivate and informally present the general approach taken by our definitions. Section 3 reviews some basic notions used to formalize the definitions. Section 4 concentrates on the case of non-adaptive adversaries in the secure channels setting. This includes a definition of security, statement of the composition theorem, and a full proof.

Section 5 generalizes the treatment of Section 4 to the case of adaptive adversaries (still in the secure channels setting). An attempt is made to keep this section as self-contained as possible, at the expense of some repetition.

Section 6 deals with adaptive adversaries in the *computational* setting. Since the treatment is very similar to that of Section 5, this section is *not* self-contained, and should be read in conjunction with Section 5. The case of *non-adaptive* adversaries in the computational setting can be inferred quite easily.

Throughout Section 4-6, we develop the cases of passive and active adversaries 'side by side' (with emphasis on the more involved case of active adversaries). Although constructions for the two cases are quite different in nature, the corresponding definitions are similar and are best considered together.

In Appendix A we briefly discuss the definitional efforts of [MR91, GL90, B91, C95, CFGN96].

We remark that the text contains a number of long footnotes. These are used to discuss issues that are not vital to the main thrust of the paper and would make the main text less fluent. In particular, the footnotes can be skipped at first reading.

2 Defining secure protocols: The general paradigm

This section motivates and sketches the general definitional approach pursued in this work. The approach is common to the various adversary models (passive, active, non-adaptive, adaptive adversaries, in the secure-channels and computational settings). Also, while this paper concentrates on the task of secure function evaluation, the approach carries to other tasks as well. Section 2.1 presents the approach for the task of secure function evaluation. This case captures much of the essence of the problem. Other tasks are briefly mentioned in Section 2.2.

2.1 Secure function evaluation

Secure function evaluation is a general task where the parties are given inputs and should produce outputs according to a given specification, cast as a function of their inputs. (This function can be probabilistic; that is, for each input it specifies a *distribution* on the corresponding outputs.) We focus on the case where only a minority of the parties are corrupted. Still, the general approach presented here can be used to capture the security requirements for the case of *dishonest majority* (and in particular the two-party case).

First attempts. Two basic requirements come to mind when trying to capture the notion of secure function evaluation. The first is *correctness:* the "good" parties (i.e., the parties that are not corrupted by the adversary), should output "a correct" value of the function evaluated at the inputs of all parties. This requirement is somewhat complicated by the fact that the function may be probabilistic (thus the output should obey some predefined distribution), and more importantly by the fact that if the adversary is active then the corrupted parties cannot, in general, be prevented from arbitrarily changing their inputs to the computation.

The second requirement is *secrecy,* meaning that the adversary should not learn (from interacting with the parties) anything other than the (original) inputs of the corrupted parties, and the "correct" function values that the corrupted parties are to obtain. This requirement seems to call for a definition based on some notion of "simulation" of the adversary's view (as in the case of probabilistic encryption or zero-knowledge [GM84, G93, GMR89]), but it is not clear at this point in what setting the "simulator" should operate and what should be required of it.

A naive approach towards defining security may proceed by separately requiring correctness and secrecy. Yet, as observed in [MR91], this decomposition is problematic since the two requirements are "intertwined": On the one hand, the secrecy requirement depends on our definition of a "correct" function value. On the other hand, the correctness requirement must make sure that the input values that the corrupted parties "contribute" to the computation be chosen without knowledge of the inputs of the uncorrupted parties.

Let us sketch a simple example that demonstrates this issue. Assume that two parties wish to compute the exclusive-or of their one-bit inputs, and use the following protocol: First party A sends its input to party B; then B announces the result. Intuitively, this protocol is insecure since a corrupted B can influence the output of A by choosing the value it contributes to the computation based on A's input. Yet, this protocol maintains secrecy (which holds vacuously for this problem since each party can infer the input of the other party from its own input and the function value), and is certainly "correct" in the sense that the output fits the input that B "contributes" to the computation.

This example highlights the problems associated with *active* adversaries. Other, more subtle examples for definitions that allow an active adversary to "illegally" influence the outputs of the uncorrupted parties are described in [MR91]. Additional problems arise when from dealing with *probabilistic* functions. Interestingly, these problems arise even when the adversary is *passive.* Remark 2 in Section 4.2 contains an example that highlights these problems.

One may be tempted to try to augment the "correctness" and "secrecy" requirements so as to handle the problems exposed above. However, following this approach may be difficult and error-prone (if at all possible). Consequently, our definition follows a different approach, that blends together "correctness" and "secrecy" into a single security requirement. We first envision an "ideal process" for secure multi-party function evaluation. This process captures all that we want from a secure computation (and in particular, the above requirements). Then we say that a computation is secure if it "emulates" the ideal process, in some well-defined manner.

Our approach. The definition proceeds in three steps. First we formalize the 'real-life' computation, in a straightforward way. Here the parties interact according to their protocol, in some specific model of distributed computation (e.g., either synchronous or asynchronous), and in the presence of a real-life adversary that controls a set of corrupted parties and behaves according to some adversarial model (e.g., either passive or active, non-adaptive or adaptive, etc.). At the end of the computation the uncorrupted parties output whatever is specified in their protocol. The corrupted parties output a special symbol specifying that they are corrupted. The adversary, controlling the corrupted parties, outputs some arbitrary value; This value may include output any information gathered by the adversary during the computation.[1]

Next the following ideal process for multi-party function evaluation is formulated, in order to capture our requirements from a secure function evaluation. (The specifics of the ideal process correspond to the type of adversary in consideration, e.g. passive or active.) First an ideal-process adversary gets to control a set of corrupted parties (which is either fixed beforehand or chosen adaptively), and learns the inputs of the corrupted parties. If active adversaries are modeled, then the ideal-process adversary can also *modify* these inputs based on the information gathered so far. Next, all parties hand their (possibly modified) inputs to an incorruptible trusted party. The trusted party evaluates the given function at the given inputs and hands each party its designated output. The evaluated function can be probabilistic, in which case the trusted party tosses the necessary coins and uses the outcome to determine the function value. Finally, the uncorrupted parties output whatever they receive from the trusted party, the corrupted parties output some special symbol, and the adversary outputs some arbitrary value. (Also here, the adversary's output may contain any information gathered by the adversary in the ideal process. However, here this information is very limited: it consists only of the adversary's random input, the identities of the corrupted parties, their inputs, and the values they received from the trusted party.)

We say that a protocol π for evaluating a function is secure if it *emulates* the ideal evaluation process of the function, in the sense that any effect on the real-life computation achieved by a real-life adversary (from some class of real-life adversaries) can be also achieved in the ideal process by *some* ideal-process adversary (from the corresponding class of ideal-process adversaries). This requirement is formulated as follows. We first define, in both the ideal and real-life models, the *global output* of a computation on a given input. This is a random variable that consists of the concatenation of the outputs of all the parties and the adversary. Next we require that for *any* real-life adversary $\mathcal{A}$ (from some class) attacking a secure protocol π there *exists* an ideal-process adversary $\mathcal{S}$ (from the corresponding class) such that, *on any input,* the global output of the real-life computation in the presence of $\mathcal{A}$ is distributed similarly to the global output of the ideal

[1] In an equivalent and somewhat more natural formalization the corrupted parties output whatever is instructed by the adversary and the adversary has no output. The formalization here will be more convenient in the sequel.

process computation in the presence of $\mathcal{S}$. (By defining similarity to be either "equal distribution" or "statistical closeness" or "computational indistinguishability" we obtain different notions of security.)

Requiring that the outputs of the *corrupted* parties be distributed similarly in the ideal process and in the real-life computation forces the ideal-process adversary to generate an output that "looks like" the output of the real-life adversary, in spite of the fact that it only sees the information available in the ideal process. This guarantees *secrecy,* in the sense that the information gathered by the real-life adversary is computable even in the ideal process. Requiring that the output of the *uncorrupted* parties be similarly distributed in the ideal process and in the real-life computation guarantees *correctness,* in the sense that the real-life adversary cannot influence the outputs of the corrupted parties more than is possible in the ideal process. Furthermore, combining the outputs of the corrupted and the uncorrupted parties into a single random variable guarantees that the "intertwined" secrecy and correctness requirement, discussed above, is satisfied. (See also Remark 2, Section 4.2.)

We remark that the above notion of a protocol in some adversary model *emulating* an ideal process can be naturally extended to having the protocol emulate *another protocol* in some other adversary model. This extended notion of emulation is quite useful. In particular, it plays a key role in our presentation of the composition theorems.

Enabling secure composition. The definitional approach sketched above is aimed at capturing the security requirements from a protocol, in a simplified setting where a single protocol execution is considered *in vitro.* In order to guarantee security in a setting where several protocol executions may co-exist, and in particular in order to be closed under composition of protocols, a definition of security must guarantee the following property: Even adversaries that have already gathered some information on the current execution (say, via other protocol executions) will be unable to gather *additional* information on the current execution, or otherwise gain some unwanted advantage.

In the case of non-adaptive adversaries this property is guaranteed by letting the adversary have some arbitrary auxiliary input at the onset of the interaction. The auxiliary input represents the information gathered by the adversary during other protocol executions occurring before the current execution. The notion of emulation, sketched above, is extended to hold for *any* auxiliary input. See more details in Section 4. (Auxiliary inputs were first introduced in [GO94], in the context of sequential composition of Zero-Knowledge proofs. Further discussion appears there, as well as in [G95].)

In the case of adaptive adversaries the "information flow" between a single protocol execution and other executions cannot be fully captured by a piece of information given at the onset of the execution. In a nutshell, the problem is that whenever a party gets corrupted by the adversary, either during the protocol execution or after the execution is completed, the adversary sees internal data of this party both from that execution and from other protocol executions run by the party. We model this information flow by introducing an additional algorithmic entity, representing the external environment, both to the real-life and to the ideal models. This entity interacts with the adversary and the parties at several points throughout the execution. At these points, the environment provides the adversary with additional information, and receives information from the adversary. The notion of emulation is adapted as follows: A protocol π emulates the ideal process for evaluating f (namely, π securely evaluates f) if for any real-life adversary $\mathcal{A}$ (from some class of real-life adversaries), *and for any environment* $\mathcal{Z}$, there exist an ideal-model adversary $\mathcal{S}$ (from the corresponding class of ideal-process adversaries) such that the effect of $\mathcal{A}$ with environment $\mathcal{Z}$ on parties running π can be emulated by $\mathcal{S}$ in the ideal model for evaluating f *with the same* *environment $\mathcal{Z}$.* See more details in Section 5.

2.2 Beyond secure function evaluation

Although secure function evaluation is a very general task, it does not capture all the interesting functionalities of cryptographic protocols. Let us elaborate a bit. First, some cryptographic tasks are reactive, in the sense that they have several phases, where the output of one phase may be part of the input of the next phase, and where the security of the task imposes requirements on the outputs of all phases taken together. (Examples include commitment, secret-sharing, and more complex tasks such as encryption or signature schemes where the same key is used for the processing of many messages.) In addition, the requirement that a secure protocol evaluates a pre-defined function of the inputs may be too restrictive: Many cryptographic tasks can be securely carried out by protocols that do not evaluate *any* pre-defined function of the inputs. (Such protocols would still guarantee that some input-output relation is satisfied.)

Nonetheless, the definitional approach described in Section 2.1 can be adapted to capture the security requirements of other tasks. In fact, some definitions used in the literature to capture the security requirements of other tasks can be regarded as examples of such an adaptation. Examples include the tasks of distributed proactive signature schemes [CHH98], key-exchange and authentication [BCK98], and distributed public-key encryption [CG99]. This sub-section sketches the general paradigm that underlies these definitions and can possibly be used to capture the security requirements of other cryptographic tasks. The idea is to proceed in three steps, as follows:

1. Formulate an ideal model for executing the task at hand. Typically, this ideal model involves a trusted party whose functionality captures the security requirements from the task. This functionality will typically involve *repeated interaction* with the parties. An important ingredient in this step is defining the global output of an execution in the ideal model.

2. Formalize the global output of an execution of a protocol in the "real-life" model under consideration.

3. Say that a protocol π securely performs the task at hand if it "emulates" an execution in the ideal model, in the usual way: For any real-life adversary $\mathcal{A}$ there should exist an ideal-model adversary $\mathcal{S}$ such that the global output of running π with $\mathcal{A}$ in the real-life model is distributed similarly to the global output of running $\mathcal{S}$ in the ideal model. In the case of adaptive adversaries the notion of emulation is extended to include the environment machine, as sketched above.

3 Preliminaries

In this section we review some basic notions that underlie our formalization of the definitions. A distribution ensemble $X = \{X(k,a)\}_{k\in\mathbf{N},a\in D}$ is an infinite sequence of probability distributions, where a distribution $X(k,a)$ is associated with each values of $k \in \mathbf{N}$ and $a \in D$ for some domain D. (Typically, $D = \{0,1\}^*$.)

The distribution ensembles we consider in the sequel are outputs of computations (either in an ideal or in a 'real-life' model), where the parameter a corresponds to various types of inputs, and the parameter k is taken to be the security parameter. All complexity characteristics of our constructs are measured in terms of the security parameter. In particular, we will be interested in the behavior of our constructs when the security parameter tends to infinity.

Definition 1 (Equal distribution) *We say that two distribution ensembles X and Y are* equally distributed *(and write $X \overset{\mathrm{d}}{=} Y$) if for all k and all a we have that distributions $X(k,a)$ and $Y(k,a)$ are identical.*

Slightly abusing notations, we will also use $X(k,a) \overset{\mathrm{d}}{=} Y(k,a)$ to denote that distributions $X(k,a)$ and $Y(k,a)$ are identical.

Say that a function $\delta : \mathbf{N} \rightarrow [0,1]$ is negligible if for all $c > 0$ and for all large enough $k \in \mathbf{N}$ we have $\delta(k) < k^{-c}$.

Definition 2 (Statistical indistinguishability) *Let $\delta : \mathbf{N} \rightarrow [0,1]$. Two distribution ensembles X and Y have* statistical distance δ *if for all sufficiently large k and all a we have that*

$$\mathrm{SD}(X(k,a), Y(k,a)) < \delta(k)$$

where SD *denotes statistical distance, or total variation distance (that is,*
$\mathrm{SD}(Z_1, Z_2) = \frac{1}{2}\sum_a |\mathrm{Prob}(Z_1 = a) - \mathrm{Prob}(Z_2 = a)|$*).*

If δ is a negligible function then we say that X and Y are statistically indistinguishable *(and write $X \overset{\mathrm{s}}{\approx} Y$)*

Definition 3 (Computational indistinguishability [GM84, Y82A]**)** *Let $\delta : \mathbf{N} \rightarrow [0,1]$. We say that two distribution ensembles X and Y have* computational distance at most δ *if for every algorithm $\mathcal{D}$ that is probabilistic polynomial-time in its first input, for all sufficiently large k, all a, and all auxiliary information $w \in \{0,1\}^*$ we have:*

$$|\mathrm{Prob}(\mathcal{D}(1^k, a, w, x) = 1) - \mathrm{Prob}(\mathcal{D}(1^k, a, w, y) = 1)| < \delta(k)$$

Where x is chosen from distribution $X(k,a)$, y is chosen from distribution $Y(k,a)$, and the probabilities are taken over the choices of x, y, and the random choices of $\mathcal{D}$.

If ensembles X and Y have computational distance at most k^{-c} for all $c > 0$ then we say that X and Y are computationally indistinguishable *and write $X \overset{\mathrm{c}}{\approx} Y$.*

Note that Definition 3 gives the distinguisher $\mathcal{D}$ access to an arbitrary auxiliary information string w (thus making the definition a non-uniform complexity one). It is stressed that w is fixed before the random choices of X and Y are made.

Multiparty functions. The functions to be evaluated by the parties are formalized as follows. An n-party function (for some $n \in \mathbf{N}$) is a probabilistic function $f : \mathbf{N} \times (\{0,1\}^*)^n \times \{0,1\}^* \rightarrow (\{0,1\}^*)^n$, where the first input is the security parameter and the last input is taken to be the random input. We will be interested in functions that are computable in time that is polynomial in the security parameter. In particular, the lengths of the inputs and outputs are assumed to be bounded by a polynomial in the security parameter. See [G98] for a more complete discussion of conventions regarding such functions. (Extending the treatment to more complex multiparty function requires some small technical modifications.)

Intuitively, n-party functions are interpreted as follows. Let $e \overset{\mathrm{R}}{\leftarrow} D$ mean that element e is drawn uniformly at random from domain D, and let $f(k, \vec{x}, r_f)_i$ denote the ith component of $f(k, \vec{x}, r_f)$. Each party P_i (out of $P_1, \ldots, P_n$) has input $x_i \in \{0,1\}^*$, and wishes to evaluate $f(k, \vec{x}, r_f)_i$ where $r_f \overset{\mathrm{R}}{\leftarrow} \{0,1\}^t$ and t is a value determined by the security parameter. For concreteness we concentrate on inputs and random inputs in $\{0,1\}^*$. Other domains (either finite or infinite) can be encoded in $\{0,1\}^*$ in standard ways.

4 Non-adaptive adversaries

As discussed in the preamble of Section 5, non-adaptive security (i.e., security against non-adaptive adversaries) is considerably weaker than adaptive security. Still, we first present the non-adaptive case in full. This is done for two reasons. First, the definition and (especially) the proof of the composition theorem are considerably simpler in the non-adaptive case. Thus, it is a good 'warm-up' for the adaptive case. Second, some important protocols in the literature (e.g., [GMW87, F87]) are known to be secure only against non-adaptive adversaries (see [G98]). Thus, treatment of this case is of independent interest.

Throughout this section we restrict ourselves to the secure channels setting, where the adversary may be computationally unbounded and learns only messages sent to corrupted parties. In Section 6 we show how the treatment is adapted to settings where no secure channels exist, and security is provided only against probabilistic polynomial time adversaries.

Section 4.1 contains the definition of secure protocols. Further discussion on the definition is presented in Section 4.2. Section 4.3 presents the composition theorem, to be proven in Section 4.4.

4.1 Definition of security: The non-adaptive case

We define secure protocols in the non-adaptive case. The definitions for passive and active adversaries are developed side by side, noting the differences throughout the presentation.

Following the outline presented in Section 2, we first formalize the real-life model; next we describe the ideal process; finally the notion of emulation of the ideal process by a computation in the real-life model is presented.

The real-life model. An n-party protocol π is a collection of n interactive, probabilistic algorithms. Formally, each algorithm is an Interactive Turing machine, as defined in [GMR89]. We use the term party P_i to refer to the ith algorithm. (Figuratively, party P_i is a computer that executes the ith algorithm.) Each party P_i starts with input $x_i \in \{0,1\}^*$, random input $r_i \in \{0,1\}^*$, and the security parameter k. Informally, we envision each two parties as connected via a private communication channel. A more complete description of the communication among parties is presented below.[2]

A (non-adaptive) real-life adversary, $\mathcal{A}$, is another interactive (computationally unbounded) Turing machine describing the behavior of the corrupted parties. Adversary $\mathcal{A}$ starts off with input that contains the identities of the corrupted parties and their inputs. In addition, $\mathcal{A}$ receives additional, auxiliary input and a value k for the security parameter. We let z denote the input of $\mathcal{A}$. (The auxiliary input is a standard tool that allows us to prove the composition theorem. See Section 2.1 for discussion.) In addition, $\mathcal{A}$ has random input. [3]

Say that an adversary is t-limited if it controls at most t parties. (Formally, a t-limited adversary halts whenever its input contains the identities of more than t corrupted parties.)[4]

[2] We view n, the number of parties, as independent from the security parameter, k. This allows discussing cases where n is small with respect to the security parameter (e.g., a constant), as well as cases where n tends to infinity and has some some fixed relation with k. Furthermore, note that the parties do not necessarily know n in advance.

[3] We remark that the adversary, being computationally unbounded, need not be probabilistic. In fact, our formalization of the security requirement will be a non-uniform complexity one. In such a setting deterministic adversaries are as powerful as probabilistic adversaries *with comparable complexity*. Yet, we find it conceptually appealing to formulate the definition in terms of probabilistic adversaries.

[4] This paper concentrates on t-limited adversaries, where t is some threshold value. That is, it is assumed that the

In the sequel we often use a slightly less formal language for describing the participating entities and the computation. A formal description (in terms of interactive Turing machines) can be easily extracted from the one here.

The computation proceeds in rounds, where each round proceeds as follows. (The description below captures a fully connected, ideally authenticated, synchronous network with *rushing.* The term *rushing* refers to allowing the corrupted parties to learn the messages sent by the uncorrupted parties in each round, before sending their own messages for this round.) First the uncorrupted parties generate their messages of this round, as described in the protocol. (That is, these messages appear on the outgoing communication tapes of the uncorrupted parties.) The messages addressed to the corrupted parties become known to the adversary (i.e., they appear on the adversary's incoming communication tape). Next the adversary generates the messages to be sent by the corrupted parties in this round. If the adversary is passive then these messages are determined by the protocol. An active adversary determines the messages sent by the corrupted parties in an arbitrary way. Finally each uncorrupted party receives all the messages addressed to it in this round (i.e., the messages addressed to P_i appear on P_i's incoming communication tape.)[5]

At the end of the computation all parties locally generate their outputs. The uncorrupted parties output whatever is specified in the protocol. The corrupted parties output a special symbol, $-$, specifying that they are corrupted. (Figuratively, these parties did not participate in the computation at all.) In addition, the adversary outputs some arbitrary function of its view of the computation. The adversary view consists of its auxiliary input and random input, followed by the corrupted parties' inputs, random inputs, and all the messages sent and received by the corrupted parties during the computation. Without loss of generality, we can imagine that the adversary's output consists of its entire view.Figure 1 summarizes the real-life computational process.

We use the following notation. Let $\mathrm{ADVR}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})$ denote the output of real-life adversary $\mathcal{A}$ with auxiliary input z and when interacting with parties running protocol π on input $\vec{x}=x_1\ldots x_n$ and random input $\vec{r}=r_0\ldots r_n$ and with security parameter k, as described above (r_0 for $\mathcal{A}$, x_i and r_i for party P_i). Let $\mathrm{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})_i$ denote the output of party P_i from this execution. Recall that if P_i is uncorrupted then this is the output specified by the protocol; if P_i is corrupted then $\mathrm{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})_i =-$. Let

$$\mathrm{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r}) = \mathrm{ADVR}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r}), \mathrm{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})_1,\ldots,\mathrm{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})_n.$$

Let $\mathrm{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z)$ denote the probability distribution of $\mathrm{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z,\vec{r})$ where $\vec{r}$ is uniformly chosen. Let $\mathrm{EXEC}_{\pi,\mathcal{A}}$ denote the distribution ensemble $\{\mathrm{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z)\}_{k\in\mathbf{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$. (Here $\langle\vec{x},z\rangle$ denotes some natural encoding of $\vec{x},z$ as a single string.)

The ideal process. The ideal process is parameterized by the function to be evaluated. This is an n-party function $f:\mathbf{N}\times(\{0,1\}^*)^n\times\{0,1\}^*\to(\{0,1\}^*)^n$, as defined in Section 3. Each party P_i has input $x_i\in\{0,1\}^*$ and the security parameter k; no random input is needed. Recall that the parties wish to evaluate $f(k,\vec{x},r_f)_1,\ldots,f(k,\vec{x},r_f)_n$, where $r_f\overset{\mathrm{R}}{\leftarrow}\{0,1\}^s$ and s is a value determined by the security parameter, and P_i learns $f(k,\vec{x},r_f)_i$. A (non-adaptive) ideal-process-adversary $\mathcal{S}$ is an

adversary can corrupt any subset of up to t parties. This type of corruption structures was chosen for simplicity of exposition. The same definitional methodology holds with respect to other, more general corruption structures (e.g., [HM97, CDM98]), both in the non-adaptive and the adaptive cases.

[5] Different models, representing different real-life communication settings and network topologies, are of-course possible. In particular, if one is concerned only with feasibility results and is not concerned with efficiency then it may be simpler to let the parties talk in a 'round robin', where in each communication round only a single party sends messages. For sake of generality we do not restrict ourselves to this simpler model.

Execution of an n-party protocol by parties $P_1 \ldots P_n$ with adversary $\mathcal{A}$

1. (a) Each party P_i starts with the security parameter k, input x_i and random input r_i.
   (b) The adversary $\mathcal{A}$ starts with k, random input r_0, input z that includes a set $C \subset [n]$ of corrupted parties and their inputs $\{x_i | i \in C\}$, and additional auxiliary input.
2. Initialize the round number to $l \leftarrow 0$.
3. As long as there exists an uncorrupted party that did not halt, repeat:
   (a) Each uncorrupted party P_i, $i \notin C$, generates $\{m_{i,j,l} | j \in [n]\}$, where each $m_{i,j,l} \in \{0,1\}^*$ is a (possibly empty) message intended for party P_j at this round.
   (b) The adversary $\mathcal{A}$ learns $\{m_{i,j,l} | i \in [n], j \in C\}$, and generates $\{m_{i,j,l} | i \in C, j \notin C\}$.
   (c) Each uncorrupted party P_i, $i \notin C$, receives the messages $\{m_{j,i,l} | j \in [n]\}$.
   (d) $l \leftarrow l + 1$
4. Each uncorrupted party P_i, $i \notin C$, as well as $\mathcal{A}$, generate an output. The output of the corrupted parties is set to $\perp$.

Figure 1: A summary of the non-adaptive real-life computation.

interactive (computationally unbounded) Turing machine describing the behavior of the corrupted parties. Adversary $\mathcal{S}$ starts off with input that includes the identities and inputs of the corrupted parties, random input, auxiliary input, and the security parameter k.[6] In addition, there is an (incorruptible) trusted party, T, that knows k. The ideal process proceeds as follows.

Input substitution: The ideal-process-adversary $\mathcal{S}$ sees the inputs of the corrupted parties. If $\mathcal{S}$ is active then it may also alter these inputs based on the information known to it so far. Let $\vec{b}$ be the $|C|$-vector of the altered inputs of the corrupted parties, and let $\vec{y}$ be the n-vector constructed from the input $\vec{x}$ by substituting the entries of the corrupted parties by the corresponding entries in $\vec{b}$. If $\mathcal{S}$ is passive then no substitution is made and $\vec{y} = \vec{x}$.

Computation: Each party P_i hands its (possibly modified) input value, y_i, to the trusted party T. Next, T chooses $r_f \overset{\text{R}}{\leftarrow} \mathcal{R}_f$, and hands each P_i the value $f(k, \vec{y}, r_f)_i$.[7]

Output: Each uncorrupted party P_i outputs $f(k, \vec{y}, r_f)_i$, and the corrupted parties output $-$. In addition, the adversary outputs some arbitrary function of the information gathered during the computation in the ideal process. This information consists of the adversary's random input, the corrupted parties' inputs and the resulting function values $\{f(k, \vec{y}, r_f)_i : \ P_i$ is corrupted$\}$.

Let $\text{ADVR}_{f,\mathcal{S}}(k, \vec{x}, z, \vec{r})$, where $\vec{r} = (r_f, r)$, denote the output of ideal process adversary $\mathcal{S}$ on security parameter k, random input r, and auxiliary input z, when interacting with parties having

[6] In contrast with the real-life adversary, it is essential that the ideal-process adversary be probabilistic. This holds even in our non-uniform complexity setting. Also, there is no need to explicitly limit the number of corrupted parties in the ideal process. The definition will guarantee that the identities of the corrupted parties in the ideal process are identical to the identities of the corrupted parties in the real life model.

[7] This formalization means that r_f, the 'internal random choices of f', remains unknown to the parties except for the information provided by the value of f.

input $\vec{x} = x_1 \ldots x_n$, and with a trusted party for evaluating f with random input r_f. Let the $(n+1)$-vector

$$\textsc{ideal}_{f,\mathcal{S}}(k,\vec{x},z,\vec{r}) = \textsc{advr}_{f,\mathcal{S}}(k,\vec{x},z,\vec{r}), \textsc{ideal}_{f,\mathcal{S}}(k,\vec{x},z,\vec{r})_1 \ldots \textsc{ideal}_{f,\mathcal{S}}(k,\vec{x},z,\vec{r})_n$$

denote the outputs of the parties on inputs $\vec{x}$, adversary $\mathcal{S}$, and random inputs $\vec{r}$ as described above (P_i outputs $\textsc{ideal}_{f,\mathcal{S}}(k,\vec{x},z,\vec{r})_i$). Let $\textsc{ideal}_{f,\mathcal{S}}(k,\vec{x},z)$ denote the distribution of $\textsc{ideal}_{f,\mathcal{S}}(k,\vec{x},z,\vec{r})$ when $\vec{r}$ is uniformly distributed. Let $\textsc{ideal}_{f,\mathcal{S}}$ denote the distribution ensemble $\{\textsc{ideal}_{f,\mathcal{S}}(k,\vec{x},z)\}_{k\in\mathrm{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$.

Comparing computations in the two models. Finally we require that protocol π emulates the ideal process for evaluating f, in the following sense. For any (t-limited) real-life adversary $\mathcal{A}$ there should exist an ideal-process adversary $\mathcal{S}$, such that $\textsc{ideal}_{f,\mathcal{S}} \overset{\mathrm{d}}{=} \textsc{exec}_{\pi,\mathcal{A}}$. Spelled out, this requirement means that for any value of the security parameter k, for any input vector $\vec{x}$ and any auxiliary input z, the global outputs $\textsc{ideal}_{f,\mathcal{S}}(k,\vec{x},z)$ and $\textsc{exec}_{\pi,\mathcal{A}}(k,\vec{x},z)$ should be identically distributed.[8]

We require that the complexity of the ideal-process adversary $\mathcal{S}$ be comparable to (i.e., polynomial in) the computational complexity of the real-life adversary $\mathcal{A}$. Introducing complexity issues in this seemingly "information theoretic" model may appear awkward and out of place at a first glance. However, a second inspection will verify that this requirement is very desirable. See Remark 1 in Section 4.2.[9]

Definition 4 (non-adaptive security in the secure channels setting) *Let f be an n-party function and let π be a protocol for n parties. We say that π* non-adaptively, t-securely evaluates f *if for any (non-adaptive) t-limited real-life adversary $\mathcal{A}$ there exists a (non-adaptive) ideal-process adversary $\mathcal{S}$ whose running time is polynomial in the running time of $\mathcal{A}$, and such that*

$$\textsc{ideal}_{f,\mathcal{S}} \overset{\mathrm{d}}{=} \textsc{exec}_{\pi,\mathcal{A}}. \tag{1}$$

If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries then we say that π non-adaptively, t-privately evaluates g.

Relaxed variants of Definition 4 are obtained by requiring that the two sides of (1) be only statistically indistinguishable, or even only computationally indistinguishable. (The last relaxation is aimed at the case where the adversary is assumed to be probabilistic polynomial time.) Furthermore, if Definition 4 is satisfied with the exception that the two sides of (1) have statistical (resp., computational) distance at most δ then we say that protocol π achieves statistical (resp., computational) distance δ.

4.2 Discussion

This section contains further discussion on Definition 4.

[8] In the case where the inputs are taken from a finite domain and equal distribution is required, a simpler formalization that does not introduce ensembles is sufficient. (Basically, the simpler formalization fixes the security parameter to an arbitrary value.) We use the current formalization in order to accommodate infinite input domains, indistinguishability of ensembles, and computationally bounded adversaries.

[9] Here we implicitly assume that the complexity of the protocol π run by the uncorrupted parties is bounded by a polynomial in the complexity of the adversary. If this is not the case then $\mathcal{S}$ is allowed to be polynomial in the complexity of π.

Remark 1: On the complexity of the ideal-process adversary. We motivate our requirement that the running time of the ideal-process adversary be polynomial in that of the real-life adversary, even in this seemingly "information theoretic" setting. The ideal-process adversary is an imaginary concept whose purpose is to formalize the following requirement: "Whatever gain the adversary obtains from interacting with parties running π, could have also been obtained in an ideal setting where a trusted party is used". Arguably, this requirement also means that interacting with π should not allow the adversary to obtain some gain "for free", where obtaining the same gain in the ideal process requires considerable computational resources. This aspect of the security requirement is captured by appropriately limiting the computational power of the ideal process adversary. As seen below, failing to do so results in a considerably weaker notion of security. (We remark that this weaker notion may still be of some interest for studying purely information-theoretic aspects of secure computation.)

Let us illustrate this distinction via an example. Let $f(x,y) = g(x \oplus y)$ where g is a one-way permutation and $\oplus$ denotes bitwise exclusive-or. Assume that parties A and B have inputs x and y respectively, and consider the following protocol for evaluating f: Party A announces x, party B announces y, and both parties evaluate $f(x,y)$. Our intuition is that this protocol is insecure against adversaries that may corrupt one party (say, B): It "gives away for free" both x and y, whereas computing x given only y and $f(x,y)$ may take the adversary a large amount of time. Indeed, if the real-life and ideal-process adversaries are limited to probabilistic polynomial time (and one-way permutations exist), then this protocol is *not* secure against adversaries that corrupt one party. However, if $\mathcal{S}$ is allowed unlimited computational power regardless of $\mathcal{A}$'s complexity, this protocol is considered secure since $\mathcal{S}$ can invert g.

Another distinction between the two notions has to do with constructing protocols in the *computational* setting. A convenient design paradigm for secure protocols in this setting proceeds as follows: First design a secure protocol π in the secure channels setting. Then, construct a protocol π' from π by encrypting each message. Indeed, it can be readily seen that if π is secure in the secure channels setting according to the definition here (and an appropriate encryption scheme is used) then π' is secure in the computational setting.[10] However, if the above, weaker notion of security is used then this transformation does not necessarily work.

Finally, we remark that other definitions of secure protocols do not make this distinction. (Examples include the [B91] definition, as well as the definition of private protocols in [CK89, K89, KKMO97].) Nonetheless, the protocols described in these works seem to be secure even according to the definition here. (In fact, we are not aware of protocols in the literature that were proven secure according to the above weaker definition, but are insecure according to the definition here.)

Remark 2: Combining correctness and secrecy. The requirement, made in Definition 4, that the global outputs of the two computations be equally distributed imposes several requirements on the ideal-process adversary. In particular, it implies:

(a) Secrecy. *The output of the real-life adversary is distributed equally to the output of the ideal-process adversary.*

(b) Correctness. *The outputs of the uncorrupted parties are equally distributed in the two models.*

Can the definition be weakened to require only that the global output of the ideal-process satisfies (a) and (b)?

It was argued in Section 2 that separately requiring secrecy and correctness does not restrict

[10] For instance, semantically secure encryption (as in [GM84]) is sufficient in the non-adaptive model, provided that a different pair of public and private keys are used for each pair of parties. We omit further details.

the “influence” of the adversary on the outputs of the uncorrupted parties, thereby resulting in unsatisfactory definitions. Yet, the weakened definition proposed here does combine correctness and secrecy to some extent (since the same ideal-process adversary has to satisfy both requirements). Indeed, the example protocol given in Section 2 (and also the examples in [MR91]) are insecure even under this weakened definition.

Nonetheless, we argue that the two *entire* $(n+1)$-vectors describing the global outputs of the two computations must be identically distributed, and it does not suffice to separately require (a) and (b) (i.e., that the two relevant projections of the global outputs are identically distributed). This point is demonstrated via an example: Consider two parties A and B that wish to evaluate the following 2-party function. Both parties have empty input; A should output a random bit, and B should have empty output. Of-course, A can simply output a random bit without any interaction; yet, consider the protocol where A also sends B the value of its output. B is instructed to ignore A's message and output nothing. This protocol is clearly insecure; yet it satisfies the above weakened definition.[11]

Put in other words, the above example highlights an additional weakness of separating the correctness and secrecy requirements, on top of the weakness discussed in Section 2. While the discussion in Section 2 concentrates on problems related to active adversaries, the example here highlights problems related to probabilistic functions. In particular, the insecure protocol suggested here satisfies the weakened definition even if the adversary is passive. This means that, when dealing with probabilistic functions, secrecy and correctness cannot be separately required *even for passive adversaries.*

Remark 3: On one pass black-box simulation. In [MR91, B91, C95] the notion of emulation is more restrictive in two respects. First, it is required that the ideal-process adversary be restricted to having only black-box access to the real-life adversary. More substantially, the adversary can be run only once and is never ‘rewinded’. We call this type of simulation one pass black-box. The second restriction is quite limiting. In particular, in the computational setting it prohibits usage of zero-knowledge protocols within secure protocols. (This is so since demonstrating the zero-knowledge property via black-box simulation requires rewinding the adversary.)

It was speculated in [C95, CFGN96] (and, implicitly, also in [MR91, B91]) that restricting the ideal-process adversary to one pass black-box simulation is needed in order to prove a general composition theorem. In this work we show that the modular composition theorem holds in the non-concurrent case even if the ideal-process adversary is not restricted to black-box simulation.

Recall that in the context of zero-knowledge existence of a black-box simulator implies existence of a simulator even for adversaries that have arbitrary auxiliary input [GO94]. Using the same technique, it can be seen that a similar result holds with respect to Definition 4.

Remark 4: On universal adversaries. The introduction of the auxiliary input (and the quantification over all auxiliary inputs) makes the quantification over all real-life adversaries unnecessary: It suffices to consider a *single* real-life adversary, namely the ‘universal adversary’ $\mathcal{U}$. Adversary $\mathcal{U}$ will receive in its auxiliary input a description of an arbitrary adversary machine $\mathcal{A}$ and will run $\mathcal{A}$. (Note that the complexity of $\mathcal{U}$ running $\mathcal{A}$ is only slightly more than the complexity of

[11] We sketch a proof. The case where A is corrupted is straightforward. If B is corrupted then, for each real-life adversary $\mathcal{B}$ (that controls B), construct the following ideal process adversary $\mathcal{S}$: Run a copy of $\mathcal{B}$, giving it a random bit b' for the output of $\mathcal{A}$, and output whatever $\mathcal{B}$ outputs. The bit b' will be different (with probability one half) from the output of A in this execution, thus Equation (1) will not be satisfied. Yet, as long as the outputs of parties A and B are considered separately the simulation is valid.

$\mathcal{A}$.) Consequently, in order to show security of a protocol it suffices to show a single ideal-process adversary: the one that satisfies Definition 4 with respect to $\mathcal{U}$.

Another consequence of this observation follows. One may wish to strengthen Definition 4 to require that there exists an *efficient transformation* from real-life adversaries to the corresponding ideal-process adversaries. The above argument shows that such strengthening is unnecessary.

Remark 5: On "initially-adaptive" adversaries. Consider the following variant of Definition 4. Instead of having the set of corrupted parties given to the adversary as part of its input, let the adversary (both in the real-life and ideal models) choose the identities of the corrupted parties, one by one in an adaptive way, but under the restriction that all corruptions must be made before any communication takes place among the parties. Call this model initially-adaptive.

We observe that security in the initially-adaptive model is equivalent to security in the non-adaptive model (as in Definition 4). Intuitively, this follows from the fact that, until the point where the first message is sent, the real-life and ideal models are identical. Therefore, any advantage (over non-adaptive adversaries), gained in the real-life model by the ability to adaptively corrupt parties before the interaction starts, can also be gained in the initially-adaptive ideal model.

A sketch of proof follows. Clearly initially-adaptive security implies non-adaptive security. (The argument is similar to that of Remark 1, Section 5.2.) Assume that a protocol π is secure according to Definition 4, and let $\mathcal{A}$ be an initially-adaptive real-live adversary. We construct an initially-adaptive ideal model adversary $\mathcal{S}$ that emulates $\mathcal{A}$.

Let $\mathcal{A}'$ be the adversary in the (standard) non-adaptive real-life model that gets in its auxiliary input an internal state of $\mathcal{A}$ at the point where $\mathcal{A}$ is done corrupting parties, and runs $\mathcal{A}$ from that state on. Let $\mathcal{S}'$ be the ideal-model adversary, guaranteed by Definition 4, that emulates $\mathcal{A}'$. Construct the ideal-model adversary $\mathcal{S}$ as follows. First $\mathcal{S}$ follows, in the ideal model, the corruption instructions of $\mathcal{A}$. Let σ be the state of $\mathcal{A}$ once it is is ready to start interacting with the parties. Next, $\mathcal{S}$ runs $\mathcal{S}'$ with state σ given as auxiliary input. It can be seen that $\mathcal{S}$ is a valid initially-adaptive ideal-model adversary, and that $\mathcal{S}$ emulates $\mathcal{A}$.

Remark 6: On related inputs. Definition 4 requires the protocol to "behave properly" on *any* set of inputs to the parties. However, in many real-world situations the participants expect to have inputs that are correlated in some way (say, the parties have some common input, or inputs that are taken from a certain distribution), and no requirements are made from the protocol in the case that the inputs are not of the expected form. The definition can be relaxed to accommodate such weakened security properties by placing appropriate restrictions on the domain of the inputs of the parties. (Alternatively, the evaluated function could be re-defined to return some error value in cases where the inputs are not in the appropriate domain.)

4.3 Modular composition: The non-adaptive case

Recall that we want to break a given task (i.e., a protocol problem) into several partial sub-tasks, design protocols for these partial sub-tasks, and then use these protocols as subroutines in a solution for the given task. For this purpose, we want to formalize and prove the informal goal stated in the Introduction. We do this for the non-concurrent case, where at most one subroutine call is made at any communication round. This section concentrates on non-adaptive adversaries in the secure channels setting.

Formalization and derivation of the composition theorem is done in two steps. First, we state a more general theorem, that holds for *any* protocol π (not only protocols that securely evaluate functions): replacing ideal evaluation calls made by π, with sub-protocols that securely evaluate the corresponding functions, results in a protocol that has essentially the same input-output functionality as π. The composition theorem from the Introduction follows as an easy corollary.

The hybrid model. We start by specifying the model for evaluating an n-party function g with the assistance of a trusted party for evaluating n-party functions $f_1, \ldots, f_m$, and define secure protocols in that model. The model, called the hybrid model with ideal access to $f_1, \ldots, f_m$ (or in short the $(f_1, \ldots, f_m)$-hybrid model), is obtained as follows. We start with the real-life model of Section 4.1. This model is augmented with an incorruptible trusted party T for evaluating $f_1, \ldots, f_m$. The trusted party is invoked at special rounds, determined by the protocol. (For simplicity of exposition we assume that the number of ideal evaluation calls, the rounds in which the ideal calls take place, and the functions to be evaluated depend only on the security parameter. In addition we assume that m, the number of different ideally evaluated functions, is fixed.[12]) In each such round a function f (out of $f_1, \ldots, f_m$) is specified. The computation at each special round mimics the ideal process. That is, all parties hand their f-inputs to T (party P_i hands x_i^f). As in the ideal process, an active adversary decides on the input values that the corrupted parties hand the trusted party. If the adversary is passive then even corrupted parties hand T values according to the protocol. Next the parties are handed back their respective outputs: P_i gets $f(k, x_1^f..x_n^f, r_f)_i$, where r_f is the random input to f. Fresh randomness is used in each ideal evaluation call.

Let $\mathrm{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}}(k, \vec{x}, z)$ denote the random variable describing the output of the computation in the $(f_1, \ldots, f_m)$-hybrid model with protocol π, adversary $\mathcal{A}$, security parameter k, inputs $\vec{x}$ and auxiliary input z for the adversary, analogously to the definition of $\mathrm{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, z)$ in Section 4.1. (We stress that here π is a hybrid of a real-life protocol with ideal evaluation calls to T.) Let $\mathrm{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}}$ denote the distribution ensemble $\{\mathrm{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}}(k, \vec{x}, z)\}_{k\in\mathrm{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$.

Replacing an ideal evaluation call with a subroutine call. Next we describe the 'mechanics' of replacing an ideal-evaluation call of protocol π at round l with an invocation of an n-party protocol ρ. This is done in a straightforward way. That is, the description of π for round l is modified as follows. (Other rounds remain unaffected.)

1. At the onset of round l each party P_i saves its internal state (relevant to protocol π) on a special tape. Let σ_i denote this state.

2. The call to the trusted party T is replaced with an invocation of P_i's code for protocol ρ. Party P_i's input and random input for ρ are determined as follows. The input x_i^ρ is set to the value that P_i was to hand the trusted party T at round l, according to protocol π. The random input r_i^ρ is uniformly chosen in the appropriate domain.

3. Once P_i completes the execution of protocol ρ with local output v_i^ρ, it resumes the execution of protocol π for round l, starting from state σ_i, with the exception that the value to be received from T is set to v_i^ρ.

[12] We remark that these restrictions can be "circumvented" in a number of ways. For instance we can imagine that at each other round the parties make an ideal evaluation call to a "universal function", U, defined as follows. Each party P_i hands the trusted party a description of an n-party function f and an input x_i. If a majority of the parties agree on f then P_i is handed $f(\vec{x})_i$; otherwise a null value is returned. This convention allows us to apply the composition theorems to protocols where the parties decide in an adaptive way (say, using some agreement protocol) on the number of ideal evaluation calls and on the function to be evaluated at different calls.

Let $\pi^{\rho_1 \cdots \rho_m}$ denote protocol π (originally designed for the $(f_1 \ldots f_m)$-hybrid model) where each ideal evaluation call to f_i is replaced by a subroutine call to protocol ρ_i.

It is stressed that no uncorrupted party resumes execution of protocol π before the current execution of protocol ρ_i is completed. Furthermore, we assume that all the uncorrupted parties terminate each execution of ρ_i at the same round. Otherwise, some parties may resume executing the calling protocol while others still execute the subroutine protocol, and the non-concurrency condition is violated.[13]

Theorem 5, stated below, takes a somewhat different approach to the composition operation than the informal theorem made in the Introduction. It does not require any security properties from protocol π. Instead, it essentially states that the "input-output functionality" of *any* protocol π in the hybrid model is successfully "emulated" by $\pi^{\rho_1,\ldots,\rho_m}$ in the real-life model. On top of being somewhat more straightforward, this more general statement is relevant even in cases where π performs a task other than secure function evaluation.

Theorem 5 (non-adaptive modular composition: General statement) *Let $t < n$, let $m \in \mathbf{N}$, and let $f_1, \ldots, f_m$ be n-party functions. Let π be an n-party protocol in the $(f_1, \ldots, f_m)$-hybrid model where no more than one ideal evaluation call is made at each round, let $\rho_1, \ldots, \rho_m$ be n-party protocols where ρ_i non-adaptively t-securely (resp., t-privately) evaluates f_i, and let $\pi^{\rho_1,\ldots,\rho_m}$ be the composed protocol described above. Then, for any non-adaptive t-limited active (resp., passive) real-life adversary $\mathcal{A}$ there exists a non-adaptive active (resp., passive) adversary $\mathcal{A}_\pi$ in the $(f_1, \ldots, f_m)$-hybrid model, whose running time is polynomial in the running time of $\mathcal{A}$, and such that*

$$\mathrm{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}_\pi} \stackrel{\mathrm{d}}{=} \mathrm{EXEC}_{\pi^{\rho_1,\ldots,\rho_m},\mathcal{A}}. \tag{2}$$

For completeness, we also rigorously state the informal goal stated in the Introduction. For that, we first define protocols for securely evaluating a function g in the $(f_1, \ldots, f_m)$-hybrid model. This is done via the usual comparison to the ideal process for g:

Definition 6 *Let $f_1, \ldots, f_m, g$ be n-party functions and let π be a protocol for n parties in the $(f_1, \ldots, f_m)$-hybrid model. We say that π* non-adaptively t-securely evaluates g in the $(f_1, \ldots, f_m)$-hybrid model *if for any non-adaptive t-limited adversary $\mathcal{A}$ (in the $(f_1, \ldots, f_m)$-hybrid model) there exists a non-adaptive ideal-process adversary $\mathcal{S}$ whose running time is polynomial in the running time of $\mathcal{A}$, and such that*

$$\mathrm{IDEAL}_{g,\mathcal{S}} \stackrel{\mathrm{d}}{=} \mathrm{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}}. \tag{3}$$

If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries then we say that π non-adaptively t-privately evaluates g in the $(f_1, \ldots, f_m)$-hybrid model.

Corollary 7 (non-adaptive modular composition: Secure function evaluation) *Let $t < n$, let $m \in \mathbf{N}$, and let $f_1, \ldots, f_m, g$ be n-party functions. Let π be an n-party protocol that non-adaptively t-securely (resp., t-privately) evaluates g in the $(f_1, \ldots, f_m)$-hybrid model where no more*

[13] Consider, for instance, the following example. Parties A, B, C wish to evaluate the following function, g: C should output the input of B; B should output the input of A; A should have empty output. Assume a hybrid model with ideal access to a function f where C outputs the input of B. A protocol π for evaluating g in this hybrid model instructs parties A, B and C to first ideally evaluate f. Next party A is instructed to send B its input. It is easy to see that π securely evaluates g in the f-hybrid model. Let ρ be a protocol that securely evaluates f. Protocol ρ takes several rounds to complete, but party A completes ρ after the first round.

Now, assume that A sends its input to B as soon as it is done with the execution of ρ (and, in particular, before B and C have completed the execution of ρ). In this case, a corrupted B may be able to influence the output of C in ways that depend on A's input. This would make protocol π^ρ insecure, although both π and ρ are secure.

than one ideal evaluation call is made at each round, and let $\rho_1, \ldots, \rho_m$ be n-party protocols such that ρ_i non-adaptively t-securely (resp., t-privately) evaluates f_i. Then the protocol $\pi^{\rho_1,\ldots,\rho_m}$ non-adaptively t-securely (resp., t-privately) evaluates g.

Proof: Let $\mathcal{A}$ be a (non-adaptive) t-limited real-life adversary that interacts with parties running $\pi^{\rho_1,\ldots,\rho_m}$. Theorem 5 guarantees that there exists an adversary $\mathcal{A}_\pi$ in the $(f_1, \ldots, f_m)$-hybrid model such that $\mathrm{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}_\pi} \overset{\mathrm{d}}{=} \mathrm{EXEC}_{\pi^{\rho_1,\ldots,\rho_m},\mathcal{A}}$. The security of π in the $(f_1, \ldots, f_m)$-hybrid model guarantees that there exists an ideal model adversary (a "simulator") $\mathcal{S}$ such that $\mathrm{IDEAL}_{g,\mathcal{S}} \overset{\mathrm{d}}{=} \mathrm{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}_\pi}$. The corollary follows by combining the two equalities. □

4.4 Proof of Theorem 5

We prove the theorem only for the case of active adversaries (i.e., t-security). The case of passive adversaries (i.e., t-privacy) can be obtained by appropriately degenerating the current proof.

In addition, we first treat only the case where $m = 1$ and the trusted party T is called only once. The case of multiple functions and multiple (but non-concurrent) calls to T is a straightforward extension, and is treated at the end of the proof.

Section 4.4.1 contains an outline of the proof. The body of the proof is in Section 4.4.2. Section 4.4.3 contains some extensions of the proof (and of the theorem).

4.4.1 Proof outline

Let f be an n-party function, let π be an n-party protocol in the f-hybrid model, let ρ be a protocol that t-securely evaluates f, let π^ρ be the composed protocol. Let $\mathcal{A}$ be a (non-adaptive) real-life adversary that interacts with parties running π^ρ. We wish to construct an adversary $\mathcal{A}_\pi$ in the f-hybrid model that 'simulates' the behavior of $\mathcal{A}$. That is, $\mathcal{A}_\pi$ should satisfy

$$\mathrm{EXEC}_{\pi^\rho,\mathcal{A}} \overset{\mathrm{d}}{=} \mathrm{EXEC}^{f}_{\pi,\mathcal{A}_\pi}. \tag{4}$$

Our plan for carrying out this proof proceeds as follows.

1. We construct out of $\mathcal{A}$ a real-life adversary, denoted $\mathcal{A}_\rho$, that operates against protocol ρ as a stand-alone protocol. The security of ρ guarantees that $\mathcal{A}_\rho$ has a simulator (ie, an ideal-process adversary), $\mathcal{S}_\rho$, such that $\mathrm{EXEC}_{\rho,\mathcal{A}_\rho} \overset{\mathrm{d}}{=} \mathrm{IDEAL}_{f,\mathcal{S}_\rho}$.

2. Out of $\mathcal{A}$ and $\mathcal{S}_\rho$ we construct an adversary, $\mathcal{A}_\pi$, that operates against protocol π as a stand-alone protocol in the f-hybrid model. We then show that $\mathcal{A}_\pi$ satisfies (4).

Let us sketch the above steps. In a way, $\mathcal{A}_\rho$ represents the "segment" of $\mathcal{A}$ that interacts with protocol ρ. That is, $\mathcal{A}_\rho$ starts with a set C of corrupted parties, the inputs of the parties in C, and an auxiliary input. It expects its auxiliary input to describe an internal state of $\mathcal{A}$, controlling the parties in C, and after interacting with parties running protocol π^ρ up to the round, l_ρ, where ρ is invoked. (If the auxiliary input is improper then $\mathcal{A}_\rho$ halts.) Next, $\mathcal{A}_\rho$ interacts with its network by simulating a run of $\mathcal{A}$ from the given state, and following $\mathcal{A}$'s instructions. At the end of its interaction with parties running ρ, adversary $\mathcal{A}_\rho$ outputs the current state of the simulated $\mathcal{A}$.

Adversary $\mathcal{A}_\pi$ represents the "segment" of $\mathcal{A}$ that interacts with protocol π, where the interaction with of $\mathcal{A}$ with ρ is replaced with an interaction with $\mathcal{S}_\rho$. That is, $\mathcal{A}_\pi$ starts by invoking a copy of $\mathcal{A}$ and following $\mathcal{A}$'s instructions, up to round l_ρ. At this point, $\mathcal{A}$ expects to interact with parties running ρ, whereas $\mathcal{A}_\pi$ interacts with parties that invoke a trusted party for ideal evaluation of f. To continue the execution of $\mathcal{A}$, adversary $\mathcal{A}_\pi$ runs $\mathcal{S}_\rho$. For this purpose, $\mathcal{S}_\rho$ is given auxiliary input that describes the *current* state of $\mathcal{A}$ at round l_ρ. The information from $\mathcal{S}_\rho$'s trusted party is emulated by $\mathcal{A}_\pi$, using $\mathcal{A}_\pi$'s own trusted party for f. Recall that the output of $\mathcal{S}_\rho$ is a (simulated) internal state of $\mathcal{A}$ at the completion of protocol ρ. Once protocol ρ completes its execution and the parties return to running π, adversary $\mathcal{A}_\pi$ returns to running $\mathcal{A}$ (starting from the state in $\mathcal{S}_\rho$'s output) and follows the instructions of $\mathcal{A}$. When $\mathcal{A}$ terminates, $\mathcal{A}_\pi$ outputs whatever $\mathcal{A}$ outputs.

Let us address one detail regarding the construction (among the many details that were left out in this sketch). When adversary $\mathcal{A}_\pi$ runs $\mathcal{S}_\rho$, the latter expects to see the inputs of the corrupted parties to protocol ρ; however $\mathcal{A}_\pi$ does not know these values. In fact, these values may not even be defined in the execution of $\mathcal{A}$ with π^ρ. The answer to this apparent difficulty is simple: it does not matter which values $\mathcal{A}_\pi$ hands $\mathcal{S}_\rho$ as the inputs of the corrupted parties. The simulation is valid even if these inputs are set to some arbitrary values (say, the value 0). Intuitively, the reason is that we construct $\mathcal{A}_\rho$ in such a way that it does not 'look at' these input values at all. Thus the output of $\mathcal{A}_\rho$ (and consequently also the output of $\mathcal{S}_\rho$) is independent of these arbitrary input values.

4.4.2 A detailed proof

Let $\mathcal{A}$ be an adversary (interacting with parties running π^ρ). First we present the constructions of adversaries $\mathcal{A}_\rho$ and $\mathcal{A}_\pi$. Next we analyze $\mathcal{A}_\pi$, showing (4).

Some inevitable terminology. An execution of a protocol (either in the real-life or in the f-hybrid model) is the process of running the protocol with a given adversary on given inputs, random inputs, and auxiliary input for the adversary. (In the f-hybrid model an execution is determined also by the random choices of the trusted party for f.) The internal state (or, configuration) of an uncorrupted party at some round of an execution consists of the contents of all tapes of this party, the head position and the control state, taken at the end of this round. In particular, the internal state includes all the messages sent to this party at this round. We assume that the internal state includes the *entire* random input of the party for the computation, including the yet-unused parts. The internal state of the adversary is defined similarly. The global state of the system at some round of an execution is the concatenation of the internal states of the parties and the adversary at this round.

Let $\mathrm{IS}_{\pi,\mathcal{A}}(l,k,\vec{x},z,\vec{r})_0$ denote the internal state at round l of adversary $\mathcal{A}$ with auxiliary input z and when interacting with parties running protocol π on input $\vec{x} = x_1 \ldots x_n$, random input $\vec{r} = r_0 \ldots r_n$ and with security parameter k, as described above (r_0 for $\mathcal{A}$, x_i and r_i for party P_i). Let $\mathrm{IS}_{\pi,\mathcal{A}}(l,k,\vec{x},z,\vec{r})_i$ denote the internal state of party P_i at round l of this execution. (If P_i is corrupted then $\mathrm{IS}_{\pi,\mathcal{A}}(l,k,\vec{x},z,\vec{r})_i = -$.) Let

$$\mathrm{GS}_{\pi,\mathcal{A}}(l,k,\vec{x},z,\vec{r}) = \mathrm{IS}_{\pi,\mathcal{A}}(l,k,\vec{x},z,\vec{r})_0, \mathrm{IS}_{\pi,\mathcal{A}}(l,k,\vec{x},z,\vec{r})_1, \ldots, \mathrm{IS}_{\pi,\mathcal{A}}(l,k,\vec{x},z,\vec{r})_n.$$

Let $\mathrm{GS}_{\pi,\mathcal{A}}(l,k,\vec{x},z)$ denote the probability distribution of $\mathrm{GS}_{\pi,\mathcal{A}}(l,k,\vec{x},z,\vec{r})$ where $\vec{r}$ is uniformly chosen.

Note that the global state of the system at some round of an execution uniquely determines the continuation of the execution from this round until the completion of the protocol. In particular, the global output of the system is uniquely determined given the global state (at any round).

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

"Running adversary $\mathcal{A}$ from internal state z" means simulating a run of $\mathcal{A}$ starting at the internal state described in z. Recall that z contains all the information needed for the simulation; in particular, it contains all the necessary randomness.

Construction of $\mathcal{A}_\rho$. The construction follows the outline described above. More specifically, adversary $\mathcal{A}_\rho$ proceeds as described in Figure 2, given adversary $\mathcal{A}$.

Adversary $\mathcal{A}_\rho$

Adversary $\mathcal{A}_\rho$, interacting with parties $P_1, ..., P_n$ running protocol ρ, starts with a value k for the security parameter, a set C of corrupted parties, inputs and random inputs for the parties in C, and auxiliary input z^ρ. Next, do:

1. Ignore the input values of the corrupted parties.
2. Let l_ρ be the round where protocol π^ρ starts running protocol ρ. (I.e., this is the round where π calls the trusted party). Verify that the auxiliary input, z^ρ, is a valid internal state of $\mathcal{A}$, controlling the parties in C, at round $l_\rho \perp 1$. If z^ρ is not valid, then halt with no output. Else:
   (a) Run $\mathcal{A}$ from internal state z^ρ. Let $P'_1 ... P'_n$ denote the (imaginary) set of parties with which $\mathcal{A}$ interacts.
   (b) Whenever some uncorrupted party P_i (running ρ) sends a message m to a corrupted party P_j, $\mathcal{A}_\rho$ lets the simulated $\mathcal{A}$ see message m sent from party P'_i (running π^ρ) to party P'_j.
   (c) Whenever $\mathcal{A}$ instructs some corrupted party P'_j to send a message m to an uncorrupted party P'_i, adversary $\mathcal{A}_\rho$ instructs party P_j to send message m to party P_i.
3. Once $\mathcal{A}$ halts, $\mathcal{A}_\rho$ outputs the current internal state of $\mathcal{A}$ and halts.

Figure 2: Description of Adversary $\mathcal{A}_\rho$ in the non-adaptive model

It now follows from the security of protocol ρ that there exists an ideal-process adversary $\mathcal{S}_\rho$ such that $\text{IDEAL}_{f,\mathcal{S}_\rho} \overset{\text{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho}$. Note that $\mathcal{A}_\rho$ is deterministic, since all of the randomness used by $\mathcal{A}$ is provided in the auxiliary input z^ρ. Yet, the simulator $\mathcal{S}_\rho$ is (inherently) probabilistic, since it should generate a distribution ensemble that is equal to $\text{EXEC}_{\rho,\mathcal{A}_\rho}$. In particular, it should mimic the randomness used by the uncorrupted parties running ρ.

We observe that the special structure of $\mathcal{A}_\rho$ implies that $\mathcal{S}_\rho$ has an additional property, described as follows. Recall that $\mathcal{A}_\rho$ ignores the inputs of the corrupted parties, in the sense that its actions and output do not not depend on these input values. In particular, the copy of $\mathcal{A}$ run by $\mathcal{A}_\rho$ is not affected by these values. Therefore, the distribution of the output of $\mathcal{A}_\rho$, as well as the global output of the system after running ρ with $\mathcal{A}_\rho$, remains unchanged if we set the input values of the corrupted parties to 0. Consequently, the distribution of the global output of the ideal process for evaluating f with $\mathcal{S}_\rho$ has the same property. We formalize this discussion as follows. Given an input vector $\vec{x}^\rho$, let $\vec{x}^\rho|_0$ denote the vector obtained by replacing all the inputs of the corrupted parties with 0. Then, we have:

Claim 8 *For any value of the security parameter k, any input vector $\vec{x}^\rho$ and auxiliary input z^ρ we have*

$$\text{IDEAL}_{f,\mathcal{S}_\rho}(k,\vec{x}^\rho,z^\rho) \overset{\text{d}}{=} \text{IDEAL}_{f,\mathcal{S}_\rho}(k,\vec{x}^\rho|_0,z^\rho)$$

Proof: We have argued above that $\text{EXEC}_{\rho,\mathcal{A}_\rho}(k,\vec{x}^\rho,z^\rho) \overset{\text{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho}(k,\vec{x}^\rho|_0,z^\rho)$. However, $\text{IDEAL}_{f,\mathcal{S}_\rho}(k,\vec{x}^\rho,z^\rho) \overset{\text{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho}(k,\vec{x}^\rho,z^\rho)$, and $\text{IDEAL}_{f,\mathcal{S}_\rho}(k,\vec{x}^\rho|_0,z^\rho) \overset{\text{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho}(k,\vec{x}^\rho|_0,z^\rho)$. The claim follows. □

Construction of $\mathcal{A}_\pi$. Adversary $\mathcal{A}_\pi$ follows the outline described in Section 4.4.1. More specifically, it proceeds as described in Figure 3.

Adversary $\mathcal{A}_\pi$

Adversary $\mathcal{A}_\pi$, interacting with parties $P_1,\ldots,P_n$ running protocol π and given access to a trusted party T for evaluating f, starts with a value k for the security parameter, a set C of corrupted parties, inputs $\vec{x}_C$ and random inputs $\vec{r}_C$ for the parties in C, and auxiliary input z. Next, do:

1. Invoke $\mathcal{A}$ on $C,\vec{x}_C,\vec{r}_C,z$ and follow the instructions of $\mathcal{A}$ up to round $l_\rho \perp 1$. (Recall that so far π and π^ρ are identical.) In addition, keep another piece of the random input 'on the side'. This piece, denoted r^ρ, is used below.
2. At the onset of round l_ρ, $\mathcal{A}$ expects to start interacting with parties running protocol ρ (as subroutine), whereas parties $P_1,\ldots,P_n$ call a trusted party for ideal evaluation of function f. In order to continue the run of $\mathcal{A}$, invoke simulator $\mathcal{S}_\rho$ as follows.
   (a) $\mathcal{S}_\rho$ is given the set C of corrupted parties. The inputs of these parties are set to 0, and their random input are set to r^ρ. (Recall that the inputs of the corrupted parties do not affect the distribution of the global output of evaluating f with $\mathcal{S}_\rho$.) The auxiliary input z^ρ for $\mathcal{S}_\rho$ is set to the current internal state of $\mathcal{A}$.
   (b) When $\mathcal{S}_\rho$ hands its trusted party the inputs of the corrupted parties and asks for the evaluated values of f, invoke the trusted party, T, with the same input values for the corrupted parties, and hand the value provided by the trusted party back to $\mathcal{S}_\rho$.
3. Recall that the output of $\mathcal{S}_\rho$ is an internal state of $\mathcal{A}$ at the end of the execution of ρ. Once this output, denoted v, is generated, run $\mathcal{A}$ from internal state v, and return to following $\mathcal{A}$'s instructions until the completion of protocol π.
4. Once protocol π is completed, output whatever $\mathcal{A}$ outputs and halt.

Figure 3: Description of adversary $\mathcal{A}_\pi$ in the non-adaptive model.

Analysis of $\mathcal{A}_\pi$. It is evident that the running time of $\mathcal{A}_\pi$ is linear in the running time of $\mathcal{A}$, plus the running time of $\mathcal{S}_\rho$, plus the running time of π^ρ. Fix an input vector $\vec{x}$ and auxiliary input z for the parties and adversary, as well as some value of the security parameter k. (In particular, the set C of corrupted parties is now fixed.) Steps I-III below demonstrate that:

$$\text{EXEC}_{\pi^\rho,\mathcal{A}}(k,\vec{x},z) \overset{\text{d}}{=} \text{EXEC}^f_{\pi,\mathcal{A}_\pi}(k,\vec{x},z), \tag{5}$$

which establishes the theorem for the case of a single ideal evaluation call. (In Equation (5) and for the rest of the proof the symbol $\overset{\text{d}}{=}$ is used to denote equality of *distributions,* not ensembles.)

Let us set some additional notation. Recall that l_ρ is the round where protocol π makes the ideal evaluation call, and protocol π^ρ invokes ρ. Given vectors $\vec{r}^\pi = r_0^\pi, ..., r_n^\pi$ and $\vec{r}^\rho = r_0^\rho, ..., r_n^\rho$ (where $\vec{r}^\pi$ is interpreted as random input for the execution of π^ρ except for the execution of ρ, and $\vec{r}^\rho$ is interpreted as random input for the execution of ρ), let $\vec{r}^{\pi,\rho} = r_0^{\pi,\rho}, ..., r_n^{\pi,\rho}$ denote the combination of $\vec{r}^\pi$ and $\vec{r}^\rho$ to a full random-input vector for the execution of π^ρ. (That is, party P_i uses r_i^ρ for the execution of ρ and r_i^π for the execution of π, and the adversary uses r_0^ρ during the execution of ρ and r_0^π at other rounds.) Similarly, given $r^\pi = r_0^\pi, ..., r_n^\pi$ and $\vec{r}^f$, where $\vec{r}^\pi$ is as above and $\vec{r}^f$ is interpreted as a random vector for round l_ρ in the f-hybrid model (that is, $\vec{r}^f = r_0^f, r_1^f$ where r_0^f is the random input for the adversary for this round and r_1^f is the random input for the trusted party for f), let $\vec{r}^{\pi,f}$ denote the combination of $\vec{r}^\pi$ and $\vec{r}^f$ to a full random-input vector for the execution of π in the f-hybrid model.

Step I. Until round $l_\rho - 1$, protocols π and π^ρ "behave the same". That is, fix some value $\vec{r}^\pi$ as the random input for the system. We have:

$$\mathrm{GS}_{\pi^\rho,\mathcal{A}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi) = \mathrm{GS}_{\pi,\mathcal{A}_\pi}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi). \tag{6}$$

Step II. We show that the global state in the hybrid model at the end of round l_ρ is distributed identically to the global state in the real-life model at the round where protocol ρ returns. This is done in three sub-steps, as follows. (Recall that a value $\vec{r}^\pi$ was fixed in Step I.)

1. We first assert that the parameters set in the hybrid model for the ideal evaluation of f are identical to the parameters set in the real-life model for the invocation of ρ. that is, let $\vec{x}^\rho = x_1^\rho, ..., x_n^\rho$, where x_i^ρ is determined as follows. If P_i is uncorrupted then x_1^ρ is the input value of P_i for protocol ρ, as determined in $\mathrm{GS}_{\pi^\rho,\mathcal{A}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)$. If P_i is corrupted then $x_i^\rho = 0$. Let Let z^ρ denote the internal state of $\mathcal{A}$ at round $l_\rho - 1$ in this execution. Similarly, let x_i^f denote the value that party P_i hands the trusted party for f, as determined in $\mathrm{GS}_{\pi,\mathcal{A}_\pi}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)$, let $\vec{x}^f = x_1^f, ..., x_n^f$, and let z^f denote the internal state of $\mathcal{A}$ (within $\mathcal{A}_\pi$'s code) at round $l_\rho - 1$ of this execution. Then, it follows from Equation (6) that $\vec{x}^\rho = \vec{x}^f|_0$ and $z^\rho = z^f$.

2. Next we assert that the global output of the execution of ρ, that is implicit in the run of π^ρ with adversary $\mathcal{A}$, is distributed identically to the global output of the ideal evaluation of f that is implicit in round l_ρ of a run of π in the hybrid model. That is, from the security of ρ, from Step II.1 and from Claim 8, we have that:

$$\mathrm{EXEC}_{\rho,\mathcal{A}_\rho}(k, \vec{x}^\rho, z^\rho) \overset{\mathrm{d}}{=} \mathrm{IDEAL}_{f,\mathcal{S}_\rho}(k, \vec{x}^\rho, z^\rho) = \mathrm{IDEAL}_{f,\mathcal{S}_\rho}(k, \vec{x}^f|_0, z^f) \overset{\mathrm{d}}{=} \mathrm{IDEAL}_{f,\mathcal{S}_\rho}(k, \vec{x}^f, z^f). \tag{7}$$

3. Finally we show that the global state in the hybrid model at the at the end of round l_ρ is distributed identically to the global state in the real-life model when protocol ρ returns. That is, let l_π denote the round where the call to protocol ρ returns (within protocol π^ρ). Then, it follows from the definition of π^ρ and the constructions of $\mathcal{A}_\rho$ and $\mathcal{A}_\pi$ that:

   (a) Let $\vec{r}^\rho$ be some random-input vector for protocol ρ. Then, $\mathrm{GS}_{\pi^\rho,\mathcal{A}}(l_\pi, k, \vec{x}, z, \vec{r}^{\pi,\rho})$ is obtained from $\mathrm{GS}_{\pi^\rho,\mathcal{A}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)$ and $\mathrm{EXEC}_{\rho,\mathcal{A}_\rho}(k, \vec{x}^\rho, z^\rho, \vec{r}^\rho)$ via a (simple, deterministic) process, denoted $\mathcal{C}$. (Essentially, process $\mathcal{C}$ combines and updates the internal states of the adversary and the parties. More precisely, this process first modifies each internal state $\mathrm{IS}_{\pi^\rho,\mathcal{A}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)_i$ by adding $\mathrm{EXEC}_{\rho,\mathcal{A}_\rho}(k, \vec{x}^\rho, z^\rho, \vec{r}^\rho)_i$ in the appropriate place. Next it outputs the internal state of $\mathcal{A}$ as it appears in $\mathcal{A}_\rho$'s output in

$\text{EXEC}_{\rho,\mathcal{A}_\rho}(k,\vec{x}^\rho,z^\rho,\vec{r}^\rho)$, and appends to it the modified internal states of the uncorrupted parties.)

(b) Given some random input vector $\vec{r}^f$ for the ideal process for evaluating f, the global state $\text{GS}_{\pi,\mathcal{A}_\pi}(l_\rho,k,\vec{x},z,\vec{r}^{\pi,f})$ is obtained from $\text{GS}_{\pi,\mathcal{A}_\pi}(l_\rho-1,k,\vec{x},z,\vec{r}^\pi)$ and $\text{IDEAL}_{f,\mathcal{S}_\rho}(k,\vec{x}^f,z^f,\vec{r}^f)$ via *the same process, $\mathcal{C}$,* as in the real-life model.

It follows that for any value of $\vec{r}^\pi$, and for vectors $\vec{r}^\rho$ and $\vec{r}^f$ that are uniformly chosen in their respective domains, we have $\text{GS}_{\pi^\rho,\mathcal{A}}(l_\pi,k,\vec{x},z,\vec{r}^{\pi,\rho}) \stackrel{\text{d}}{=} \text{GS}_{\pi,\mathcal{A}_\pi}(l_\rho,k,\vec{x},z,\vec{r}^{\pi,f})$. Now, let $\vec{r}^\pi$ be randomly chosen in its domain. It follows that:

$$\text{GS}_{\pi^\rho,\mathcal{A}}(l_\pi,k,\vec{x},z) \stackrel{\text{d}}{=} \text{GS}_{\pi,\mathcal{A}_\pi}(l_\rho,k,\vec{x},z). \tag{8}$$

Step III. We assert Equation (5). From the resumption of protocol π until its conclusion, adversary $\mathcal{A}_\pi$ returns to following the instructions of $\mathcal{A}$. Consequently, the distributions $\text{EXEC}_{\pi^\rho,\mathcal{A}}(k,\vec{x},z)$ and $\text{EXEC}^f_{\pi,\mathcal{A}_\pi}(k,\vec{x},z)$ are obtained by applying the same process to the corresponding sides of (8).

This completes the proof for the case of a single ideal evaluation call.

On multiple ideal evaluation calls. The case of multiple ideal evaluation calls is a straightforward generalization of the case of a single call. We sketch the main points of difference:

1. An adversary $\mathcal{A}_{\rho_i}$ is constructed for each protocol ρ_i. All the $\mathcal{A}_{\rho_i}$'s are identical to adversary $\mathcal{A}_\rho$ described above, with the exception that protocol ρ is replaced by ρ_i. (If $\rho_i=\rho_j$ for some i,j then $\mathcal{A}_{\rho_i}=\mathcal{A}_{\rho_j}$.)

2. Construct an adversary $\tilde{\mathcal{A}}_\pi$ that is identical to $\mathcal{A}_\pi$ described above, with the exception that at each round where π instructs the parties to ideally evaluate f_i, adversary $\tilde{\mathcal{A}}_\pi$ runs a copy of $\mathcal{S}_{\rho_i}$ in the same way as $\mathcal{A}_\pi$ runs $\mathcal{S}_\rho$. The auxiliary input of $\mathcal{S}_{\rho_i}$ is set to the current internal state of the simulated $\mathcal{A}$ within $\mathcal{A}_\pi$. (Note that there may be several invocations of the same simulator $\mathcal{S}_{\rho_i}$, where each invocation corresponds to a different ideal evaluation call to f_i. These invocations will have different auxiliary inputs. Also, a separate piece of $\tilde{\mathcal{A}}_\pi$'s random input is used for each invocation of some $\mathcal{S}_{\rho_i}$.)

3. As in the case of a single ideal evaluation call, it is evident that the running time of $\tilde{\mathcal{A}}_\pi$ is linear in the running time of $\mathcal{A}$, plus the sum of the running times of all the invocations of $\mathcal{S}_{\rho_1},\ldots,\mathcal{S}_{\rho_m}$, plus the running time of $\pi^{\rho_1,\ldots,\rho_m}$. Showing that $\text{EXEC}_{\pi^{\rho_1,\ldots,\rho_m},\mathcal{A}}(k,\vec{x},z) \stackrel{\text{d}}{=} \text{EXEC}^{f_1,\ldots,f_m}_{\pi,\tilde{\mathcal{A}}_\pi}(k,\vec{x},z)$ is done in several steps, as follows. Let $l^{(j)}$ denote the round in which protocol π makes the jth ideal evaluation call in the hybrid model. The argument of Step I above demonstrates that the global states at round $l^{(j)}-1$ are identical in the two executions. Now, for each $j\geq 1$ proceed in two steps:

   (a) Apply the argument of Step II to establish that the global state in the hybrid model at the end of round $l^{(j)}$ is distributed identically to the global state in the real-life model at the round where the jth subroutine call (to some ρ_i) returns.

   (b) Apply the argument of Step III to establish that the global state in the hybrid model at round $l^{(j+1)}-1$ is distributed identically to the global state in the real-life model at the round where the $(j+1)$th subroutine call is made. If the execution is completed without making the $(j+1)$th subroutine call then we have established that $\text{EXEC}_{\pi^{\rho_1,\ldots,\rho_m},\mathcal{A}}(k,\vec{x},z) \stackrel{\text{d}}{=} \text{EXEC}^{f_1,\ldots,f_m}_{\pi,\tilde{\mathcal{A}}_\pi}(k,\vec{x},z)$, as required.

4.4.3 Extensions

On the propagation of statistical distance. Somewhat relaxed versions of Definitions 4 and 6 allows the two sides of (1) and of (3) to be statistically indistinguishable, rather than equally distributed. We note that the composition theorem holds in this case as well. That is:

1. Theorem 5 holds with the exception that the two sides of (2) are statistically indistinguishable. More specifically, in the case of a single ideal evaluation call, if protocol ρ achieves statistical distance δ_1 then the statistical distance between the two sides of (2) is at most δ_1. (The construction and analysis of $\mathcal{A}_\pi$ remain unchanged, with the exception that the two leftmost distributions in (7) have statistical distance δ_1.)

   In the case of multiple ideal evaluation calls the total statistical distance between the two sides of (2) is at most the sum of the statistical distances achieved by all the individual protocol invocations made by the composed protocol. That is, if protocol ρ_i achieves statistical distance δ_i, and is invoked v_i times, then the total statistical distance between the two sides of (2) is at most $\sum_{i=1}^{m} v_i \cdot \delta_i$.

2. Corollary 7 holds with the exception that the two sides of (1) are statistically indistinguishable. More specifically, in the case of a single ideal evaluation call, if protocol ρ achieves statistical distance δ_1 and protocol π achieves statistical distance δ_2 then protocol π^ρ achieves statistical distance $\delta_1 + \delta_2$.

   In the case of multiple ideal evaluation calls the statistical distance achieved by $\pi^{\rho_1,\ldots,\rho_m}$ is at most the sum of the statistical distances achieved by all the individual protocol invocations, plus the statistical distances achieved by π in the $(f_1, \ldots, f_m)$-hybrid model. That is, assume that protocol π achieves statistical distance δ in the hybrid model, and that protocol ρ_i achieves statistical distance δ_i, and is invoked v_i times. Then protocol $\pi^{\rho_1,\ldots,\rho_m}$ achieves statistical distance at most $\delta + \sum_{i=1}^{m} v_i \cdot \delta_i$.

On computational indistinguishability. The composition theorem holds also for the case where the two sides of (1), and also of (2), are only computationally indistinguishable. We defer the treatment of this case to Section 6.

On black-box simulation. A straightforward extension of the proof of Corollary 7 shows the following additional result. Assume that the security of protocol π in the hybrid model is proven via black-box simulation (see Remark 3, Section 4.2). Then the security of protocol π^ρ can also be proven via black-box simulation. Furthermore, if the simulator associated with π does not rewind the adversary, then the simulator associated with π^ρ does not rewind as well. Note that no additional requirements are made from protocol ρ. In particular, the security of protocol ρ need not be proven via black-box simulation.

Remark: The reader may notice that the fact that the communication links are ideally secure does not play a central role in the proof of Theorem 5. Indeed, the same proof technique (with trivial modifications) is valid in a setting where the adversary sees all the communication among the parties. See more details in Section 6.

5 Adaptive adversaries

This section defines secure protocols, presents and proves the composition theorem for the case of adaptive adversaries. Both the definition of adaptive security of protocols and the proof of the composition theorem in this case are considerably more complex than for the non-adaptive case. Furthermore, proving adaptive security of protocols is typically harder. We thus start with some motivation for this more complex model.

While adaptive security looks like a natural extension of non-adaptive security, a second look reveals some important differences between the two models and the security concerns they capture. Informally, the non-adaptive model captures scenarios where the parties do not trust each other, but believe that parties that are 'good' remain so throughout. There, the adversary is an *imaginary concept* that represents a collection of 'bad parties'. In contrast, the adaptive model captures scenarios where parties may become corrupted during the course of the computation — either on their own accord, or, more realistically, via an external "break-in". Here the adversary models an *actual entity* that takes active part in the computation. Indeed, external attackers who have the ability to adaptively "break-into" parties impose a viable security threat on existing systems and networks.

Non-adaptive security is implied by adaptive security (see Remark 1, Section 5.2). However, the converse does not hold. In particular, while the non-adaptive model captures many security concerns regarding cryptographic protocols, it fails to capture some important concerns that are addressed in the adaptive model. One such concern is the need to deal with the fact that an adversary may use the communication to decide which parties are worth corrupting. (See Remark 2 there.) Another such concern relates to the fact that the adversary may gain considerable advantage from seeing the internal data of parties upon corruption (or a "break-in"), after some computational steps have taken place. This means that data kept by the uncorrupted parties should never be regarded as safe, and the threat of this data being exposed should play an important part in the security analysis of a protocol. See Remark 3, Section 5.2.[14]

This section attempts to be as self-contained as possible, at the price of some repetition. Still, in cases where the text is very similar to the non-adaptive case with immediate modifications we only note the changes from the corresponding parts of Section 4.

Throughout this section we restrict the presentation to the secure channels setting. The computational setting is dealt with in the Section 6. Section 5.1 contains the definition of secure protocols. All the remarks made in Section 4.2 and in footnotes throughout Section 4 are relevant here as well, but are not repeated. In addition, Section 5.2 holds remarks specific to the adaptive case. Section 5.3 presents the composition theorem, to be proven in Section 5.4.

5.1 Definition of security: The adaptive case

As in the non-adaptive case, we develop the definitions for the cases of active and passive adversaries side by side, noting the differences throughout the presentation. We first describe the real-life model; next we describe the ideal process; finally the definition is presented, using essentially the same notion of emulation as in the non-adaptive case.

One obvious difference from the definition of non-adaptive security is that here the adversary

[14] Limiting the advantage gained by the adversary from exposing the secret data of parties is sometimes called forward secrecy in the literature. In the context of key exchange, for instance, forward secrecy refers to preventing an adversary from learning, upon corrupting a party, keys that are no longer in use [DOW92]. Indeed, the adaptive setting provides a framework for analyzing forward secrecy of protocols.

chooses the identities of the corrupted parties in an adaptive way; upon corruption, it sees the internal data of the corrupted party. (See more discussion on this point in the sequel.)

An additional, more 'technical' difference is the way in which the interaction between the outside environment and a single protocol execution is captured. In the non-adaptive case this interaction is captured by the parties' inputs and outputs, plus an auxiliary input z given to the adversary before the computation starts. There, this representation sufficed for proving the composition theorem. In the adaptive case there is an additional way in which the external environment interacts with a given protocol execution: whenever the adversary corrupts a party it sees the party's entire internal state, including the state for *all* the protocol executions which involve this party. This fact has two manifestations. Consider a protocol execution $\mathcal{E}$ that is part of a larger protocol, involving other protocol executions. First, when a party is corrupted during execution $\mathcal{E}$ the adversary sees the party's internal state also from other protocol executions, both completed and uncompleted ones. (Here information flows *into* execution $\mathcal{E}$ from the outside environment.) Second, when a party is corrupted in another protocol execution, the adversary sees the party's internal state relevant to execution $\mathcal{E}$. (Here information flows *from* execution $\mathcal{E}$ to the outside environment.) A particularly problematic case is that of corruptions that occur *after* execution $\mathcal{E}$ is completed.

To model this information flow, we introduce an additional entity, representing the external environment, to both the real-life model and the ideal process. This entity, called the environment and denoted $\mathcal{Z}$, is an interactive Turing machine that interacts with the adversary and the parties in a way described below. The notion of emulation is extended to include the environment.

The real-life model. Multiparty protocols are defined as in the non-adaptive case. That is, an n-party protocol π is a collection of n interactive, probabilistic algorithms, where the ith algorithm is run by the ith party, P_i. (Formally, each algorithm is an Interactive Turing machine, as defined in [GMR89].) Each P_i has input $x_i \in \{0,1\}^*$, random input $r_i \in \{0,1\}^*$, and the security parameter k. Informally, we envision each two parties as connected via a private communication channel. A more complete description of the communication among parties is presented below.

An adaptive real-life adversary $\mathcal{A}$ is a computationally unbounded interactive Turing machine that starts off with some random input. The environment is another computationally unbounded interactive Turing machine, denoted $\mathcal{Z}$, that starts off with input z and random input. At certain points during the computation the environment interacts with the parties and the adversary. These points and the type of interaction are specified below. An adversary is t-limited if it never corrupts more than t parties.

At the onset of the computation $\mathcal{A}$ receives some initial information from $\mathcal{Z}$. (This information corresponds to the auxiliary information seen by $\mathcal{A}$ in the non-adaptive case.) Next, the computation proceeds according to some given computational model. For concreteness, we specify the following (synchronous, with rushing) model of computation. The computation proceeds in rounds; each round proceeds in mini-rounds, as follows. Each mini-round starts by allowing $\mathcal{A}$ to *corrupt* parties one by one in an adaptive way, as long as at most t parties are corrupted altogether. (The behavior of the system upon corruption of a party is described below.) Next $\mathcal{A}$ chooses an uncorrupted party, P_i, that was not yet activated in this round and activates it. Upon activation, P_i receives the messages sent to it in the previous round, generates its messages for this round, and the next mini-round begins. $\mathcal{A}$ learns the messages sent by P_i to already corrupted parties. Once all the uncorrupted parties were activated, $\mathcal{A}$ generates the messages to be sent by the corrupted parties that were not yet activated in this round, and the next round begins.

Once a party is corrupted the party's input, random input, and the entire history of the messages sent and received by the party become known to $\mathcal{A}$. (The amount of information seen by the adversary upon corrupting a party is an important parameter of the definition. See discussion in Remark 4, Section 5.2.) In addition, $\mathcal{Z}$ learns the identity of the corrupted party, and hands some additional auxiliary information to $\mathcal{A}$. (Intuitively, this information represents the party's internal data from other protocols run by the newly corrupted party.[15]) From this point on $\mathcal{A}$ learns all the messages received by the party. If $\mathcal{A}$ is passive then the corrupted parties continue running protocol π. If $\mathcal{A}$ is active (Byzantine) then once a party becomes corrupted it follows the instructions of $\mathcal{A}$, regardless of protocol π.

At the end of the computation (say, at some pre-determined round) all parties locally generate their outputs. The uncorrupted parties output whatever is specified in the protocol. The corrupted parties output $-$. In addition, adversary $\mathcal{A}$ outputs some arbitrary function of its internal state. (Without loss of generality, we can imagine that the adversary's output consists of all the information seen in the execution. This includes the random input, the information received from the environment, the corrupted parties' internal data, and all the messages sent and received by the corrupted parties during the computation.)

Next, a "post-execution corruption process" begins. (This process models the information on the current execution, gathered by the environment by corrupting parties after the execution is completed.) First, $\mathcal{Z}$ learns the outputs of all the parties and of the adversary. Next $\mathcal{Z}$ and $\mathcal{A}$ interact in rounds, where in each round $\mathcal{Z}$ first generates a ‘corrupt P_i’ request (for some P_i), and hands this request to $\mathcal{A}$. Upon receipt of this request, $\mathcal{A}$ hands $\mathcal{Z}$ some arbitrary information. (Intuitively, this information is interpreted as P_i's internal data.) It is stressed that at most t parties are corrupted throughout, even if $\mathcal{Z}$ requests to corrupt more parties; in this case $\mathcal{A}$ ignores the requests of $\mathcal{Z}$. The interaction continues until $\mathcal{Z}$ halts, with some output. Without loss of generality, this output can be $\mathcal{Z}$'s entire view of its interaction with $\mathcal{A}$ and the parties. Finally, the global output is defined to be the output of $\mathcal{Z}$ (which, as said above, may include the outputs of all parties as well as of the adversary). See further discussion on the role of the environment $\mathcal{Z}$ in Remark 5, Section 5.2. The computational process in the real-life model is summarized in Figure 4.

We use the following notation. Let the global output $\textsc{exec}_{\pi,\mathcal{A},\mathcal{Z}}(k,\vec{x},z,\vec{r})$ denote $\mathcal{Z}$'s output on input z, random input r_Z and security parameter k, and after interacting with adversary $\mathcal{A}$ and parties running protocol π on inputs $\vec{x} = x_1 \dots x_n$, random input $\vec{r} = r_Z, r_0 \dots r_n$, and security parameter k as described above (r_0 for $\mathcal{A}$; x_i and r_i for party P_i). Let $\textsc{exec}_{\pi,\mathcal{A},\mathcal{Z}}(k,\vec{x},z)$ denote the random variable describing $\textsc{exec}_{\pi,\mathcal{A},\mathcal{Z}}(k,\vec{x},z,\vec{r})$ where $\vec{r}$ is uniformly chosen. Let $\textsc{exec}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble $\textsc{exec}_{\pi,\mathcal{A},\mathcal{Z}}(k,\vec{x},z)\}_{k\in\mathrm{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$. (The formalization of the global output $\textsc{exec}_{\pi,\mathcal{A},\mathcal{Z}}$ is different than in the non-adaptive case, in that here the global output contains *only* the output of the environment. We remark that the more complex formalization, where the global output contains the concatenation of the outputs of all parties and adversary, would yield an equivalent definition; this is so since the environment $\mathcal{Z}$ sees the outputs of all the parties and the adversary. We choose the current formalization for its simplicity.)

The ideal process. The ideal process is parameterized by the function to be evaluated. This is an n-party function $f : \mathbf{N} \times (\{0,1\}^*)^n \times \{0,1\}^* \to (\{0,1\}^*)^n$, as defined in Section 3. Each party P_i has input $x_i \in \{0,1\}^*$; no random input is needed. Recall that the parties wish to evaluate $f(k,\vec{x},r_f)_1, \dots, f(k,\vec{x},r_f)_n$, where $r_f \overset{\mathrm{R}}{\leftarrow} \{0,1\}^s$ and s is a value determined by the security

[15] For sake of simplicity, we do not restrict the way in which $\mathcal{Z}$ computes the data provided to the adversary upon corruption of a party. However, we note that a somewhat weaker definition where this data is fixed before the computation starts (but remains unknown to the adversary until the party is corrupted) is sufficient, both for capturing security and for the proof of the composition theorems.

Execution of an n-party protocol by parties $P_1 \ldots P_n$ with adversary $\mathcal{A}$ and environment $\mathcal{Z}$

1. (a) Each party P_i starts with the security parameter k, input x_i and random input r_i.
   (b) The adversary $\mathcal{A}$ starts with k and random input r_0. The environment $\mathcal{Z}$ starts with input z and random input r_Z.
2. Initialize the round number to $l \leftarrow 0$. $\mathcal{A}$ receives an initial message from $\mathcal{Z}$.
3. As long as there exists an uncorrupted party that did not halt, do:
   (a) As long as there exists an uncorrupted party that was not activated in this round, do:
      i. As long as $\mathcal{A}$ decides to corrupt more parties, do:
         A. $\mathcal{A}$ chooses a party P_i to corrupt. $\mathcal{Z}$ learns the identity of P_i.
         B. $\mathcal{A}$ receives P_i's input, random input, and all the messages that P_i received in this interaction. In addition, $\mathcal{A}$ receives a message from $\mathcal{Z}$.
      ii. $\mathcal{A}$ activates an uncorrupted party P_i. If $l > 1$ then P_i receives the messages $\{m_{j,i,l-1} | j \in [n]\}$ sent to it in the previous round. Next, P_i generates $\{m_{i,j,l} | j \in [n]\}$, where each $m_{i,j,l} \in \{0,1\}^*$ is a (possibly empty) message intended for party P_j at this round. The adversary $\mathcal{A}$ learns $\{m_{i,j,l} | P_j \text{ is corrupted}\}$.
   (b) $\mathcal{A}$ generates the messages $\{m_{i,j,l} | P_i \text{ is corrupted and } j \in [n]\}$.
   (c) $l \leftarrow l + 1$
4. Each uncorrupted party P_i, as well as $\mathcal{A}$, generates an output. $\mathcal{Z}$ learns all outputs.
5. As long as $\mathcal{Z}$ did not halt, do:
   (a) $\mathcal{Z}$ sends $\mathcal{A}$ a message, interpreted as "corrupt P_i" for some uncorrupted party P_i.
   (b) $\mathcal{A}$ may corrupt more parties, as in Step 3(a)i above.
   (c) $\mathcal{A}$ sends $\mathcal{Z}$ a message, interpreted as P_i's internal data.
6. $\mathcal{Z}$ halts with some output.

Figure 4: A summary of the adaptive real-life computation.

parameter, and P_i learns $f(k, \vec{x}, r_f)_i$. The model also involves an adaptive ideal-process-adversary $\mathcal{S}$, which is an interactive Turing machine that has random input r_0 and security parameter k, and an environment $\mathcal{Z}$ which is a computationally unbounded interactive Turing machine that starts with input z, random input r_Z and the security parameter.[16] In addition, there is an (incorruptible) trusted party, T. The ideal process proceeds as follows.

First corruption stage: First, as in the real-life model, $\mathcal{S}$ receives auxiliary information from $\mathcal{Z}$. Next, $\mathcal{S}$ proceeds in iterations, where in each iteration $\mathcal{S}$ may decide to corrupt some party, based on $\mathcal{S}$'s random input and the information gathered so far. Once a party is corrupted its input becomes known to $\mathcal{S}$. In addition, $\mathcal{Z}$ learns the identity of the corrupted party and hands some extra auxiliary information to $\mathcal{S}$. Let B denote the set of corrupted parties at

[16] There is no need to explicitly restrict the number of parties corrupted by $\mathcal{S}$. The definition of security (in particular, the fact that the identities of the corrupted parties appear in the global output) will guarantee that an ideal-model adversary $\mathcal{S}$ (emulating some real-life adversary $\mathcal{A}$) corrupts no more parties than $\mathcal{A}$ does. In fact, it will be guaranteed that the distribution ensembles describing the parties corrupted by $\mathcal{A}$ and by $\mathcal{S}$ are identical.

the end of this stage.

Computation stage: Once $\mathcal{S}$ completes the previous stage, the parties hand the following values to the trusted party T. The uncorrupted parties hand their inputs to the computation. The corrupted parties hand values chosen by $\mathcal{S}$, based on the information gathered so far. (If $\mathcal{S}$ is passive then even the corrupted parties hand their inputs to T.)

Let $\vec{b}$ be the $|B|$-vector of the inputs contributed by the corrupted parties, and let $\vec{y} = y_1, ..., y_n$ be the n-vector constructed from the input vector $\vec{x}$ by substituting the entries of the corrupted parties by the corresponding entries in $\vec{b}$. Then, T receives y_i from P_i. (If $\mathcal{S}$ is passive then $\vec{y} = \vec{x}$). Next, T chooses $r_f \overset{\text{R}}{\leftarrow} \mathcal{R}_f$, and hands each P_i the value $f(k, \vec{y}, r_f)_i$.

Second corruption stage: Upon learning the corrupted parties' outputs of the computation, $\mathcal{S}$ proceeds in another sequence of iterations, where in each iteration $\mathcal{S}$ may decide to corrupt some additional party, based on the information gathered so far. Upon corruption, $\mathcal{Z}$ learns the identity of the corrupted party, $\mathcal{S}$ sees the corrupted party's input *and output,* plus some additional information from $\mathcal{Z}$ as before.

Output: Each uncorrupted party P_i outputs $f(k, \vec{y}, r_f)_i$, and the corrupted parties output $-$. In addition, the adversary outputs some arbitrary function of the information gathered during the computation in the ideal process. All outputs become known to $\mathcal{Z}$.

Post-execution corruption: Once the outputs are generated, $\mathcal{S}$ engages in an interaction with $\mathcal{Z}$, similar to the interaction of $\mathcal{A}$ with $\mathcal{Z}$ in the real-life model. That is, $\mathcal{Z}$ and $\mathcal{S}$ proceed in rounds where in each round $\mathcal{Z}$ generates some `corrupt P_i´ request, and $\mathcal{S}$ generates some arbitrary answer based on its view of the computation so far. For this purpose, $\mathcal{S}$ may corrupt more parties as described in the second corruption stage. The interaction continues until $\mathcal{Z}$ halts with an arbitrary output.

Let $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(k, \vec{x}, z, \vec{r})$, where $\vec{r} = r_Z, r_0, r_f$, denote the output of environment $\mathcal{Z}$ on input z, random input r_Z and security parameter k, after interacting as described above with an ideal-process adversary $\mathcal{S}$ and with parties having input $\vec{x} = x_1 \ldots x_n$ and with a trusted party for evaluating f with random input r_f. Let $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(k, \vec{x}, z)$ denote the distribution of $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(k, \vec{x}, z, \vec{r})$ when $\vec{r}$ is uniformly distributed. Let $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}$ denote the distribution ensemble $\{\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(k, \vec{x}, z)\}_{k \in \mathrm{N}, \langle \vec{x}, z \rangle \in \{0,1\}^*}$.

Comparing computations in the two models. As in the non-adaptive case, we require that protocol π emulates the ideal process for evaluating f. Yet here the notion of emulation is slightly different. We require that for any real-life adversary $\mathcal{A}$ *and any environment* $\mathcal{Z}$ there should exist an ideal-process adversary $\mathcal{S}$, such that $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \overset{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. 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 environment and real-life adversary $\mathcal{A}$, there should exist an ideal-process adversary that successfully simulates $\mathcal{A}$ *in the presence of this specific environment.*" Furthermore, we require $\mathcal{S}$ to be polynomial in the complexity of $\mathcal{A}$, regardless of the complexity of $\mathcal{Z}$ (see Remark 1, Section 4.2).

Definition 9 (adaptive security in the secure channels setting) *Let f be an n-party function and let π be a protocol for n parties. We say that π* adaptively, t-securely evaluates *f if for* *any adaptive t-limited real-life adversary $\mathcal{A}$, and any environment $\mathcal{Z}$, there exists an adaptive ideal-process adversary $\mathcal{S}$ whose running time is polynomial in the running time of $\mathcal{A}$, such that*

$$\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \stackrel{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}. \tag{9}$$

If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries then we say that π adaptively t-privately evaluates *g.*

Spelled out, Equation (9) means that for any value of the security parameter k, for any input vector $\vec{x}$ and any auxiliary input z, the global outputs $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(k,\vec{x},z)$ and $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,\vec{x},z)$ should be identically distributed.

5.2 Discussion

Remark 1: Adaptive security implies non-adaptive security. Intuitively, non-adaptive security appears as a restricted version of adaptive security. We affirm this intuition by observing that Definition 9 (adaptive security) implies Definition 4 (non-adaptive security).

Let us sketch a proof: Let π be a protocol that adaptively t-securely evaluates some function, and let $\mathcal{A}$ be a non-adaptive t-limited adversary. We construct a non-adaptive ideal-model adversary $\mathcal{S}$ that emulates $\mathcal{A}$.

Let $\mathcal{A}'$ be the following adaptive t-limited real-life adversary. $\mathcal{A}'$ receives from its environment a value z that is interpreted as a set C of parties to corrupt, and a value ζ. Next, $\mathcal{A}'$ corrupts the parties in C and runs $\mathcal{A}$ on the set C of corrupted parties, and with auxiliary input ζ. Let $\mathcal{Z}$ be the environment that, on input z, provides the adversary (at the beginning of the interaction) with the value z and remains inactive from this point on. Let $\mathcal{S}'$ be the (adaptive) ideal-model adversary that emulates $\mathcal{A}'$ in the presence of $\mathcal{Z}$. Note that $\mathcal{S}'$ must eventually corrupt exactly the parties in the set provided by $\mathcal{Z}$.

The non-adaptive ideal-model adversary $\mathcal{S}$ proceeds as follows. Given a set C of corrupted parties together with their inputs, plus auxiliary input ζ, ideal-model adversary $\mathcal{S}$ will proceed by running $\mathcal{S}'$; in addition, $\mathcal{S}$ plays the environment for $\mathcal{S}'$ and provides it with a value z that consists of the set C of parties to be corrupted plus the value ζ. Whenever $\mathcal{S}'$ corrupts a party in C, $\mathcal{S}$ provides $\mathcal{S}'$ with the input of that party. Finally, $\mathcal{S}$ outputs whatever $\mathcal{S}'$ outputs. It is evident that $\mathcal{S}$ emulates $\mathcal{A}$.

Remark 2: Additional concerns captured by adaptive security (I). We highlight one aspect of the additional security offered by the adaptive-adversary model, namely the need to account for the fact that the adversary may learn from the communication which parties are worth corrupting more than others. This is demonstrated via an example, taken from [CFGN96]. Consider the following secret sharing protocol, run in the presence of an adversary that may corrupt $t = O(n)$ out of the n parties: *A dealer D chooses at random a small set S of, say, $m = \sqrt{t}$ parties. (In fact, any value $\omega(\log n) < m < t$ will do.) Next, D shares its secret among the parties in S using an m-out-of-m sharing scheme. In addition D publicizes the set S.* (For concreteness, assume that the protocol evaluates the null function.) Intuitively, this scheme lacks in security since S is public and $|S| < t$. Indeed, an adaptive adversary can easily find D's secret, without corrupting D, by corrupting the parties in S. However, any non-adaptive adversary that does not corrupt D learns D's secret only if S happens to be identical to the pre-defined set of corrupted parties. This happens only with probability that is exponentially small (in m). Consequently, this protocol is secure in the presence of non-adaptive adversaries, if a small error probability is allowed. (In particular, if n is polynomial in k then Definition 4 is satisfied with the exception that the two sides of (1) are statistically indistinguishable.)

Remark 3: Additional concerns captured by adaptive security (II). Another security concern that is addressed in the adaptive model, and remains unaddressed in the non-adaptive model, is the need to limit the information gathered by the adversary when it corrupts (or breaks into) parties and sees their internal data. This means that even the internal memory contents of "honest" parties cannot be regarded as "safe" and could compromise the security.

The definition of adaptive security addresses this concern by requiring, essentially, that the internal state seen by the adversary upon corrupting a party is generatable (by the ideal-process adversary) given only the input of this party and the adversary's view so far. Let us demonstrate how this requirement affects the definition, via a the following example. Consider a protocol where each party is instructed to publicize a *commitment* to its input, and then halt with null output. For concreteness, assume that each party has binary input and the commitment is realized via a claw-free permutation pair f_0, f_1 that is known in advance. That is, each party chooses a random element r in the common domain of f_0, f_1 and broadcasts $f_b(r)$, where b is the party's input. It is easy to see that in the non-adaptive model this protocol securely evaluates the null function. However, we do not know how to prove adaptive security of this protocol. In fact, if n, the number of parties, is polynomial in the security parameter and claw-free permutations exist then this protocol does *not* t-securely evaluate the null function in the adaptive model, for $t > \omega(\log n)$. (A proof appears in a slightly different form in [CO99].)

The above discussion may bring the reader to wonder whether it is justifiable to assert that the above protocol is insecure. Indeed, at first glance this protocol appears to be "harmless", in the sense that it has no apparent security weakness. This appearance may be strengthened by the fact that the commitment is perfectly secure, i.e. the messages sent by the parties are statistically independent from the inputs. Nonetheless, we argue that this appearance is false, and the above protocol has a serious security flaw. Indeed, the protocol provides the adversary with a (computationally binding) commitment to the inputs of the parties; this commitment may be useful in conjunction with additional information that may become available to the adversary (say, via other protocol executions). Such a commitment could not have been obtained without interacting with the parties.

Remark 4: Erasing local data. A natural method for limiting the information seen by the adversary upon corrupting a party is to include special *erasure* instructions in the protocol, thereby enabling the parties to remove sensitive data from their local state when this data is no longer necessary.

Indeed, timely erasures of sensitive data can greatly simplify the design and analysis of protocols. (The case of encryption is an instructive example [BH92, CFGN96].) However, basing the security of a protocol on such erasures is often problematic. One reason is that in real-world systems erasures do not always work: System backups are often hard to prevent (they are even made without a protocol's knowledge), and retrieving data that was stored on magnetic media and later erased is often feasible. An even more severe reason to not trust erasure instructions is that they cannot be verified by an outside observer. Thus, in settings where the parties are mutually distrustful it is inadvisable to base the security of one party on the "good will" and competence of *other* parties to effectively erase data as instructed. Consequently, a protocol that offers security without using data erasures is in general preferable to one that bases its security on data erasures.

Let us highlight an important scenario where putting trust in internal erasures is more reasonable. This is the case of threshold cryptography (see, e.g., [DF89]) where the parties are typically special-purpose servers controlled by a single administrative authority, and use erasures to maintain the overall security of the system in the face of break-ins by outsiders. In particular, in the case of proactive security [OY91, CGHN97] trust in erasures is unavoidable since there the attacker may break into *all* parties at one time or another.

The distinction between trusting or dis-trusting data erasures is manifested in the definition via the amount of information seen by the real-life adversary upon corrupting a party. Trusting erasure instructions to be fulfilled and successful is modeled by letting the adversary see only the *current* internal state of the party. Dis-trusting the success of such instructions is modeled by allowing the adversary to see the entire past internal states of the party. (This amounts to allowing the adversary to see the party's input, random input, and all the messages ever received by the party.) In this work we concentrate on the case where erasures are not trusted. Nonetheless, the composition theorem holds in both cases.

Finally, we remark that there exist additional, potentially harmful ways for parties to deviate from the specified protocol in a manner that is undetectable by an outside observer. For instance, a party can use its random input in a different way than specified in the protocol. Proving security of protocols in a model where all parties, even uncorrupted ones, may carry out such deviations is much harder (in fact, it is impossible in some settings). Consequently we do not consider such models; They are mentioned in [CFGN96] and studied in more depth in [CO99]. (The motivation there is to deal with situations where *all parties* may deviate from the protocol, as long as the deviation remains undetected by other parties.)

Remark 5: On the modeling of the environment. Recall that the environment machine is a generalization of the notion of auxiliary input. Indeed, the environment can be used to provide the adversary with auxiliary input at the onset of the interaction. In addition, it can disclose more information to the adversary in an adaptive way throughout the computation. Furthermore, the environment obtains information from the adversary, again in an adaptive way, even after the execution of the protocol is completed.

Informally, in the adaptive model the auxiliary information can be thought of as consisting of two components: a "non-uniform" component, represented by the input z of the environment machine; and an "algorithmic" component, represented by the environment machine itself, that adaptively decides on the way in which information is "released" to the adversary and obtained from it throughout the computation.

Let us address two additional points regarding the modeling of the environment:

On the need in the environment as a separate entity. A natural question is whether it is possible to simplify the definition of adaptive security by merging the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ into a single adversarial entity. We argue that the separation is essential. In particular, the roles played by the two entities in the definition are quite different. Let us stress two main technical differences. Firstly, the environment remains the same in the real-life computation and in the ideal process, whereas the adversary may be modified. Secondly, the environment sees much more information than the adversaries $\mathcal{A}$ and $\mathcal{S}$. In particular, the input of $\mathcal{Z}$ may contain the inputs of *all* parties at the onset of the computation. (Indeed, the proof of the composition theorem below uses an environment machine that sees all this information.) Furthermore, $\mathcal{Z}$ sees the outputs of *all* parties from the computation.

Nonetheless, one can do without the environment machine in some simplified cases. More specifically, the definition of security can be simplified as follows, in the case where local data erasures by parties are allowed. (This is the case discussed in Remark 4, where the adversary sees only the current internal state of a newly corrupted party). First adopt the convention that, whenever a party completes executing a protocol, it *erases* all the internal data relevant to this protocol execution, except for the local output. Next, the definition is simplified in two steps:

First, note that the post-execution corruption phase is no longer necessary. This is so, since corrupting a party after the execution of the protocol is completed reveals only the party's local output. However, the environment anyhow learns the local outputs of all parties as soon as these are generated. Consequently, the post-execution corruption phase does not provide the environment with any new information.

Second, notice that now the role of the environment is restricted to providing the adversary with initial auxiliary input and with an additional auxiliary input whenever a party is corrupted. But these auxiliary inputs represent information that was fixed before the current protocol began. (These are the internal states of the corrupted parties from other protocol executions.) Thus, the environment machine can be replaced by a set $z_1, ..., z_n$ of auxiliary inputs, where the adversary obtains z_i upon the corruption of party P_i.

ON THE ORDER OF QUANTIFIERS. An alternative formulation to Definition 9 requires that a single ideal-process adversary $\mathcal{S}$ will satisfy (9) with respect to any environment $\mathcal{Z}$. We note that this seemingly stronger formulation is in fact implied by (and thus equivalent to) Definition 9.[17] We choose the current formulation because it appears a bit more natural. It also makes the proof of the composition theorem somewhat clearer.

5.3 Modular composition: The adaptive case

We formalize the composition theorem for the non-concurrent case, with adaptive adversaries, in the secure channels setting. As in the non-adaptive case, we first define the hybrid model and describe how an ideal evaluation call is replaced by a subroutine protocol. Next we state the composition theorem in its more general form. The theorem from the Introduction follows as an easy corollary.

The hybrid model. The (adaptive) hybrid model with ideal access to $f_1, ..., f_m$ (or in short the $(f_1, ..., f_m)$-hybrid model), is defined analogously to the non-adaptive case. We start with the real-life model of Section 5.1. This model is augmented with an incorruptible trusted party T for evaluating $f_1, ..., f_m$. The trusted party is invoked at special rounds, determined by the protocol run by the uncorrupted parties. In each such round a function f (out of $f_1, ..., f_m$) is specified. The computation at each special round mimics the ideal process. That is, first the adversary adaptively corrupts parties, and learns the internal data of corrupted parties. In addition, for each corrupted party the adversary receives information from the environment $\mathcal{Z}$. Next the parties hand their f-inputs to T. The values handed by the uncorrupted parties are determined by the protocol. The values handed by the corrupted parties are determined by the adversary. (If the adversary is passive then even corrupted parties hand T values according to the protocol.) Once T receives the values from the parties (value x_i^f from party P_i), it hands the respective outputs back to the parties (P_i receives $f(k, x_1^f..x_n^f, r_f)_i$). Finally the adversary can again adaptively corrupt parties as before.[18]

Let $\text{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},\mathcal{Z}}(k, \vec{x}, z)$ denote the random variable describing the global output of the computation (i.e., the output of the environment $\mathcal{Z}$) in the $(f_1, ..., f_m)$-hybrid model with protocol π, adversary $\mathcal{A}$, security parameter k, inputs $\vec{x}$ for the parties and z for $\mathcal{Z}$, analogously to the definition

[17] The argument is similar to that of Remark 4 in Section 4.2: Assume that a protocol is secure according to Definition 9 and let $\mathcal{A}$ be a real-life adversary. Let $\mathcal{Z}_U$ be the "universal environment" that takes as input a description of an environment $\mathcal{Z}$ and a value z and runs $\mathcal{Z}$ on input z. Definition 9 guarantees that there exists an ideal-model adversary $\mathcal{S}_U$ that emulates $\mathcal{A}$ in the presence of $\mathcal{Z}_U$. It follows that $\mathcal{S}_U$ emulates $\mathcal{A}$ in the presence of *any* environment. That is, $\mathcal{S}_U$ satisfies the above stronger formulation.

[18] As in the non-adaptive case, we assume that the rounds in which ideal evaluations take place, as well as the functions to be evaluated, are fixed and known beforehand. This restriction can be circumvented as there.

of $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,\vec{x},z)$ in Section 5.1. (We stress that here π is not a real-life protocol and uses ideal calls to T.) Let $\mathrm{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble $\{\mathrm{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},\mathcal{Z}}(k,\vec{x},z)\}_{k\in\mathbf{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$.

Replacing an ideal evaluation call with a subroutine call. The 'mechanics' of replacing an ideal-evaluation call of protocol π with a call to a subroutine real-life protocol, ρ, are identical to the non-adaptive case (Section 4.3). Recall that $\pi^{\rho_1 \cdots \rho_m}$ denotes protocol π where each ideal evaluation of f_i is replaced by a call to ρ_i.

Theorem 10 (Adaptive modular composition: General statement) *Let $t<n$, let $m\in\mathbf{N}$, and let $f_1,\ldots,f_m$ be n-party functions. Let π be an n-party protocol in the $(f_1,\ldots,f_m)$-hybrid model where no more than one ideal evaluation call is made at each round, and let $\rho_1,\ldots,\rho_m$ be n-party protocols where ρ_i adaptively t-securely (resp., t-privately) evaluates f_i. Then, for any adaptive t-limited active (resp., passive) real-life adversary $\mathcal{A}$ and for any environment machine $\mathcal{Z}$ there exists an adaptive active (resp., passive) adversary $\mathcal{S}$ in the $(f_1,\ldots,f_m)$-hybrid model whose running time is polynomial in the running time of $\mathcal{A}$, and such that*

$$\mathrm{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{S},\mathcal{Z}} \stackrel{\mathrm{d}}{=} \mathrm{EXEC}_{\pi^{\rho_1,\ldots,\rho_m},\mathcal{A},\mathcal{Z}}. \tag{10}$$

As in the non-adaptive case, Theorem 10 does not assume any security properties from protocol π. Instead, it essentially states that the "input-output functionality" of *any* protocol π in the hybrid model is successfully "emulated" by $\pi^{\rho_1,\ldots,\rho_m}$ in the real-life model. Before rigorously stating the informal composition theorem from the Introduction in the adaptive setting, we define protocols for securely evaluating a function g in the $(f_1,\ldots,f_m)$-hybrid model:

Definition 11 *Let $f_1,\ldots,f_m,g$ be n-party functions and let π be a protocol for n parties in the $(f_1,\ldots,f_m)$-hybrid model. We say that π* adaptively t-securely evaluates g in the $(f_1,\ldots,f_m)$-hybrid model *if for any adaptive t-limited adversary $\mathcal{A}$ (in the $(f_1,\ldots,f_m)$-hybrid model) and any environment machine $\mathcal{Z}$ there exists an adaptive ideal-process adversary $\mathcal{S}$, whose running time is polynomial in the running time of $\mathcal{A}$, and such that:*

$$\mathrm{IDEAL}_{g,\mathcal{S},\mathcal{Z}} \stackrel{\mathrm{d}}{=} \mathrm{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},\mathcal{Z}}. \tag{11}$$

If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries then we say that π adaptively t-privately evaluates g in the $(f_1,\ldots,f_m)$-hybrid model.

Corollary 12 (adaptive modular composition: Secure function evaluation) *Let $t<n$, let $m\in\mathbf{N}$, and let $f_1,\ldots,f_m$ be n-party functions. Let π be an n-party protocol that adaptively t-securely (resp., t-privately) evaluates g in the $(f_1,\ldots,f_m)$-hybrid model, and assume that no more than one ideal evaluation call is made at each round. Let $\rho_1,\ldots,\rho_m$ be n-party protocols that adaptively t-securely (resp., t-privately) evaluate $f_1,\ldots,f_m$, respectively. Then the protocol $\pi^{\rho_1,\ldots,\rho_m}$ adaptively t-securely (resp., t-privately) evaluates g.*

Proof: Let $\mathcal{A}$ be an adaptive t-limited real-life adversary that interacts with parties running $\pi^{\rho_1,\ldots,\rho_m}$, and let $\mathcal{Z}$ be an environment machine. Theorem 10 guarantees that there exists an adversary $\mathcal{A}_\pi$ in the $(f_1,\ldots,f_m)$-hybrid model such that $\mathrm{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}_\pi,\mathcal{Z}} \stackrel{\mathrm{d}}{=} \mathrm{EXEC}_{\pi^{\rho_1,\ldots,\rho_m},\mathcal{A},\mathcal{Z}}$. The security of π in the $(f_1,\ldots,f_m)$-hybrid model guarantees that there exists an ideal model adversary (a "simulator") $\mathcal{S}$ such that $\mathrm{EXEC}_{\pi^{\rho_1,\ldots,\rho_m},\mathcal{A},\mathcal{Z}} \stackrel{\mathrm{d}}{=} \mathrm{IDEAL}_{g,\mathcal{S},\mathcal{Z}}$, satisfying Definition 9. □

5.4 Proof of Theorem 10

As in the non-adaptive case, we only prove the theorem for the case of active adversaries (i.e., t-security). In addition, we only treat the case where the trusted party T is called only once. The extension to the case of multiple functions and multiple calls to T is the same as in the non-adaptive case. Section 5.4.1 contains an outline of the changes from the non-adaptive case. The body of the proof is in Section 5.4.2. All the extensions from Section 4.4.3 are relevant here as well.

5.4.1 Additional difficulties

The proof outline is similar to that of the non-adaptive case. We sketch the additional difficulties arising from the adaptiveness of the adversaries and simulators. Full details appear in Section 5.4.2. Recall that π is a protocol in the f-hybrid model, ρ is a protocol for evaluating f, and π^ρ is the composed protocol. $\mathcal{A}$ is a given adversary that interacts with π^ρ. In addition, we now have an environment $\mathcal{Z}$ that interacts with $\mathcal{A}$. $\mathcal{A}_\rho$ is a constructed adversary that interacts with ρ, following the relevant instructions of $\mathcal{A}$. Adversary $\mathcal{A}_\pi$ follows the instructions of $\mathcal{A}$ relevant to the interaction with protocol π; the interaction of $\mathcal{A}$ with ρ is simulated using $\mathcal{S}_\rho$, the simulator for $\mathcal{A}_\rho$.

1. Recall that $\mathcal{A}_\rho$ operates by running a copy of adversary $\mathcal{A}$. In the adaptive case $\mathcal{A}_\rho$ has to accommodate corruption requests made by $\mathcal{A}$ throughout the execution of π^ρ. For this purpose, $\mathcal{A}_\rho$ is given access to an arbitrary environment machine and proceeds as follows. Corruption requests that occur before ρ is invoked are answered using the initial data received from the environment machine. Whenever the simulated $\mathcal{A}$ requests to corrupt party P during the execution of ρ, adversary $\mathcal{A}_\rho$ corrupts P in its real-life interaction and hands P's internal data to $\mathcal{A}$. P's internal data from the (suspended) execution of protocol π is obtained from the environment. Once ρ is completed and $\mathcal{A}_\rho$ generates its output, a post-execution corruption phase starts where $\mathcal{A}_\rho$ receives corruption requests from its environment, corrupts the relevant parties, and provides the environment with the internal data of the corrupted parties.

2. In the adaptive case specifying an environment is necessary for obtaining a simulator $\mathcal{S}_\rho$ for $\mathcal{A}_\rho$. For this purpose, an environment machine, denoted $\mathcal{Z}_\rho$, is constructed as follows. (Note that $\mathcal{Z}_\rho$ is in general different than the given environment $\mathcal{Z}$.) The input of $\mathcal{Z}_\rho$ will describe a global state of an execution of π^ρ with $\mathcal{A}$ and $\mathcal{Z}$ at round $l^{(\rho)}-1$. $\mathcal{Z}_\rho$ will orchestrate a run of π^ρ from the given global state, with the following exception: $\mathcal{Z}_\rho$ will ignore the random inputs of the uncorrupted parties for the execution protocol ρ. Instead, $\mathcal{Z}_\rho$ will provide $\mathcal{A}_\rho$ with the necessary information for interacting with parties running ρ, and will extract the necessary information from the resulting interaction. More specifically, $\mathcal{Z}_\rho$ first provides $\mathcal{A}_\rho$ with the internal state of $\mathcal{A}$ when ρ is invoked; next, for each party corrupted during the execution of ρ, $\mathcal{Z}_\rho$ provides $\mathcal{A}_\rho$ with the internal state of that party from the suspended execution of π; finally, it extracts from $\mathcal{A}_\rho$ the internal state from the execution of ρ of the parties that are corrupted by $\mathcal{A}$ after ρ is completed.

3. Recall that $\mathcal{A}_\pi$ operates by simulating copies of $\mathcal{A}$ and $\mathcal{S}_\rho$. Here this is done as follows. Let $P'_1, \ldots, P'_n$ denote the set of (simulated) parties with which $\mathcal{S}_\rho$ interacts, and let $P''_1, \ldots, P''_n$ denote the set of (simulated) parties with which $\mathcal{A}$ interacts.

   (a) When adversary $\mathcal{A}_\pi$ runs the simulator $\mathcal{S}_\rho$, it has to accommodate $\mathcal{S}_\rho$'s corruption requests made in the ideal process. This is done as follows: whenever $\mathcal{S}_\rho$ requests to corrupt a party P'_i in the ideal model, adversary $\mathcal{A}_\pi$ corrupts P_i in its hybrid model, and learns the value v that P_i is about to hand its trusted party. Next, $\mathcal{A}_\pi$ "plays the environment for $\mathcal{A}_\rho$" and hands v back to $\mathcal{S}_\rho$ as the input of P'_i. If P'_i is corrupted *after* the ideal call to the trusted party is made, then the output of P'_i value is also given to $\mathcal{S}_\rho$.

(b) Adversary $\mathcal{A}_\pi$ has to accommodate $\mathcal{A}$'s corruption requests made *after* the (simulated) execution of ρ is completed. This is done as follows: Whenever $\mathcal{A}$ requests to corrupt a party P''_i, adversary $\mathcal{A}_\pi$ corrupts P_i in its hybrid model, and obtains the internal data of P_i from protocol π. In addition, $\mathcal{A}_\pi$ plays the role of the environment for $\mathcal{S}_\rho$, and asks $\mathcal{S}_\rho$ to corrupt P'_i. Then, $\mathcal{A}_\pi$ combines the internal data of P_i from protocol π and $\mathcal{S}_\rho$'s answer, obtains simulated internal data for P''_i, and hands this value to the simulated $\mathcal{A}$.

An important point in the analysis is that the way in which $\mathcal{A}_\pi$ 'plays the role of the environment for $\mathcal{S}_\rho$' is identical to an interaction between $\mathcal{S}_\rho$ and $\mathcal{Z}_\rho$.

5.4.2 A detailed proof

Let $\mathcal{A}$ be an adversary and let $\mathcal{Z}$ be an environment (interacting with parties running π^ρ). First we present the constructions of $\mathcal{A}_\rho$, $\mathcal{Z}_\rho$, and $\mathcal{A}_\pi$. Next we show that $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{Z}} \overset{\text{d}}{=} \text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{Z}}$

Terminology. We use the same notions of executions, internal states, and running an adversary from an internal state as in the non-adaptive case (Section 4.4.2). Yet here these notions refer of-course to the adaptive model. In addition, the notion of global state is modified as follows. (Recall that in the non-adaptive case the global state was the concatenation of the local states of the uncorrupted parties and the adversary.)

1. The global state is augmented to include all the information that the uncorrupted parties have ever seen in the past. That is, let the internal history of party P_i at round l be the concatenation of all the internal states from the beginning of the execution through round l. The global state at round l is now the concatenation of the internal *histories* of the uncorrupted parties, together with the internal state of the adversary.

   This convention is needed to maintain the property that the global state of an execution at any round uniquely determines the continuation of the execution until its completion. (Recall that upon corrupting a party the adversary gets access to all the information that the party knew in the past; See Remark 4 in Section 5.2 for more discussion on this definitional decision.)

2. The global state is augmented to include also the local state of the environment.

3. The global state is extended to rounds after the execution of the protocol has been completed, until the the environment halts.

Let $\text{GS}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l,k,\vec{x},z,\vec{r})$ denote the global state at round l of an execution of protocol π^ρ in the real-life model with adversary $\mathcal{A}$, environment $\mathcal{Z}$, security parameter k, inputs $\vec{x}$ for the parties and z to the environment, and random inputs $\vec{r}$. Let $\text{GS}^f_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(l,k,\vec{x},z,\vec{r})$ be similarly defined with respect to protocol π and adversary $\mathcal{A}_\pi$ in the f-hybrid model.

Adversary $\mathcal{A}_\rho$

Let $\mathcal{Z}_\rho$ denote the environment, let $P_1, ..., P_n$ denote the parties running protocol ρ, and let k be a value for the security parameter. (Note that $\mathcal{A}_\rho$ uses the code of $\mathcal{A}$.)

1. Let l_ρ be the round where protocol π^ρ starts running protocol ρ. (This is the round where π calls T). First receive a value ζ_0^ρ from the environment, and verify that ζ_0^ρ is a valid internal state of $\mathcal{A}$ at round $l_\rho \perp 1$. If ζ_0^ρ is not valid then halt with empty output.
2. Corrupt the parties that are corrupted in ζ_0^ρ, and ignore their inputs and the corresponding values received from the environment. (Call these parties the a-priori corrupted parties.)
3. Continue the above run of $\mathcal{A}$ from round l_ρ on, follow $\mathcal{A}$'s instructions, and hand the gathered information to $\mathcal{A}$. More precisely, let $P'_1, ..., P'_n$ denote the simulated parties with which $\mathcal{A}$ interacts. Then:
   (a) Whenever a message is sent from an uncorrupted party P_i to a corrupted party, hand this message to $\mathcal{A}$ as coming from P'_i.
   (b) Whenever $\mathcal{A}$ instructs some corrupted party P'_i to send a message to an uncorrupted party P'_j, instruct P_i to send the same message to P_j.
   (c) When $\mathcal{A}$ corrupts a new party, P'_i, during the execution of protocol ρ, proceed as follows. First corrupt P_i in its real-life model and obtain P_i's internal history for protocol ρ. In addition, $\mathcal{A}$ needs to be provided with the internal history of P'_i from the execution of protocol π, and with the information that $\mathcal{A}$ receives from its own environment at this point. This information is assumed to be provided by the environment, $\mathcal{Z}_\rho$, upon the corruption of P_i. That is, treat the value ζ_i^ρ received from $\mathcal{Z}_\rho$ upon the corruption of P_i as a concatenation of two values $\zeta_i^\rho = \langle a, b\rangle$. The value a is treated as the internal history of P'_i at round $l_\rho \perp 1$; it is combined with the internal history of P_i (pertaining to protocol ρ) and handed to $\mathcal{A}$ as the internal data of P'_i (pertaining to protocol π^ρ). The value b is handed to $\mathcal{A}$ as the value received from $\mathcal{A}$'s environment upon the corruption of P'_i.
4. Once protocol ρ is completed, output the current internal state of the simulated $\mathcal{A}$. Next, interact with the environment $\mathcal{Z}_\rho$, as follows: When the environment asks for corruption of P_i, if less than t parties are corrupted, corrupt P_i and hand P_i's internal history to the environment. If t parties are already corrupted then ignore the corruption request.

Figure 5: Description of Adversary $\mathcal{A}_\rho$ in the adaptive model.

Construction of $\mathcal{A}_\rho$. Given adversary $\mathcal{A}$, adversary $\mathcal{A}_\rho$ proceeds as in the above outline. A more complete description appears in Figure 5.

Construction of $\mathcal{Z}_\rho$. The environment $\mathcal{Z}_\rho$ proceeds as described in the above outline. A detailed description appears in Figure 6.

It follows from the security of protocol ρ that there exists an ideal-process adversary $\mathcal{S}_\rho$ such that $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho} \overset{\text{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}$.

The special structure of $\mathcal{A}_\rho$ implies that $\mathcal{S}_\rho$ has an additional property, described as follows.[19]

[19] This property and the related discussion are very similar to the non-adaptive case. Nonetheless, we repeat the presentation in full, with the appropriate modifications to the adaptive case. A reader that is familiar with the

The environment $\mathcal{Z}_\rho$

Environment $\mathcal{Z}_\rho$ proceeds as follows, given a value k for the security parameter and input ζ, and interacting with parties $P_1, \ldots, P_n$ running protocol ρ and with an adversary $\mathcal{A}_\rho$. (Note that $\mathcal{Z}_\rho$ uses the code of $\mathcal{Z}$ and of $\mathcal{A}$.)

1. The input ζ is assumed to describe a global state at round $l_\rho \perp 1$ of an execution of π^ρ with adversary $\mathcal{A}$ and environment $\mathcal{Z}$. Let ζ_0^ρ denote the internal state of $\mathcal{A}$, let ζ_Z denote the internal state of $\mathcal{Z}$, and let ζ_i^ρ denote the internal history of the ith party, as described in ζ. If the input ζ is not in the right format then halt with no output.
2. (This instruction is carried out throughout the execution of ρ.) Provide $\mathcal{A}_\rho$ with the value ζ_0^ρ. Furthermore, whenever $\mathcal{A}_\rho$ corrupts party P_i, provide $\mathcal{A}_\rho$ with ζ_i^ρ.
3. (This instruction is carried out at the completion of the execution of ρ.) Let u_i denote the output of party P_i, and let u_0 denote the output of $\mathcal{A}_\rho$. Recall that $\mathcal{Z}_\rho$ obtains these values when they are generated.

   Upon obtaining $u_0 \ldots u_n$, run a simulated interaction between adversary $\mathcal{A}$, environment $\mathcal{Z}$, and (simulated) parties $P_1', \ldots, P_n'$ running π^ρ, starting from the round l_π in which protocol π resumes. Adversary $\mathcal{A}$ is run from the internal state described in u_0. Environment $\mathcal{Z}$ is run from state ζ_Z. Party P_i' is run from a state ζ_i' that is obtained from ζ_i and the output u_i of ρ. (Note that ζ_i and u_i may not be sufficient for obtaining a complete internal state of P_i' at round l_π, since the internal data of P_i' from the execution of ρ is not given. However, as long as P_i' remains uncorrupted the internal data from ρ is not needed for the simulated interaction. Figuratively, the internal data from ρ is zeroed out.) When the simulated $\mathcal{A}$ corrupts party P_i', proceed as follows:

   (a) Issue a `corrupt P_i' request to $\mathcal{A}_\rho$. The response, denoted d_i, is interpreted as the internal history of P_i from the execution of ρ.

   (b) Obtain, by continuing the simulation of $\mathcal{Z}$, the value that $\mathcal{Z}$ hands $\mathcal{A}$ upon the corruption of P_i', and hand this value to $\mathcal{A}$.

   (c) Combine d_i with the current (and incomplete) internal history of P_i', obtain P_i's complete internal history for π^ρ, and hand this data to $\mathcal{A}$.
4. Halt when $\mathcal{Z}$ does, with an output value w that is structured as follows. First, w holds the input ζ, followed by $u_0, \ldots, u_n$, the local outputs of all the uncorrupted parties and the adversary at the completion of protocol ρ. Next, w holds the internal data of all the uncorrupted parties, obtained in Step 3a.

Figure 6: Description of the environment $\mathcal{Z}_\rho$.

Note that $\mathcal{A}_\rho$ completely ignores the internal history of the a-priori corrupted parties. (These are the parties that are already corrupted when protocol ρ is invoked.) Therefore, the distribution of the output of $\mathcal{A}_\rho$, as well as the global output of the system after running ρ with $\mathcal{A}_\rho$, remains unchanged if we set the input value of the a-priori corrupted parties to 0, and their internal history to null. Consequently, the distribution of the global output of the ideal process for evaluating f with $\mathcal{S}_\rho$ has the same property. We formalize this discussion as follows. Let $\vec{x}^\rho|_0$ denote the vector obtained from $\vec{x}^\rho$ by replacing all the entries that correspond to the a-priori corrupted parties with

non-adaptive case can safely skip to the construction of adversary $\mathcal{A}_\pi$.

0. Then, we have:

Claim 13 *For any input vector $\vec{x}^\rho$ for the parties and input z^ρ for $\mathcal{Z}_\rho$ we have:*

$$\textsc{ideal}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}(k,\vec{x}^\rho,z^\rho) \overset{\mathrm{d}}{=} \textsc{ideal}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}(k,\vec{x}^\rho|_0,z^\rho)$$

Proof: We have argued above that $\textsc{exec}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}(k,\vec{x}^\rho,z^\rho) \overset{\mathrm{d}}{=} \textsc{exec}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}(k,\vec{x}^\rho|_0,z^\rho)$. However, $\textsc{ideal}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}(k,\vec{x}^\rho,z^\rho) \overset{\mathrm{d}}{=} \textsc{exec}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}(k,\vec{x}^\rho,z^\rho)$, and $\textsc{ideal}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}(k,\vec{x}^\rho|_0,z^\rho) \overset{\mathrm{d}}{=}$ $\textsc{exec}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}(k,\vec{x}^\rho|_0,z^\rho)$. The claim follows. □

Construction of $\mathcal{A}_\pi$. Adversary $\mathcal{A}_\pi$ proceeds as described in the above outline. A detailed description appears in Figure 7.

Analysis of $\mathcal{A}_\pi$. It is evident that the running time of $\mathcal{A}_\pi$ is linear in the running time of $\mathcal{A}$, plus the running time of $\mathcal{S}_\rho$, plus the running time of π^ρ. Fix an input vector $\vec{x}$, an environment $\mathcal{Z}$ with input z, and some value of the security parameter. We show that

$$\textsc{exec}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(k,\vec{x},z) \overset{\mathrm{d}}{=} \textsc{exec}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(k,\vec{x},z) \tag{12}$$

where the symbol $\overset{\mathrm{d}}{=}$ is denotes equality of *distributions,* not ensembles. This is shown in three steps, as follows. (The steps are analogous to the non-adaptive case.)

Let us first set some notation. (This notation is analogous to the non-adaptive case, see Section 4.4.2). Recall that l_ρ is the round where protocol π makes the ideal evaluation call, and protocol π^ρ invokes ρ. Given vectors $\vec{r}^\pi = r^\pi_Z, r^\pi_0, \ldots, r^\pi_n$ and $\vec{r}^\rho = r^\rho_Z, r^\rho_0, \ldots, r^\rho_n$ (where $\vec{r}^\pi$ is interpreted as random input for the execution of π^ρ except for the execution of ρ, and $\vec{r}^\rho$ is interpreted as random input for the execution of ρ), let $\vec{r}^{\pi,\rho} = r^{\pi,\rho}_0, \ldots, r^{\pi,\rho}_n$ denote the combination of $\vec{r}^\pi$ and $\vec{r}^\rho$ to a full random-input vector for the execution of π^ρ. (That is, party P_i uses r^ρ_i for the execution of ρ and r^π_i for the execution of π, the adversary uses r^ρ_0 during the execution of ρ and r^π_0 at other rounds, and the environment uses r^ρ_Z during the execution of ρ and r^π_Z at other rounds.) Similarly, given $r^\pi = r^\pi_Z, r^\pi_0, \ldots, r^\pi_n$ and $\vec{r}^f$, where $\vec{r}^\pi$ is as above and $\vec{r}^f$ is interpreted as a random vector for round l_ρ in the f-hybrid model (that is, $\vec{r}^f = r^f_Z, r^f_0, r^f_1$ where r^f_Z, r^f_0 is the random inputs for the adversary and the environment for this round and r^f_1 is the random input for the trusted party for f), let $\vec{r}^{\pi,f}$ denote the combination of $\vec{r}^\pi$ and $\vec{r}^f$ to a full random-input vector for the execution of π in the f-hybrid model.

Step I. Until round $l_\rho - 1$ protocols π and π^ρ behave the same. That is, fix some value $\vec{r}^\pi$ for the random-input of the system. We have:

$$\textsc{gs}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi) = \textsc{gs}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi).$$

Step II. We show that the global state in the hybrid model at the end of round l_ρ is distributed identically to the global state in the real-life model at the round where protocol ρ returns. This is done in three sub-steps:

1. We first show that the parameters set in the hybrid model for the ideal evaluation of f are identical to the parameters set in the real-life model for the invocation of ρ. Let C be the set of a-priori corrupted parties, determined by $\vec{r}^\pi$. (That is, C is the set of corrupted parties at

Adversary $\mathcal{A}_\pi$

Adversary $\mathcal{A}_\pi$, given value k for the security parameter, and interacting with an environment machine $\mathcal{Z}$, with parties $P_1, ..., P_n$ running protocol π, and with a trusted party T for evaluating f, proceeds as follows. (Note that $\mathcal{A}_\pi$ uses the code of $\mathcal{Z}$ and of $\mathcal{A}$.)

1. As in the non-adaptive case, invoke $\mathcal{A}$ on its own input, auxiliary input and random input, and follow the instructions of $\mathcal{A}$ up to round $l_\rho \perp 1$. (Recall that so far, both in π and in π^ρ the parties run π.) In addition, keep another piece of the random input 'on the side'. This piece, denoted r^ρ, is used below.

2. At the onset of round l_ρ, $\mathcal{A}$ expects to start interacting with parties running protocol ρ (as subroutine), whereas parties $P_1, ..., P_n$ call a trusted party for ideal evaluation of function f. Thus, in order to continue the run of $\mathcal{A}$, invoke simulator $\mathcal{S}_\rho$ as follows. Let $P'_1, ..., P'_n$ denote the set of simulated parties with which $\mathcal{S}_\rho$ interacts, and let $P''_1, ..., P''_n$ denote the set of simulated parties with which $\mathcal{A}$ interacts.

   (a) The random input of $\mathcal{S}_\rho$ is set to r^ρ. The initial value that $\mathcal{S}_\rho$ expects to receive from its environment is set to the current internal state of $\mathcal{A}$.

   (b) When $\mathcal{S}_\rho$ asks to corrupt (in its ideal process) a party P'_i such that P_i is already corrupted, $\mathcal{S}_\rho$ is given input values 0 for P'_i. (Recall that these are the a-priori corrupted parties, thus their inputs and the data from the environment do not affect the distribution of the output of $\mathcal{S}_\rho$.)

   (c) When $\mathcal{S}_\rho$ asks to corrupt a party P'_i that is not yet corrupted, corrupt P_i in the f-hybrid model; let x_i^f be the value that P_i is about to hand T, the trusted party for f. Then inform $\mathcal{S}_\rho$ that the input of P'_i is x_i^f. In addition, set ζ_i^ρ to contain the internal history of P_i, and hand ζ_i^ρ to $\mathcal{A}_\rho$ as the information from the environment.

   (d) When $\mathcal{S}_\rho$ hands the inputs of the corrupted parties to its trusted party, and asks for the values of f, invoke the trusted party, T, for f with the same input values for the corrupted parties, and hand the value provided by the trusted party back to $\mathcal{S}_\rho$.

   (e) If $\mathcal{S}_\rho$ corrupts P'_i after Step 2d then $\mathcal{S}_\rho$ is given also the value that P_i received from the trusted party.

3. Let v denote the output of $\mathcal{S}_\rho$, before it starts the post-execution corruption phase. Recall that v is an internal state of $\mathcal{A}$ at the round, l_π, where the execution of π resumes. Continue the current run of $\mathcal{A}$ from internal state v until the completion of protocol π, and follow $\mathcal{A}$'s instructions. When $\mathcal{A}$ corrupts a party P''_i at this stage, proceeds as follow.

   (a) Corrupt P_i in its f-hybrid model and obtain the internal history of P pertaining to protocol π.

   (b) Play the role of the environment for $\mathcal{S}_\rho$, and request corruption of P'_i. Then obtain the (simulated) internal history of P'_i pertaining to protocol ρ. (In the process $\mathcal{S}_\rho$ may corrupt P'_i in its ideal process. In this case hand $\mathcal{S}_\rho$ the input for P'_i and the value from the environment as described in Step 2c.)

   (c) Combine the data from the previous two steps to obtain the internal history of P''_i pertaining to protocol π^ρ, add the value received from $\mathcal{A}_\pi$'s environment, and hand all this data to $\mathcal{A}$.

4. Once protocol π terminates, output whatever $\mathcal{A}$ outputs, and continue to simulate $\mathcal{A}$ as in Step 3 throughout the post-execution corruption phase.

Figure 7: Description of adversary $\mathcal{A}_\pi$ in the adaptive model.

the onset of round l_ρ.) The set C is identical in the two executions. Let z^ρ, an input value for environment $\mathcal{Z}_\rho$, consist of the global state $z^\rho = \mathrm{GS}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)$.

Let x_i^ρ denote the input value of uncorrupted party P_i for protocol ρ, as determined in $\mathrm{GS}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)$. If P_i is corrupted then $x_i^\rho = 0$. Let $\vec{x}^\rho = x_1^\rho, ..., x_n^\rho$. Similarly, let x_i^f denote the value that party P_i hands the trusted party for f, as determined in $\mathrm{GS}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)$, and let $\vec{x}^f = x_1^f, ..., x_n^f$. It follows that $\vec{x}^\rho = \vec{x}^f|_0$.

2. Next we assert that the global output of the execution of ρ, that is implicit in the run of π^ρ with adversary $\mathcal{A}$, is distributed identically to the global output of the ideal evaluation of f that is implicit in round l_ρ of the run of π in the hybrid model. That is, from the validity of $\mathcal{S}_\rho$, from Step II.1, and from Claim 13 we have:

$$\mathrm{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}(k, \vec{x}^\rho, z^\rho) \overset{\mathrm{d}}{=} \mathrm{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}(k, \vec{x}^\rho, z^\rho) = \mathrm{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}(k, \vec{x}^f|_0, z^\rho) \overset{\mathrm{d}}{=} \mathrm{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}(k, \vec{x}^f, z^\rho) \quad (13)$$

(Note that Equation (13) applies also to the interaction between the environment $\mathcal{Z}_\rho$ and the respective adversaries, after ρ is completed. This fact plays a central role in Step III.)

3. We show that the global state in the hybrid model at the at the end of round l_ρ is distributed identically to the global state in the real-life model when protocol ρ returns. That is, Let l_π denote the round where the call to protocol ρ returns (within protocol π^ρ). Then, it follows from the definition of π^ρ and the constructions of $\mathcal{A}_\rho$, $\mathcal{Z}_\rho$, and $\mathcal{A}_\pi$ that:

   (a) Let $\vec{r}^\rho$ be a random-input vector for protocol ρ. Let $\hat{\mathrm{GS}}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l, k, \vec{x}, z, \vec{r}^{\pi,\rho})$ be the vector $\mathrm{GS}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l, k, \vec{x}, z, \vec{r}^{\pi,\rho})$ after removing, for each uncorrupted party, all the internal states pertaining to protocol ρ except for the output from ρ. Then, $\hat{\mathrm{GS}}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l_\pi, k, \vec{x}, z, \vec{r}^{\pi,\rho})$ can be obtained from $\mathrm{GS}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)$ and $\mathrm{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}(k, \vec{x}^\rho, z^\rho, \vec{r}^\rho)$ via a deterministic, simple process, denoted $\mathcal{C}$. (Process $\mathcal{C}$ essentially updates the internal histories of the parties and the internal state of the adversary. More precisely, recall that $w \overset{\mathrm{def}}{=} \mathrm{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}(k, \vec{x}^\rho, z^\rho, \vec{r}^\rho)$ is the output of $\mathcal{Z}_\rho$ from that execution. Process $\mathcal{C}$ first modifies the internal history of each uncorrupted party P_i by adding the appropriate portion of w to $\mathrm{GS}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)$ in the appropriate place. Next $\mathcal{C}$ outputs the internal state of $\mathcal{A}$ as it appears in $\mathcal{A}_\rho$'s output in w, together with the modified internal histories of the uncorrupted parties.)

   (b) Let $\vec{r}^f$ be a random input vector for the ideal evaluation process of f. Then, $\mathrm{GS}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(l_\rho, k, \vec{x}, z, \vec{r}^{\pi,f})$ is obtained from $\mathrm{GS}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)$ and $\mathrm{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}(k, \vec{x}^f, z^f, \vec{r}^f)$ via *the same process, $\mathcal{C}$, as in the real-life execution.*

   It follows that for any value of $\vec{r}^\pi$, and for vectors $\vec{r}^\rho$ and $\vec{r}^f$ that are uniformly chosen in their respective domains, we have

$$\hat{\mathrm{GS}}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l, k, \vec{x}, z, \vec{r}^{\pi,\rho}) \overset{\mathrm{d}}{=} \mathrm{GS}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(l_\rho, k, \vec{x}, z, \vec{r}^{\pi,f}).$$

Step III. We assert Equation (12). We have:

1. For each round $l > l_\pi$ the vector $\hat{\mathrm{GS}}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l, k, \vec{x}, z, \vec{r}^{\pi,\rho})$ can be obtained from $\hat{\mathrm{GS}}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l - 1, k, \vec{x}, z, \vec{r}^\pi)$ and $w \overset{\mathrm{def}}{=} \mathrm{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}(k, \vec{x}^\rho, z^\rho, \vec{r}^\rho)$ via the following process, $\mathcal{C}'$: Continue the execution for one round from the global state described in $\hat{\mathrm{GS}}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l - 1, k, \vec{x}, z, \vec{r}^\pi)$. If no new corruption occurs in this round then $\hat{\mathrm{GS}}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l, k, \vec{x}, z, \vec{r}^{\pi,\rho})$ is obtained. In case that $\mathcal{A}$

corrupts a new party, P_i, take the internal history of P_i pertaining to protocol ρ from w. (It is guaranteed that P_i is corrupted in w.)

$\mathcal{A}$'s interaction with the environment $\mathcal{Z}$ at the completion of the execution of π^ρ is determined by $\hat{\text{GS}}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(l_\pi, k, \vec{x}, z, \vec{r}^\pi)$ and w via a similar process. In particular, the global output $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(k, \vec{x}, z, \vec{r}^{\pi,\rho})$ is uniquely determined.

2. For each round $l > l_\pi$ vector $\text{GS}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(l, k, \vec{x}, z, \vec{r}^{\pi,f})$ is determined from $\text{GS}_{\pi,\mathcal{A}_\pi,\mathcal{Z}_\rho}(l-1, k, \vec{x}, z, \vec{r}^{\pi,f})$ and $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}(k, \vec{x}^\rho, z^\rho, \vec{r}^\rho)$ via the same process, $\mathcal{C}'$, as in the the real-life execution. The interaction with environment $\mathcal{Z}$ at the completion of the execution of π is also determined in the same way as there. In particular, the global output $\text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(k, \vec{x}, z, \vec{r}^{\pi,f})$ is determined in the same way as there.

It follows that for any value of $\vec{r}^\pi$, and for vectors $\vec{r}^\rho$ and $\vec{r}^f$ that are uniformly chosen in their respective domains, we have

$$\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(k, \vec{x}, z, \vec{r}^{\pi,\rho}) \overset{\text{d}}{=} \text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(k, \vec{x}, z, \vec{r}^{\pi,f}).$$

Equation (12) follows by letting $\vec{r}^\pi$ be randomly chosen in its domain.

This completes the proof for the case of a single ideal evaluation call. The case of multiple ideal evaluation calls is treated in the same way as in the non-adaptive case. We omit further details.

6 The computational setting

This section defines secure protocols and proves the composition theorem in the computational setting, where the adversary sees all the communication among the parties and is restricted to probabilistic polynomial time. We concentrate on the case of adaptive adversaries. The simpler case of non-adaptive adversaries can be easily inferred.

The treatment is quite similar to that of the secure channels setting (Section 5). Therefore, this section is *not* self-contained; we assume familiarity with Section 5 and only highlight the differences. Section 6.1 contains definitions of secure protocols. All the remarks from Sections 4.2 and 5.2 are relevant here. Additional remarks specific to the computational setting appear in Section 6.2. Section 6.3 presents and proves the composition theorem.

6.1 Definition of security: The computational case

We define adaptively secure multi-party computation in the computational setting. Executing a protocol π in the real-life scenario, as well as the notation $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$, are the same as in the adaptive secure channels setting, with the following exceptions:

1. The real-life adversary, $\mathcal{A}$, and the environment $\mathcal{Z}$ are probabilistic polynomial time (PPT). Note that this is a *weakening* of the security offered by this model, relative to that of Section 5. (The running time of the adversary, as well as that of all other entities involved, is measured as a function of the security parameter, k. To accommodate the convention that the running time is measured against the length of the input we envision that the string 1^k is given as an additional input.)

2. $\mathcal{A}$ sees all the communication between the uncorrupted parties. Consequently, when a party gets corrupted the *new* data learned by the adversary is only the party's input and random

input. Note that this is a *strengthening* of the security offered by this model, relative to that of Section 5.[20]

The ideal process is the same as in the secure channels setting. (Since the real-life adversary is always PPT, so is the ideal-process adversary.) The notation $\mathrm{IDEAL}_{f,\mathcal{S},\mathcal{Z}}$ remains unchanged.

We define emulation of the ideal process by a real-life computation in the same way, with the exception that here we only require that the global outputs are *computationally indistinguishable* (as defined in Section 3):

Definition 14 (adaptive security in the computational setting) *Let f be an n-party function, and let π be a protocol for n parties. We say that π* adaptively t-securely evaluates f in the computational setting*, if for any* PPT *t-limited real-life adversary $\mathcal{A}$ and any* PPT *environment $\mathcal{Z}$ there exists a* PPT *ideal-process adversary $\mathcal{S}$, such that*

$$\mathrm{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \tag{14}$$

If $\mathcal{A}$ and $\mathcal{S}$ are passive then π adaptively t-privately evaluates f in the computational setting.

6.2 Discussion

Remark 1: On the complexity of $\mathcal{Z}$. We stress that Definition 14 quantifies only over all environments $\mathcal{Z}$ that are PPT. This is so since in the computational setting we assume that all involved entities (including the environment, represented by $\mathcal{Z}$) are PPT. Indeed, a definition that allows $\mathcal{Z}$ more computational power will be hard to satisfy, since an over-powerful $\mathcal{Z}$ may be able to break cryptographic primitives used by the parties, and thus distinguish between the real-life computation and the ideal process. (Recall that our model allows $\mathcal{Z}$ access to the communication among the parties, via $\mathcal{A}$'s view.)

Remark 2: On "absolute" vs. "computational" correctness. Definition 14 only requires the two sides of (14) to be computationally indistinguishable. (That is, it is required that for any PPT distinguishing algorithm $\mathcal{D}$, and for any values of $k, \vec{x}, z$, algorithm $\mathcal{D}$ distinguishes between $(k, \vec{x}, z, \mathrm{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(k, \vec{x}, z))$ and $(k, \vec{x}, z, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, \vec{x}, z))$ only with probability that is negligible in the security parameter k.) In particular, this means that the ensemble describing the outputs of the *uncorrupted* parties in the real-life model is only required to be computationally indistinguishable from the ensemble describing these outputs in the ideal process.

Let us first discuss the consequences of this requirement in the case of *passive* adversaries. The case of active adversaries is somewhat more involved and is addressed below. In the case of passive adversaries Definition 14 imposes different requirements depending on whether the evaluated function is deterministic or probabilistic. When f is deterministic the output of each uncorrupted party in a protocol that securely evaluates f will be the (uniquely determined) value of f on the corresponding set of inputs. In this case, we say that the definition guarantees "absolute correctness". When f is probabilistic, a protocol that securely evaluates f only guarantees that the distribution of the outputs of the uncorrupted parties is computationally indistinguishable from the specified distribution. It is *not* guaranteed that the distribution of the outputs of the uncorrupted

[20] We assume that the links are *ideally authenticated*, namely the adversary cannot *alter* the communication. This assumption is used in many works on cryptographic protocols, and makes the analysis of protocols much easier. Removing this assumption can be done in a "modular" way that concentrates on the task of message authentication. See, for instance, [BCK98].

parties will be equal to the specified distribution. In this case, we say that the definition guarantees "computational correctness".

Let us demonstrate this point via an example. Assume that the function to be evaluated is $f_1(x_1, ..., x_n) = g(\oplus_{i=1}^{n} x_i)$ where g is some pseudorandom number generator, and $\oplus$ denotes bitwise exclusive or. In this case, only protocols where the uncorrupted parties output the value of $g(\oplus_{i=1}^{n} x_i)$ on *any* input sequence $x_1, ..., x_n$ will be considered secure. In contrast, assume that the evaluated function is $f_2() = r$, where r is a random value of the same length as $g(\cdot)$ above. (That is, f_2 is a probabilistic function and r is chosen using the "intrinsic randomness" of f_2.) In this case, *any* protocol in which the parties output a *pseudorandom value* of the appropriate length is secure.

In the case of active adversaries the distinction between the cases where the definition guarantees "absolute correctness" and the cases where the definition guarantees only "computational correctness" is more drastic. The reason is that here the corrupted parties (both in the real-life and in the ideal model) may contribute to the computation values chosen irrespectively of the given input values; in particular the contributed values can be chosen *randomly* according to some distribution. Consequently, the definition guarantees "absolute correctness" only for functions where the output value is uniquely determined by the inputs of the *uncorrupted* parties alone.

Let us demonstrate this point via another example. Consider the function f_1 described above. This function is deterministic; however, the value x (and, consequently, the output of the parties) is not well-defined given the inputs of the uncorrupted parties. In particular, when the corrupted parties contribute randomly chosen values, the function value is in effect $g(r)$ where r is random and independent from the inputs of the parties. Therefore it is possible to construct protocols that securely evaluate f_1 according to Definition 14, but where the parties output a random value, independently of the inputs of the parties.

In contrast, consider the function $f_3(x_1, ..., x_n) = g(x_1)$ if $x_1 = x_2 = ... = x_n$, and $f_3(x_1, ..., x_n) = -$ otherwise. Here the output of the parties is uniquely defined (up to an error value) given the inputs of the uncorrupted parties. Consequently, in a protocol that securely evaluates f_3 the uncorrupted parties output the (uniquely defined) output value on each input.

The above discussion brings us to the more general issue of how to formally cast an "intuitive task" as a function to be evaluated. We have seen that seemingly similar formalizations result in very different security requirements on protocols. Thus, care must be taken to formalize a given task in a way that correctly captures the desired security requirements.

6.3 Modular composition: The computational case

We state and prove the composition theorem and its corollary for the case of adaptive adversaries in the computational setting.

The computational hybrid model. The (computational, adaptive) $(f_1, ..., f_m)$-hybrid model is defined identically to the secure channels, adaptive case (Section 5.3), with the exception that we start from the computational real-life model, rather than from the secure-channels real-life model. The notation $\textsc{exec}^{f_1,...,f_m}_{\pi,\mathcal{A},\mathcal{Z}}$ remains unchanged (here it applies to the computational setting). The 'mechanics' of replacing an ideal-evaluation call of protocol π with a call to a subroutine real-life protocol, ρ, are also identical to the case of secure channels.

Theorem 15 (Adaptive computational modular composition: General statement) *Let $t < n$, and let $f_1, ..., f_m$ be n-party functions. Let π be an n-party protocol in the computational*

$(f_1, \ldots, f_m)$-hybrid model where no more than one ideal evaluation call is made at each round, and let $\rho_1, \ldots, \rho_m$ be n-party protocols where ρ_i adaptively t-securely (resp., t-privately) evaluates f_i in the computational setting. Then, for any PPT *adaptive t-limited active (resp., passive) real-life adversary $\mathcal{A}$ and for* PPT *any environment machine $\mathcal{Z}$ there exists a* PPT *adaptive active (resp., passive) adversary $\mathcal{S}$ in the $(f_1, \ldots, f_m)$-hybrid model such that*

$$\text{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{S},\mathcal{Z}} \stackrel{c}{\approx} \text{EXEC}_{\pi^{\rho_1,\ldots,\rho_m},\mathcal{A},\mathcal{Z}}. \tag{15}$$

Protocols for securely evaluating a function g in the computational $(f_1, \ldots, f_m)$-hybrid model are defined in the usual way:

Definition 16 *Let $f_1, \ldots, f_m, g$ be n-party functions and let π be a protocol for n parties in the computational $(f_1, \ldots, f_m)$-hybrid model. We say that π* adaptively t-securely evaluates g in the computational $(f_1, \ldots, f_m)$-hybrid model *if for any* PPT *adaptive t-limited adversary $\mathcal{A}$ (in the $(f_1, \ldots, f_m)$-hybrid model) and every* PPT *environment $\mathcal{Z}$, there exists a* PPT *adaptive ideal-process adversary $\mathcal{S}$ such that*

$$\text{IDEAL}_{g,\mathcal{S},\mathcal{Z}} \stackrel{c}{\approx} \text{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},\mathcal{Z}}. \tag{16}$$

If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries then we say that π adaptively t-privately evaluates g in the computational $(f_1, \ldots, f_m)$-hybrid model.

Corollary 17 (Adaptive computational modular composition: Secure function evaluation) *Let $t < n$, and let $f_1, \ldots, f_m, g$ be n-party functions. Let π be an n-party protocol that adaptively t-securely (resp., t-privately) evaluates g in the computational $(f_1, \ldots, f_m)$-hybrid model, and assume that no more than one ideal evaluation call is made at each round. Let $\rho_1, \ldots, \rho_m$ be n-party protocols that adaptively t-securely (resp., t-privately) evaluate $f_1, \ldots, f_m$, respectively, in the computational setting. Then protocol $\pi^{\rho_1,\ldots,\rho_m}$ adaptively t-securely (resp., t-privately) evaluates g in the computational setting.*

The proof of Corollary 17 is identical to that of Corollary 12.

Proof of Theorem 15: Again, we only prove the theorem for the case of active adversaries. The simpler case of passive adversaries can be easily inferred. As in the case of adaptive security with secure channels, we first restrict the presentation to active adversaries and to protocols where the trusted party is called only once. The case of multiple ideal evaluation calls is treated at the end of the proof.

The constructions of $\mathcal{A}_\rho$, $\mathcal{Z}_\rho$, and $\mathcal{A}_\pi$ are identical to those of Section 5.4 (the adaptive, secure channels case), with the obvious exception that the simulated adversary $\mathcal{A}$ is always being given also the messages sent among the uncorrupted parties. The complexities of $\mathcal{A}_\rho$, $\mathcal{Z}_\rho$ are linear in the complexity of $\mathcal{A}$, and the complexity of $\mathcal{A}_\pi$ is linear in the complexities of $\mathcal{A}$ and $\mathcal{S}_\rho$. We show:

$$\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \text{EXEC}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}. \tag{17}$$

Essentially, the only difference from the proof in secure channels case is in Step II.2, namely that $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}$ and $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}$ are only guaranteed to be computationally indistinguishable; but this suffices to show (17).

More precisely, given a distinguisher $\mathcal{D}$ between $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}$, construct a distinguisher $\mathcal{D}'$ between $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}$ and $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}$. On input k and a value w (which is the output of $\mathcal{Z}_\rho$), distinguisher $\mathcal{D}'$ orchestrates an execution of π^ρ with adversary $\mathcal{A}$ and corruptor $\mathcal{Z}$, on the inputs and random inputs appearing in w, and using the data in w for the parties' outputs from

ρ. Once the global output w' from this execution is generated, $\mathcal{D}'$ runs $\mathcal{D}$ on (k, w') and outputs whatever $\mathcal{D}$ outputs.

Using the same arguments as in the secure channels case, it is seen that if w has the distribution of $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}(k, \vec{x}, z)$ for some $\vec{x}, z$ then w' has the distribution of $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(k, \vec{x}, z)$. Similarly, if w has the distribution of $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}(k, \vec{x}, z)$ then w' has the distribution of $\text{EXEC}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(k, \vec{x}, z)$. Consequently, if $\mathcal{D}$ distinguishes between $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(k, \vec{x}, z)$ and $\text{EXEC}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{Z}}(k, \vec{x}, z)$ with probability that is not negligible, then $\mathcal{D}$ distinguishes between $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{Z}_\rho}(k, \vec{x}, z)$ and $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{Z}_\rho}(k, \vec{x}, z)$ with probability that is not negligible.

On multiple ideal evaluation calls. As in the secure channels model, the case of multiple ideal evaluation calls is a straightforward generalization of the case of a single call. The construction of the generalized adversary is the same as in the secure channels model; however, the analysis uses a "hybrids argument". We sketch the main points of difference from the single call, computational case. (These points are analogous to the ones discussed in the non-adaptive, secure channels case, see Section 4.4.2.)

1. An adversary $\mathcal{A}_{\rho_i}$ is constructed for each protocol ρ_i. All the $\mathcal{A}_{\rho_i}$'s are identical to adversary $\mathcal{A}_\rho$ described above, with the exception that protocol ρ is replaced by ρ_i. If $\rho_i = \rho_j$ for some i, j then $\mathcal{A}_{\rho_i} = \mathcal{A}_{\rho_j}$.

2. Similarly, an environment machine $\mathcal{Z}_{\rho_i}$ is constructed for each protocol ρ_i. All the $\mathcal{Z}_{\rho_i}$'s are identical to $\mathcal{Z}_\rho$ described above, with the exception that protocol ρ is replaced by ρ_i. (If $\rho_i = \rho_j$ for some i, j then $\mathcal{Z}_{\rho_i} = \mathcal{Z}_{\rho_j}$.)

3. Construct an adversary $\tilde{\mathcal{A}}_\pi$ that is identical to $\mathcal{A}_\pi$ described above, with the exception that at each round where π instructs the parties to ideally evaluate f_i, adversary $\tilde{\mathcal{A}}_\pi$ runs a copy of $\mathcal{S}_{\rho_i}$ in the same way that $\mathcal{A}_\pi$ runs $\mathcal{S}_\rho$. The initial value given to $\mathcal{S}_{\rho_i}$ is set to the current internal state of the simulated $\mathcal{A}$ within $\mathcal{A}_\pi$. (Recall that there may be many invocations of the same simulator $\mathcal{S}_{\rho_i}$, where each invocation corresponds to a different ideal evaluation call to f_i. These invocations will have different initial values.)

4. As in the case of a single ideal evaluation call, it is evident that the running time of $\tilde{\mathcal{A}}_\pi$ is linear in the running time of $\mathcal{A}$, plus the sum of the running times of all the invocations of $\mathcal{S}_{\rho_1}, \ldots, \mathcal{S}_{\rho_m}$, plus the running time of $\pi^{\rho_1,\ldots,\rho_m}$. We sketch a proof that $\text{EXEC}_{\pi^{\rho_1,\ldots,\rho_m},\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \text{EXEC}^{f_1,\ldots,f_m}_{\pi,\tilde{\mathcal{A}}_\pi,\mathcal{Z}}$. Let c denote the total number of ideal evaluation calls made by π in the $f_1, \ldots, f_m$-hybrid model. First define $c+1$ hybrid protocols $\pi_0, \ldots, \pi_c$, all in the $f_1, \ldots, f_m$-hybrid model, where π_j follows π until the end of the jth ideal evaluation call, and follows $\pi^{\rho_1,\ldots,\rho_m}$ for the rest of the interaction. Similarly, define $c+1$ adversaries $\mathcal{A}^0, \ldots, \mathcal{A}^c$, where $\mathcal{A}^j$ is the adversary that follows the instructions of $\mathcal{A}_\pi$ until the end of the jth ideal evaluation call, and follows the instructions of $\mathcal{A}$ for the rest of the interaction. Let H_j denote the ensemble $\text{EXEC}^{f_1,\ldots,f_m}_{\pi_j,\tilde{\mathcal{A}}^j,\mathcal{Z}}$.

   It can be seen that $H_0 = \text{EXEC}_{\pi^{\rho_1,\ldots,\rho_m},\mathcal{A},\mathcal{Z}}$ and $H_c = \text{EXEC}^{f_1,\ldots,f_m}_{\pi,\tilde{\mathcal{A}}_\pi,\mathcal{Z}}$. Furthermore, using a similar argument to the one used for the single call case, it can be seen that if there exists a distinguisher between H_j and H_{j+1} for some $j > 0$ then there exists a distinguisher between $\text{EXEC}_{\rho_i,\mathcal{A}_{\rho_i},\mathcal{Z}_{\rho_i}}$ and $\text{IDEAL}_{f_i,\mathcal{S}_\rho,\mathcal{Z}_{\rho_i}}$, where f_i is the function evaluated in the jth call. (The distinguishing probability is reduced by a factor of c.)

□

Non-adaptive security in the computational setting. A definition of non-adaptive security in the computational setting can be easily derived from Definitions 4 and 14. Furthermore, composition theorems similar to the ones here hold in that case as well.

We remark, however, that in the computational non-adaptive case the distinguisher $\mathcal{D}'$ described above does not work. This is so since there, in contrast to the adaptive case, the global output of the execution of protocol ρ does not include sufficient information for orchestrating an execution of π^ρ with $\mathcal{A}$. Consequently, $\mathcal{D}'$ will receive this information, namely the inputs and random inputs of the parties for protocol π, in its auxiliary input (see Definition 3).[21] See more details on the non-adaptive computational case in [G98].

Acknowledgments

Coming up with the definitions presented here would not have been even remotely possible without the devoted help of Oded Goldreich over a period of several years. Oded has contributed immensely to their shaping, as well as to the modular composition theorems.

Let me also thank the many people I interacted with for very helpful discussions and inputs concerning the definitions and the modular composition theorems. Among these are Rosario Gennaro, Shafi Goldwasser, Shai Halevi, Hugo Krawczyk, Eyal Kushilevitz, Silvio Micali, Moni Naor, Rafi Ostrovsky, Charlie Rackoff and Phil Rogaway.

Finally let me thank the anonymous referees for their careful and thorough reading of the manuscript and for their very helpful remarks and suggestions.

References

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

[B91A] D. Beaver, "Foundations of Secure Interactive Computing", *CRYPTO '91, Lecture Notes in Computer Science (LNCS) 576,* Springer-Verlag, 1991.

[BG89] D. Beaver and S. Goldwasser, "Multi-party computation with faulty majority", *30th Symp. on Foundations of Computer Science (FOCS),* IEEE, 1989, pp. 468-473.

[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.

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

[21] Indeed, in the case of adaptive adversaries a weaker version of Definition 3 that does not provide the distinguisher with auxiliary input would be sufficient for the composition theorem to hold. We formulate the stronger notion in order to be compatible with the non-adaptive computational case.

[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.

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

[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.

[CH94] R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In *Proceedings of CRYPTO'94, LNCS 839,* Springer-Verlag, 1994.

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

[CGHN97] R. Canetti, R. Gennaro, A. Herzberg, D. Naor, “Proactive security: Long-term Protection against break-ins”, CryptoBytes, Vol. 3, No. 1, 1997.

[CG99] R. Canetti and S. Goldwasser, “A threshold cryptosystem secure against adaptive chosen ciphertext attacks”, In *Proceedings of Eurocrypt 99*, Springer-Verlag, 1999. Fuller version available on-line at http://philby.ucsd.edu.

[CHH98] R. Canetti, S. Halevi and A. Herzberg, “How to Maintain Authenticated Communication”, this issue. Preliminary version at *16th Symp. on Principles of Distributed Computing (PODC),* ACM, 1997, pp. 15-25.

[CKOR98] R. Canetti, E. Kushilevitz, R. Ostrovsky and A. Rosen, “Randomness vs. Fault-Tolerance”, this issue. Preliminary version at *16th Symp. on Principles of Distributed Computing (PODC),* ACM, 1997, pp. 35-45.

[CO99] R. Canetti and R. Ostrovsky. Secure Computation with Honest-Looking Parties: What if nobody is truly honest? *31st Symposium on Theory of Computing (STOC),* ACM, 1999.

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

[CK89] B. Chor and E. Kushilevitz, “a Zero-One law for boolean privacy”, SIAM J. Disc. Math. 4 (1991) 36-47. Preliminary version in Symposium on Theory of Computing (STOC) 21, ACM, (1989) 62-72.

[CDM98] R. Cramer, I. Damgard and U. Maurer ,“Span programs and general secure multi-party computation”, manuscript, 1998.

[DF89] Y. Desmedt and Y. Frankel, “Threshold cryptosystems”, In G. Brassard, editor, *Advances in Cryptology — Crypto '89,* LNCS No. 435, Springer-Verlag, 1989, pp. 307–315.

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

[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.

[F87] P. Feldman, "A practical scheme for non-interactive Verifiable Secret Sharing", *28th Symp. on Foundations of Computer Science (FOCS),* IEEE, 1987, pp. 427-437.

[GHY88] Z. Galil, S. Haber and M. Yung, Cryptographic computation: Secure faut-tolerant protocols and the public-key model, *CRYPTO '87, LNCS 293*, Springer-Verlag, 1988, pp. 135-155.

[GJKR96] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold DSS signatures", *Eurocrypt '96, LNCS 1070,* Springer-Verlag, 1996, pp. 354–371.

[G93] O. Goldreich, "A Uniform Complexity Treatment of Encryption and Zero-Knowledge", *Journal of Cryptology*, Vol. 6, No. 1, Springer-Verlag, 1993, pp. 21–53.

[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.

[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.

[GMR89] 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.

[HM97] M. Hirt and U. Maurer, "Complete characterization of adversaries tolerable in secure multiparty computation", this issue. Preliminary version at *16th Symp. on Principles of Distributed Computing (PODC),* ACM, 1997, pp. 25–34.

[K89] E. Kushilevitz, "Privacy and communication complexity", SIAM Jour. disc. Math. Vol. 5. No. 2 (1992), 273-284. Preliminary version in 29th Symp. on Foundations of Computer Science (FOCS), IEEE, 1989.

[KKMO97] J. Kilian, E. Kushilevitz, S. Micali, R. Ostrovsky, "Reducibility and Completeness in Private Computations", *SIAM. J. Computing,* to appear. Preliminary versions in 23rd Symposium on Theory of Computing (STOC), ACM, 1991 by Kilian and in 35th Symp. on Foundations of Computer Science (FOCS), IEEE, 1994, by Kushilevitz, Micali and Ostrovsky.

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

[M96] S. Micali, advanced class on cryptographic protocols, given at MIT, Spring 1996.

[OY91] R. Ostrovsky and M. Yung. "How to withstand mobile virus attacks". In *Proceedings of the* 10^{th} *Annual ACM Symposium on Principles of Distributed Computing (PODC),* ACM, 1991, pp. 51–59.

[R98] T. Rabin, "A Simplified Approach to Threshold and Proactive RSA" *CRYPTO '98, LNCS ,* Springer-Verlag, 1998.

[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.

[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.

A Other definitions

We briefly review some definitions of secure multi-party computation. More specifically, we review the definitions of Micali and Rogaway [MR91], Goldwasser and Levin [GL90], Beaver [B91A, B91], and Canetti, Feige, Goldreich and Naor [C95, CFGN96]. These definitions vary in their level of restrictiveness. In addition, the works vary in the level of detail and rigor in which the definitions are presented. The most comprehensively and rigorously presented set of definitions appears in [MR91].

The definition of Micali and Rogaway. Micali and Rogaway envision an ideal process, similar to the one here, for secure function evaluation. However, the ideal process remains as a motivating intuition and is not explicitly used in the actual definition, sketched below. (This definition deals only with the secure channels setting, and only with protocols that evaluate deterministic functions.)

First the input that each party contributes to the computation, as well as its output, should be determined exclusively from the communication of that party with the other parties. The functions that determine the input and output, called input awareness and output awareness functions, should be computable in polynomial time. (The adversary cannot evaluate these functions since in the secure channels setting it does not have access to the entire communication of an uncorrupted party with the other parties.)

Correctness is guaranteed by requiring that, in any execution of the protocol, the outputs of the uncorrupted parties (determined by applying the output awareness function to the communication) should equal the value of the evaluated function applied to the contributed inputs (determined by applying the input awareness function to the communication). Security is guaranteed by requiring that there exists a 'black-box simulator' that generates, in probabilistic polynomial time, a simulated conversation between the (real-life) adversary and the uncorrupted parties. The simulator is restricted to *one pass* simulation (i.e., it cannot rewind the adversary), and receives external information regarding the inputs of the corrupted parties and their outputs. This external information is related to the values of the input and output awareness functions applied to the simulated conversation. Furthermore, it is received in a timely fashion: the simulator receives the designated outputs of the corrupted parties (i.e., the appropriate function values) only at a certain pre-specified round (this is the round where the inputs become determined by the input-awareness function applied to the comunication); in addition, only when a party is corrupted by the adversary can the simulator receive the input value of that party.

This definition of security seems to imply ours (in the settings where it applies). In fact, it seems considerably more restrictive. Let us highlight three aspects of this extra restrictiveness. First, the requirement that the input and output awareness functions be computable from the communication alone implies that protocols where parts of the computation are done locally without interaction (e.g., the trivial protocol where no communication takes place and each party computes its output locally) are considered insecure. Second, limiting the simulator to *one pass* black-box simulation excludes a proof technique that seems essential for proving security of a wide range of protocols (e.g., zero-knowledge proofs [GMR89, GMW91]). Third, requiring that the simulator receives the outputs of the corrupted parties only after the inputs are determined by the communication excludes an additional set of protocols.[22]

The definition of Goldwasser and Levin. Goldwasser and Levin take a different approach. First they formalize the 'inevitable advantages' of the adversary in the ideal process (we briefly sketch these 'inevitable advantages' below). Next they say that a protocol is robust if for any adversary there exists an 'equivalent' adversary that is limited to these 'inevitable privileges', and that has the same effect on the computation. Their notion of robustness of protocols has the advantage that it is independent of the specific function to be evaluated (except for some technical subtleties ignored here).

The 'inevitable privileges' of the adversary, extracted from the ideal process, can be sketched as follows. First, the adversary may choose to corrupt parties (either adaptively or non-adaptively). Next, if the adversary is active then the inputs of the corrupted parties may be modified. (However, this is done without knowledge of the inputs of the uncorrupted parties). Next, the adversary may learn the specified outputs of the corrupted parties. This may inevitably reveal some information on the inputs of the uncorrupted parties. Furthermore, if the adversary is adaptive then it can corrupt parties, after the computation is completed, based on the output of the computation.[23]

The difference between the [GL90] approach and ours may be viewed as follows. Instead of directly comparing (as we do) executions of the protocol in real-life to an ideal process where a specific function is evaluated, they first compare real-life executions of the protocol to executions of the *same protocol* in an idealized model where the adversary is limited as described above. So far one does not need to specify what functionality the protocol is fulfilling. In a second step (which is implicit in [GL90]), one claims that executing the protocol in the idealized model is equivalent to an ideal evaluation process of a specific function.

[22] For instance, let the "bit transmission" function be such that the output of party R (the receiver) equals the input of party S (the sender). Consider the protocol where S simply sends its input to R over the private channel. This protocol is rejected by the definition of [MR91] since the simulator is required to provide a corrupted receiver with the value of the transmitted bit *before* this value becomes known. (This protocol securely evaluates the bit transmission function according to the definition here.)

[23] If a majority of the parties are corrupted then, in addition to the privileges described above, the adversary cannot be prevented from "quitting early", i.e. disrupting the computation at any time. However, this is done without knowing the output with more certainty than the uncorrupted parties.

The definition of Beaver. Beaver's definition [B91, B91A] takes a similar approach to the one here. We sketch this approach using the terminology of [B91]. First a general notion of comparing security of protocols is formulated, as follows. Consider two protocols α and β for evaluating the same function. Protocol α is at least as secure as protocol β if there exists an interface that turns any adversary $\mathcal{A}$ attacking α into an adversary $\mathcal{A}'$ attacking β, such that for any inputs the global output of the two computations are identically distributed. The global output is defined similarly to here. The interaction between the interface and $\mathcal{A}$ is apparently black-box, and rewinding the adversary is not allowed. (The definition does not fully specify the details of the interaction between the interface and $\mathcal{A}$.) A protocol for evaluating a function is secure if it is at least as secure as the trivial protocol for evaluating the function in an ideal model similar to the one here. To allow for secure sequential composition, the definition allows the adversary to receive additional auxiliary information upon corrupting a party. In addition it requires the protocol to be post-protocol corruptible. That is, the adversary should be able to respond to "any sequence of post-execution corruption requests" with the internal data of the relevant parties.

Disallowing rewinding is a considerable limitation, especially in the computational setting. (See Remark 3 in Section 4.2). An additional weakness of this definition is that, unlike here, $\mathcal{A}'$ is not required to be as efficient as $\mathcal{A}$. (See Remark 1 in Section 4.2). Compared with our notion of an environment machine, the requirement of post-protocol corruptibility has two main drawbacks. First, it does not take into account the fact that the post-execution corruption requests can be adaptive and depend on the execution of the protocol itself and on the data learned from previous corruptions (rather than being fixed in advance). Second, this formalization does not generalize to the computational setting, where the corruption requests must be generated by a PPT machine (see Remark 1 in Section 6.2).

The definition of Canetti et. al. The definitions of [C95, CFGN96] differs from the one here in the following aspects. First, as in [B91], these definitions require the ideal-process adversary to operate via black-box simulation with no rewinds. Next, they do not incorporate auxiliary input in the definition, and do not include an environment machine. Finally, these definitions have additional structure whose purpose is to formalize the amount of internal deviation from the protocol allowed to *uncorrupted* parties. That is, first they define what it means for a protocol π' to be a semi-honest protocol for a known protocol π. (Essentially, π' allows even uncorrupted parties to internally deviate from π, as long as this deviation is undetectable by the other parties.) Next they say that π is secure only if any semi-honest protocol for π is secure.