Security and Composition of Multi-party Cryptographic Protocols

Ran Canetti*

October 27, 1998

Abstract

We present general definitions of security for multi-party cryptographic protocols.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. In particular, 'black-box simulation' is no longer required. The composition operation is essentially the natural 'subroutine substitution' operation, formalized by Micali and Rogaway.

We consider several 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 preserved under composition.

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

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

Contents

1 Introduction

Designing protocols that are *secure* against malicious attacks 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 outputs (according to some pre-defined functionality), together with a set of secrecy requirements regarding the local data.

A general study of the security of cryptographic protocols started with the pioneering works of Yao [40, 41], and Goldreich, Micali, and Wigderson [26]. 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 (e.g., [6, 14, 38, 28, 36]), as well as protocols for more specific tasks (e.g., [17, 21, 37]).

In sharp contrast to the great advent in constructing secure protocols, our understanding of the very notion of security of protocols progresses more slowly. The first works in this field (and in particular [40, 41, 26]) contain only an intuitive exposition of this notion. Several general definitions of security of protocols subsequently appeared (most notably [28, 34, 2], the second one being considerably more comprehensive). While the general approach of these definitions is roughly the same, the definitions differ from each other in several substantial ways.

Indeed, while the notion of "secure cryptographic 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 appearing in the literature, including most of the constructions mentioned above, have never been rigorously proven secure.[1]

This paper aims at improving our understanding of the nature of secure computation and our ability to prove security of cryptographic protocols. We build on the definitional approach of [7] that seems convenient and flexible. (The approach of [7] is more recent and builds on [34, 28, 2]. It is closest to the approach of [2].) In particular, this approach has been used in a number of quite varied settings, e.g. [5, 9, 11, 4, 12, 13]. As a first step, we somewhat modify and simplify the [7] definitions, and present them in a clear and concise way.

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 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, and others.

We show that our notion of security provides this guarantee, in several important settings and with respect to a natural and modular composition operation suggested in [34]. (Previously only the [34] definition 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 yet full proofs of security of known protocols, such as [26, 6, 9, 4].

The definitional approach. Let us sketch our definitional approach (following [34, 2, 7]). We first formulate an 'ideal process' that captures the desired *functionality* of the task at hand and, in particular, rules out any unwanted behavior. The ideal process involves no communication among the parties; instead, all parties hand their inputs to an incorruptible "trusted party", who locally

[1] Recently a detailed exposition and analysis of [26] has been made available in [23].

computes the desired outputs and hands them back to the parties. Thus, in the ideal process the adversary 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 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 [29] to define security of encryption functions and in [30] to define Zero-Knowledge protocols. Yet, the approach 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 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, in real-world situations adversaries may often be adaptive in nature.

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. The secure channels setting is often regarded as a 'stepping stone' on the way to obtaining protocols in the (more realistic) computational setting. In Appendix A we discuss several other adversary models and security concerns.

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 [40, 26, 3, 28, 23] for definitions and protocols for the case of dishonest majority.

Relation to the [2, 7] definitions. Although [2, 7] share the same definitional approach, their actual formalizations suffer from shortcomings and unnecessary complexities that are 'cleaned up' here. Let us highlight two such issues. 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. Furthermore, it was required that the simulated adversary is run only once and is not 'rewinded'. (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 restriction seems essential for a good treatment of the computational model.

Another modification, relevant only to the case of adaptive adversaries, is the treatment of corruption requests that occur *after* the protocol is completed. (Such requests may occur when the adversary interacts with other protocols run by the parties, and accommodating these requests seems essential for secure composition.) We model these requests by introducing an additional entity, called a post-execution corruptor, that represents the external environment and interacts with the adversary after the protocol terminates. This modeling seems to better represent the effect of the external environment on the current computation; in particular it allows us deal with composition of protocols even for the case of computationally bounded adversaries.

Modular composition. When designing a protocol for some task, we want to be able to break down the task to 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 [34]. 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. For this purpose, we prove a general theorem informally stated as follows:

Theorem (modular composition, informal). *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 computes g.*

Several other composition operations for 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 regard modular composition as the main general tool for modular protocol design. This paper concentrates on the non-concurrent case, where only a single subroutine invocation is in execution at any given time. The more general case, where several subroutine invocations may be running at the same time, will be dealt with in [8], and seems to require a stronger notion of security of protocols.

We formalize and prove the above theorem for the settings described above (i.e., non-adaptive, adaptive, passive, active adversaries in the secure channels and computational settings). In particular, this is the first time a composition theorem is even *stated* in the computational setting with respect to *any* definition.

Other related work. Micali and Rogaway present a detailed and comprehensive definition of secure function evaluation, for the case of *deterministic* functions, in *the secure channels setting* [34]. In this setting, they define modular composition of protocols and state a composition theorem. Our statement of the composition theorem is essentially taken from there.

The [34] definition involves some additional requirements from secure protocols that are not made in our definition. It was previously believed that some of these requirements were *necessary* for proving that modular composition preserves security. Here we show that this is not the case. We remark that the various versions of [34] contain many important and enlightening observations, discussions, and examples regarding secure multi-party protocols. We have benefited a lot from reading this work, as well as from attending [35].

On top of a general definition of security of protocols, Beaver [2] states a 'sequential composition' theorem. 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 computes the composed function, as long as all intermediate results are part of the output. The proof requires 'black-box simulation' with no rewinds.

Yet another approach is taken by Goldwasser and Levin [28]. In a nutshell, they define a notion of 'robustness' (or, resilience to faults) of protocols that is to a large extent independent from the computed functionality. A protocol securely computes some functionality if it is robust and in addition it computes the functionality whenever the adversary is limited to 'legal' behavior. In all, their approach is not far from the approach taken here. In particular, theirs is the only definition prior to [7] that addresses the computational model (yet, they do not discuss secure composition). We briefly describe the [34, 2, 28, 7] definitions in Appendix B.

Goldreich [23] presents a detailed exposition and proof of the [26] general construction, 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, [23] does not present general purpose composition theorem for active adversaries, but rather composes the constructed protocols "on the fly".

A (somewhat weaker) notion of security for the case of deterministic functions, non-adaptive, passive adversaries in the secure channels setting is studied by Chor and Kushilevitz [15, 32]. Reducibility of protocols w.r.t. the [15, 32] notion of security is discussed in [33]. Yet the [33] notion of reducibility is different than the one here in that there no communication is allowed in the high-level protocol except for joint invocations of an oracle for the reduced-to function.

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 sketch some extensions of the definitional approach to other settings and concerns. In Appendix B we briefly discuss other definitional efforts.

2 Defining secure protocols: The general paradigm

The paper presents definitions of secure protocols in several adversary models. (We consider passive, active, non-adaptive and adaptive adversaries, in either the secure channels or the computational setting.[2]) Yet, the definitional approach is the same for all these settings, with 'local modifications' made to accommodate the setting at hand. In the rest of this section we motivate and sketch this general definitional approach.

As mentioned in the introduction, we concentrate on the case of honest majority. In addition, the definitions are presented for the problem of multi-party function evaluation. That is, the parties are given inputs and should produce outputs according to a given specification, casted as a (possibly probabilistic) function of their inputs.[3] This is a very general formalization that captures a large class of cryptographic tasks. In Appendix A we sketch how the definition can be extended to capture on-going, *reactive* tasks, where inputs arrive at different times during the computation, and where new inputs depend on previous behavior of the network.

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 anything from interacting with the parties on top of 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 (as in the case of probabilistic encryption or zero-knowledge [29, 30]), but it is not clear at this point in what setting the simulator should operate and what should be the requirements from it.

A naive approach towards defining security may proceed by separately requiring correctness and secrecy. Yet, as observed in [34], this decomposition is problematic since the two requirements are "intertwined": On 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.

[2]Other variations of these settings may of course be interesting. For instance, many works assume an *authenticated broadcast channel.* 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., [19, 10, 21, 23, 37]). The definitions can be easily adapted to these settings.

[3]Probabilistic function can be regarded as functions that have an additional argument, taken to be the random input.

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[4]) 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 [34]. Additional problems arise from dealing with *randomized* 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 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 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., 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., passive, active, non-adaptive, adaptive, etc.). At the end of the computation the uncorrupted parties output whatever is specified in their protocol. The corrupted parties output whatever is instructed by the adversary.

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 whatever is instructed by the adversary.[5]

We say that a protocol π for computing a function is secure if it "emulates" the ideal evaluation of the function, in the sense that any effect on the real-life computation achieved by a real-life

[4]This is so since each party can infer the input of the other party from its own input and the function value.

[5]In both the real-life computation and ideal process the outputs of the corrupted parties represent the information gathered by the adversary. Yet, in the ideal process 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.

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 contains the outputs of all parties. 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 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.)

We stress that combining the adversary's output and the output of the uncorrupted parties into one random variable (whose distribution is compared in the two interactions) is an essential part of the definition. In particular it guarantees that the "intertwined" secrecy and correctness requirements are satisfied. For instance, the real-life adversary cannot influence the outputs of the uncorrupted parties more than it could in the ideal process, nor can it learn information than is unavailable in the ideal process. (See also Remark 2, Section 4.2.)

3 Preliminaries

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

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 $X \stackrel{\mathrm{d}}{=} Y$) if for all sufficiently large k and all a we have that distributions $X(k,a)$ and $Y(k,a)$ are identical.*

Overloading notation, we will also use $X(k,a) \stackrel{\mathrm{d}}{=} Y(k,a)$ to denote that distributions $X(k,a)$ and $Y(k,a)$ are identical.

Definition 2 (Statistical indistinguishability) *We say that two distribution ensembles X and Y are* statistically indistinguishable *(and write $X \stackrel{\mathrm{s}}{\approx} Y$) if for every positive polynomial p, and for all sufficiently large k and all a we have that*

$$\mathrm{SD}(X(k,a), Y(k,a)) < \frac{1}{p(k)}$$

where SD *denotes statistical distance, or total variation distance.*

Definition 3 (Computational indistinguishability [29, 39]) *We say that two distribution ensembles X and Y are* computationally indistinguishable *(and write $X \stackrel{\mathrm{c}}{\approx} Y$) if for every positive polynomial p, for every algorithm 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 that*

$$|\text{Prob}(D(1^k, a, w, X(k,a)) = 1) - \text{Prob}(D(1^k, a, w, Y(k,a)) = 1)| < \frac{1}{p(k)}.$$

Note that Definition 3 gives the distinguisher D access to an arbitrary auxiliary information string w (thus forcing the definition to be a non-uniform complexity one). It is stressed that w is independent from the random choices of X and Y.

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 restrict ourselves to functions where the length of the random input is some fixed value determined by the security parameter. (Figuratively, the function first truncates the random input to the appropriate length.) Typically, the lengths of the inputs and the outputs will also be determined by the security parameter. In addition, we will be interested in functions where the lengths of the inputs and outputs are bounded by a polynomial in the security parameter. See [23] for a more complete discussion of conventions regarding such functions.

Intuitively, n-party functions are interpreted as follows: Each party P_i (out of $P_1, ..., P_n$) has input $x_i \in \{0,1\}^*$, and wishes to compute $f(k, \vec{x}, r_f)_i$ where $r_f \stackrel{\text{R}}{\leftarrow} \{0,1\}^t$ and t is the value determined by the security parameter.[6] 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

Although non-adaptive security (i.e., security against non-adaptive adversaries) is considerably weaker than than adaptive security (see Remark 1 in Section 5.2), we choose to 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., [26, 19]) are known to be secure only against non-adaptive adversaries (see [23]). 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 the adversary learns only messages sent to the corrupted parties. In Section 6 we show how the treatment is adapted to settings where the adversary is restricted to probabilistic polynomial time and no secure channels exist.

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.

[6] Here $e \stackrel{\text{R}}{\leftarrow} D$ means that element e is drawn uniformly at random from domain D, and $f(k, \vec{x}, r_f)_i$ denotes the ith component of $f(k, \vec{x}, r_f)$.

Following the outline presented in Section 2, we first describe 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 Turing machines, where the ith machine is associated with the ith party, P_i.[7] 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.

A (non-adaptive) t-limited real-life adversary, $\mathcal{A}$, is a pair $\mathcal{A} = (C, \hat{\mathcal{A}})$, where $C \subset \{P_1 ... P_n\}$ with $|C| \leq t$ is the set of corrupted parties and $\hat{\mathcal{A}}$ is an interactive (computationally unbounded) Turing machine describing the behavior of the corrupted parties. (Abusing notation, we usually use $\mathcal{A}$ to denote both $\mathcal{A}$ and $\hat{\mathcal{A}}$.) Adversary $\mathcal{A}$ starts off with the inputs of the corrupted parties, their random inputs, some auxiliary input, and the security parameter k.[8]

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.[9] First the uncorrupted parties send their messages of this round, as described in the protocol. The messages addressed to the corrupted parties become known to the adversary. 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.

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, $\perp$, specifying that they are corrupted. (Note that, in some sense, 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

[7] We regard n, the number of parties, as an independent parameter from the security parameter, k. This allows discussing cases where the number of parties 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.

[8] The auxiliary input is a standard tool that allows us to prove the composition theorem. Intuitively, the auxiliary input captures information gathered by the adversary from other interactions occurring before the current interaction. (Auxiliary inputs were first introduced in [27], in the context of Zero-Knowledge proofs. See more discussion there, as well as in [22].)

We remark that the adversary, being computationally unbounded, need not be randomized. (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 randomized adversaries with comparable complexity.) Yet, we find it more appealing conceptually to formulate the definition in terms of randomized adversaries.

[9] The description here captures a *fully connected, 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.) This distinction is significant only in the case of active adversaries.

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.)

adversary's output consists of its entire view.)[10]

We use the following notation. Let $\text{ADV}_{\pi,\mathcal{A}}(k, \vec{x}, z, \vec{r})$ 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 $\text{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, z, \vec{r})_i$ denote the output of party P_i after running protocol π on input $\vec{x}$, random input $\vec{r}$, security parameter k, and with a real life adversary $\mathcal{A}$ having auxiliary input z. Recall that if P_i is uncorrupted then this is the output specified by the protocol; if P_i is corrupted then $\text{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, z, \vec{r})_i = \perp$. Let

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

Let $\text{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, z)$ denote the probability distribution of $\text{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, z, \vec{r})$ where $\vec{r}$ is uniformly chosen. Let $\text{EXEC}_{\pi,\mathcal{A}}$ denote the probability ensemble $\{\text{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, z)\}_{k\in\mathbf{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$.[11]

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 compute $f(k, \vec{x}, r_f)_1, \ldots, f(k, \vec{x}, r_f)_n$, where $r_f \stackrel{\text{R}}{\leftarrow} \{0,1\}^t$ and t is the 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 a pair $\mathcal{S} = (C, \hat{\mathcal{S}})$, where $C \subset \{P_1 \ldots P_n\}$ is the set of corrupted parties and $\hat{\mathcal{S}}$ is an interactive (computationally unbounded) Turing machine describing the behavior of the corrupted parties.[12] Also here we usually interchange $\hat{\mathcal{S}}$ and $\mathcal{S}$. Adversary $\mathcal{S}$ starts off with the inputs of the corrupted parties, random input, some auxiliary input, and the security parameter k.[13] In addition, there is an (incorruptible) trusted party, T. The ideal process proceeds as follows.

Input substitution: If the ideal-process-adversary $\mathcal{S}$ is active then it may alter the inputs of the corrupted parties 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 this step does not take place, 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 $r_f \stackrel{\text{R}}{\leftarrow} \mathcal{R}_f$, and hands each P_i the value $f(k, \vec{y}, r_f)_i$.[14]

Output: Each uncorrupted party P_i outputs $f(k, \vec{y}, r_f)_i$, and the corrupted parties output $\perp$. 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$\}$.

[10] In Section 2 we used a slightly different convention, where the adversary does not have an output and the corrupted parties output whatever the adversary instructs. The two conventions are clearly equivalent (in the sense that they let the same information appear in the outputs). Yet the convention used here, while slightly unnatural, is cleaner since the adversary's output (i.e., its view) appears only once, at a fixed location.

[11] Let $\langle\vec{x}, z\rangle$ denote some natural encoding of $\vec{x}, z$ as a single string.

[12] As seen in the sequel, there is no need to explicitly limit the size of C in the ideal process. The definition guarantees that the identities of the corrupted parties in the ideal process are distributed identically to the identities of the corrupted parties in the real life model.

[13] In contrast with the real-life adversary, it is essential that the ideal-process adversary be randomized. This holds even in our non-uniform complexity setting.

[14] 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.

Let $\text{ADV}_{f,\mathcal{S}}(k,\vec{x},z,\vec{r})$, where $\vec{r} = \langle r_f, r\rangle$, denote the output of ideal process adversary $\mathcal{S}$ on security parameter k random input r and auxiliary input z, and when interacting with parties having input $\vec{x} = x_1 \ldots x_n$, and with a trusted party for computing f with random input r_f. Let the $(n+1)$-vector

$$\text{IDEAL}_{f,\mathcal{S}}(k,\vec{x},z,\vec{r}) = \text{ADV}_{f,\mathcal{S}}(k,\vec{x},z,\vec{r}), \text{IDEAL}_{f,\mathcal{S}}(k,\vec{x},z,\vec{r})_1 \ldots \text{IDEAL}_{f,\mathcal{S}}(k,\vec{x},z,\vec{r})_n$$

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

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 real-life adversary $\mathcal{A}$ there should exist an ideal-process adversary $\mathcal{S}$, such that $\text{IDEAL}_{f,\mathcal{S}} \stackrel{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A}}$. Spelled out, this requirement means that for any large enough value of the security parameter k, for any input vector $\vec{x}$ and any auxiliary input z, the global outputs $\text{IDEAL}_{f,\mathcal{S}}(k,\vec{x},z)$ and $\text{EXEC}_{\pi,\mathcal{A}}(k,\vec{x},z)$ should be identically distributed.[15] We remark that in [34, 2, 7] the notion of emulation is more restrictive. It was speculated that this is needed in order for the composition theorem to work. Here we show that this is not the case. See Remark 3 in Section 4.2 for more discussion.

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 inevitable. See Remark 1 in Section 4.2.

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 computes 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*

$$\text{IDEAL}_{f,\mathcal{S}} \stackrel{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A}} \tag{1}$$

If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries then we say that π non-adaptively, t-privately computes 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 makes sense only in the case where the adversaries are restricted to probabilistic polynomial time.)

4.2 Discussion

This section contains further discussion on Definition 4.

[15] 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.

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 statement: "whatever gain the adversary obtains from interacting with parties running π, could have also been obtained, *in comparable computational effort,* in an ideal setting where a trusted party is available". Thus, a protocol that allows the real-life adversary to obtain some gain "for free", where obtaining this gain in the ideal process requires considerable computational resources, offers only a weak form of security. Consequently, a definition that does not limit the computational power of the ideal process adversary is considerably weaker. (We remark that this weaker definition may still be of some interest for studying purely information-theoretic aspects of secure computation.)

Let us illustrate this distinction via the following 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 computing f: Party A announces x, party B announces y, and both parties compute $f(x,y)$. 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, if π is secure in the secure channels setting according to the definition here (and a secure encryption scheme is used) then π' is secure in the computational setting. However, if the above, weaker notion of security is used then this transformation does not necessarily work.

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 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 [34]) are insecure even under this weakened definition.

Nonetheless, we argue that requirements (a) and (b) must be combined, being two facets of the same computation. That is, the two *entire* $(n+1)$-vectors describing the global outputs of the two computations must be identically distributed, and does not suffice to require only that the two relevant projections of the global outputs are identically distributed. This point is demonstrated via an example, as follows. Consider the following functionality for two parties A and B: both parties have empty input and A should output a random bit. 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.[16]

Put in other words, the above example highlights an additional weakness of separately requiring correctness and secrecy, 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.*

Remark 3: On Black-Box simulation. In [34, 2, 7] the notion of emulation is more restrictive. There, it is required that the ideal-process adversary be restricted to having only black-box access to the real-life adversary. In addition, the adversary can be run only once and is never 'rewinded'. These restrictions are quite limiting. In particular, in the computational setting they prohibit usage of zero-knowledge protocols within secure protocols. (This is so since proving zero-knowledge requires rewinding the adversary.)

It was speculated in [7, 9] (and, implicitly, also in [34, 2]) that restricting the ideal-process adversary to this type of black-box simulation is needed in order to prove a general composition theorem. In this work we show that the modular composition theorem holds 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 [27]. 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 $\mathcal{A}$.) 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 and require that there exists an *efficient transformation* from real-life adversaries to the corresponding ideal-process adversaries. Yet, the above argument shows that such strengthening is unnecessary.

Remark 5: On the duration of the computation. Definition 4 does not impose any requirement on the duration of the computation (in terms of number of rounds), beyond the requirement that the duration will not reveal any information that is not avaliable in the ideal process. In particular, The definition allows secure protocols where the number of communication rounds is related to the public portion of the outputs of the parties in an arbitrary way. For instance, a

[16]We sketch a proof: The interesting case is where B is corrupted; in this case, for each real-life adversary (that controls B) construct the following ideal process adversary $\mathcal{S}$: Since $\mathcal{S}$ interacts with the ideal process, it does not know the output of A, denoted b. Yet, $\mathcal{S}$ will simulate an independent random bit b' sent from A to B. The bit b' will be different (with probability one half) from the the output of A in this execution, thus Equation (1) will not be satisfied. Yet, as long as the outputs of A and B are considered separately the simulation is valid.

protocol for jointly generating an unbiased random bit may be considered secure even if, say, the parity of the number of rounds lapsed until the output is known equals the value of the output bit.

In cases where it is important to specify the relations between the output of the computation and its duration, the definition may be modified as follows. First, the computed functionality will specify also the number of rounds to be lapsed until the output is known. Next, we add a 'round counter' to the ideal process. The parties hand their inputs to the trusted party at round 0; the trusted party delays the handing of the outputs to the parties for the specified number of rounds. Finally, the parties will include the number of rounds lapsed in their outputs.

Remark 6: Other adversary structures. Definition 4, as well as the rest of the paper, deals with t-limited adversaries, where t is some threshold; that is, it is assumed that the 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., [31, 16]), both in the non-adaptive and the adaptive cases.

4.3 Modular composition: The non-adaptive case

Recall that we want to break down a given task (i.e., a protocol problem) to 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 theorem stated in the Introduction. We do this for the non-concurrent case, where at most one subroutine call is made at any communication round. In this section we concentrate on non-adaptive adversaries in the secure channels setting.

The hybrid model. We start by specifying the model for computing an n-party function g with the assistance of a trusted party for computing n-party functions $f_1, ..., f_m$, and define secure protocols in that model. The model, called the hybrid model with ideal access to $f_1, ..., f_m$ (or in short the $(f_1, ..., f_m)$-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 computing $f_1, ..., f_m$. 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, all parties hand their f-inputs to T (party P_i hands x_i^f), and 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). 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.

Let $\text{EXEC}^{f_1,...,f_m}_{\pi,\mathcal{A}}(k, \vec{x}, z)$ denote the random variable describing the output of the computation in the $(f_1, ..., f_m)$-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 $\text{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, z)$ in Section 4.1. (We stress that here π is a hybrid of a real-life protocol with ideal evaluation calls to T.) Let $\text{EXEC}^{f_1,...,f_m}_{\pi,\mathcal{A}}$ denote the probability ensemble $\{\text{EXEC}^{f_1,...,f_m}_{\pi,\mathcal{A}}(k, \vec{x}, z)\}_{k\in\mathbf{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$.

Protocols for securely computing a function g in the $(f_1, ..., f_m)$-hybrid model are defined via the usual comparison to the ideal process for g:

Definition 5 *Let $f_1, ..., f_m, g$ be n-party functions and let π be a protocol for n parties in the $(f_1, ..., f_m)$-hybrid model. We say that π* non-adaptively t-securely computes g in the $(f_1, ..., f_m)$-hybrid model *if for any non-adaptive t-limited adversary $\mathcal{A}$ (in the f-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*

$$\text{IDEAL}_{g,\mathcal{S}} \stackrel{\text{d}}{=} \text{EXEC}^{f_1,\dots,f_m}_{\pi,\mathcal{A}}. \tag{2}$$

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

Replacing an ideal evaluation call with a subroutine call. Next we describe the 'mechanics' of replacing an ideal-evaluation call of protocol π with a call to an appropriate subroutine real-life protocol, ρ. This is done in a straightforward way: the code of π within each party is changed so that the relevant call to T is replaced with an invocation of ρ. The value to be sent to T is used as local input (of the party) to ρ; in addition, in each invocation of ρ the party uses fresh random input. While protocol ρ is running, protocol π remains inactive. Once the execution of ρ is completed the local output is treated as the value returned by T, and the execution of π resumes. We assume that all parties terminate protocol ρ at the same round.[17] Let $\pi^{\rho_1\cdots\rho_m}$ denote protocol π (originally designed for the $(f_1 \dots f_m)$-hybrid model) where each ideal evaluation call to f_i is replaced by a call to ρ_i.

We stress that the above description implies that the internal state of protocol $\pi^{\rho_1\cdots\rho_m}$ within some uncorrupted party, immediately after some ρ_i has terminated with output v, is identical to internal state of π immediately after receiving value v from the trusted party for the corresponding invocation of f_i.

Theorem 6 (non-adaptive modular composition) *Let $t < n$, and let $f_1,\dots,f_m,g$ be n-party functions. Let π be an n-party protocol that non-adaptively t-securely (resp., t-privately) computes g in the $(f_1,\dots,f_m)$-hybrid model, and assume that no more than one ideal evaluation call is made at each round. Let $\rho_1,\dots,\rho_m$ be n-party protocols that non-adaptively t-securely (resp., t-privately) compute $f_1,\dots,f_m$, respectively. Then the protocol $\pi^{\rho_1,\dots,\rho_m}$ non-adaptively t-securely (resp., t-privately) computes g.*

4.4 Proof of Theorem 6

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 calls to T is treated at the end of the proof. (This extension is straightforward, as long as at most one call to T is made at each round.)

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).

[17] The assumption that all parties terminate at the same round is an important one. Otherwise, some parties may resume executing the calling protocol where others still execute the subroutine protocol, and the non-concurrency condition is violated.

For concreteness, consider an example. Parties A, B, C wish to compute the following function, g: C should output the exclusive-or of the inputs of A and 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 gets the exclusive-or of the inputs of A and B. Protocol π for computing 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. The ideal evaluation of f is then replaced by a protocol ρ that securely computes f, but where A participates only in the first round. Now, if 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), then a corrupted B may be able to influence the output of C in ways that make protocol π^ρ insecure, although both π and ρ are secure.

4.4.1 Proof outline

Let $\mathcal{A}$ be a (non-adaptive) real-life adversary for protocol π^ρ. We wish to construct an ideal-process adversary $\mathcal{S}$ that 'simulates' the behavior of $\mathcal{A}$. (In the sequel we will call $\mathcal{S}$ a **simulator**.) That is, $\mathcal{S}$ should satisfy

$$\text{EXEC}_{\pi^\rho,\mathcal{A}} \stackrel{\text{d}}{=} \text{IDEAL}_{g,\mathcal{S}}. \tag{3}$$

A simplistic plan for carrying out this proof may proceed as follows. First, find a way to partition $\mathcal{A}$ into two separate modules: one module that operates "only against protocol ρ", and another module that operates "only against protocol π" in the f-hybrid model. Next use the security of ρ and the security of π to obtain ideal-process adversaries that (separately) simulate each one of these modules. Finally, somehow combine the two simulators to obtain a simulator for $\mathcal{A}$.

This plan seems hard to follow at first: It is not clear how to partition $\mathcal{A}$ in such a way; moreover, it is not clear how to combine the simulators for the two modules into a simulator for $\mathcal{A}$. We will, nevertheless, do something quite similar to this plan. More precisely, we proceed in three steps as follows:

1. 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 $\text{EXEC}_{\rho,\mathcal{A}_\rho} \stackrel{\text{d}}{=} \text{IDEAL}_{f,\mathcal{S}_\rho}$.
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. The security of ρ guarantees that this adversary has a simulator $\mathcal{S}_\pi$ such that $\text{EXEC}^f_{\pi,\mathcal{A}_\pi} \stackrel{\text{d}}{=} \text{IDEAL}_{g,\mathcal{S}_\pi}$.
3. Show that the simulator $\mathcal{S} = \mathcal{S}_\pi$ satisfies (3).

Let us sketch the above steps. Adversary $\mathcal{A}_\rho$ controls the same set of parties that $\mathcal{A}$ controls. It expects its auxiliary input to describe an internal state of of $\mathcal{A}$ 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, $\mathcal{A}_\rho$ continues a simulated run of $\mathcal{A}$ from the given state, and follows $\mathcal{A}$'s instructions. (The information seen by $\mathcal{A}_\rho$ is relayed to $\mathcal{A}$.) At the end of its interaction $\mathcal{A}_\rho$ outputs the current state of the simulated $\mathcal{A}$.

Adversary $\mathcal{A}_\pi$ controls the same set of parties. It 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 interaction between $\mathcal{S}_\rho$ and its trusted party for f is relayed by $\mathcal{A}_\pi$ to $\mathcal{A}_\pi$'s own trusted party for f. Recall that the output of $\mathcal{S}_\rho$ is a (simulated) 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.

Before sketching our analysis of $\mathcal{S} = \mathcal{S}_\pi$ 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 essentially independent of these arbitrary input values.

To show (3) is done in two steps:

$$\text{EXEC}_{\pi^\rho,\mathcal{A}} \stackrel{\text{d}}{=} \text{EXEC}^f_{\pi,\mathcal{A}_\pi} \stackrel{\text{d}}{=} \text{IDEAL}_{g,\mathcal{S}_\pi} \tag{4}$$

The rightmost equality follows immediately from the security of π in the f-hybrid model. The leftmost equality follows (after some work) from the security of protocol ρ.

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 $\mathcal{A}_\pi$, proving Claim 7 along the way. Next we analyze $\mathcal{S}$, showing (3).

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 input 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 $\text{IS}_{\pi,\mathcal{A}}(l,k,\vec{x},z,\vec{r})_0$ 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 $\vec{x}=x_1\ldots x_n$ 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 $\text{IS}_{\pi,\mathcal{A}}(l,k,\vec{x},z,\vec{r})_i$ denote the internal state of party P_i at round l of this execution. (If P_i is corrupted then $\text{IS}_{\pi,\mathcal{A}}(l,k,\vec{x},z,\vec{r})_i=\perp$.) Let

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

Let $\text{GS}_{\pi,\mathcal{A}}(l,k,\vec{x},z)$ denote the probability distribution of $\text{GS}_{\pi,\mathcal{A}}(l,k,\vec{x},z,\vec{r})$ 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 a valid internal state of party P at round l if z decodes to *some* internal state of P at round l. 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 $\mathcal{A}_\rho$. Given adversary $\mathcal{A}$, adversary $\mathcal{A}_\rho$ proceeds as follows. $\mathcal{A}_\rho$ controls the same set of parties controlled by $\mathcal{A}$. 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).

1. $\mathcal{A}_\rho$ ignores the input values of the corrupted parties.

2. $\mathcal{A}_\rho$ verifies that its auxiliary input, z^ρ, is a valid internal state of $\mathcal{A}$ at round $l_\rho - 1$. If z^ρ is not valid, then $\mathcal{A}_\rho$ halts with no output.

3. $\mathcal{A}_\rho$ runs $\mathcal{A}$ from internal state z^ρ, follows $\mathcal{A}$'s instructions, and hands the learned information to $\mathcal{A}$. More precisely:

   (a) Whenever $\mathcal{A}_\rho$ receives a message sent from an uncorrupted party to a corrupted party, it forwards this message to $\mathcal{A}$.

   (b) Whenever $\mathcal{A}$ instructs some corrupted party to send a message to an uncorrupted party, $\mathcal{A}_\rho$ sends this message as instructed.

4. Once protocol ρ is completed, $\mathcal{A}_\rho$ outputs the current internal state of $\mathcal{A}$.

It now follows from the security of protocol ρ that there exists an ideal-process adversary $\mathcal{S}_\rho$ such that $\text{IDEAL}_{f,\mathcal{S}_\rho} \stackrel{\text{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho}$. 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) randomized, since it should generate a distribution ensemble that is equally distributed as $\text{EXEC}_{\rho,\mathcal{A}_\rho}$. 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. Let $\vec{x}^\rho|_0$ denote the vector obtained from $\vec{x}^\rho$ by replacing all the inputs of the corrupted parties with 0. Then, we have:

Claim 7 *For any value of the security parameter k, any input vector $\vec{x}^\rho$ and auxiliary input z^ρ we have*

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

Proof: We have argued above that

$$\text{EXEC}_{\rho,\mathcal{A}_\rho}(k, \vec{x}^\rho, z^\rho) \stackrel{\text{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho}(k, \vec{x}^\rho|_0, z^\rho).$$

However, $\text{IDEAL}_{f,\mathcal{S}_\rho}(k, \vec{x}^\rho, z^\rho) \stackrel{\text{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho}(k, \vec{x}^\rho, z^\rho)$, and $\text{IDEAL}_{f,\mathcal{S}_\rho}(k, \vec{x}^\rho|_0, z^\rho) \stackrel{\text{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho}(k, \vec{x}^\rho|_0, z^\rho)$. The claim follows. □

Construction of $\mathcal{A}_\pi$. Adversary $\mathcal{A}_\pi$ proceeds as follows.

1. $\mathcal{A}_\pi$ invokes $\mathcal{A}$ on its own input, auxiliary input and random input, and follows the instructions of $\mathcal{A}$ up to round $l-1$. (Recall that so far, both in π and in π^ρ the parties run π.) In addition, $\mathcal{A}_\pi$ keeps another piece of its 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 adversary $\mathcal{A}_\pi$ interacts with parties that call a trusted party for ideal evaluation of function f. In order to continue the run of $\mathcal{A}$, adversary $\mathcal{A}_\pi$ invokes simulator $\mathcal{S}_\rho$. The auxiliary input z^ρ for $\mathcal{S}_\rho$ is set to the current internal state of $\mathcal{A}$. The random input of $\mathcal{S}_\rho$ is set to r^ρ. Next, $\mathcal{S}_\rho$ is run as follows.

(a) $\mathcal{S}_\rho$ is given input values 0 for the corrupted parties. (Recall that the inputs of the corrupted parties do not affect the distribution of the global output of computing f with $\mathcal{S}_\rho$.)

(b) When $\mathcal{S}_\rho$ hands its trusted party the inputs of the corrupted parties and asks for the computed values of f, then $\mathcal{A}_\pi$ invokes its own trusted party for f with the same input values for the corrupted parties, and hands the answer 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, $\mathcal{A}_\pi$ runs $\mathcal{A}$ from internal state v, and returns to following $\mathcal{A}$'s instructions until the completion of protocol π.

4. Once protocol π is completed, $\mathcal{A}_\pi$ outputs whatever $\mathcal{A}$ outputs.

The security of $\mathcal{A}_\pi$ guarantees that there exists a simulator $\mathcal{S}_\pi$ such that $\text{IDEAL}_{g,\mathcal{S}_\pi} \stackrel{\text{d}}{=} \text{EXEC}^f_{\pi,\mathcal{A}_\pi}$. We let $\mathcal{S} = \mathcal{S}_\pi$.

Analysis of $\mathcal{S}$. Fix an input vector $\vec{x}$ and auxiliary input z for the parties and adversary, as well as some large enough value of the security parameter k. We show:

$$\text{EXEC}_{\pi^\rho,\mathcal{A}}(k,\vec{x},z) \stackrel{\text{d}}{=} \text{EXEC}^f_{\pi,\mathcal{A}_\pi}(k,\vec{x},z) \stackrel{\text{d}}{=} \text{IDEAL}_{g,\mathcal{S}_\pi}(k,\vec{x},z)$$

The rightmost equality follows from the security of π in the f-hybrid model. The leftmost equality is shown in three steps, as follows.

Step 1. Recall that l_ρ is the round where protocol π makes the ideal evaluation call, and protocol π^ρ invokes ρ. Then, until round $l_\rho - 1$ protocols π and π^ρ behave the same. That is, let $\vec{r}^\pi$ be some fixed random-input vector, shortened to include only the portion that is used in protocol π (both before and after the execution of ρ). We have:

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

Step 2. Let $\vec{x}^\rho$ denote the input values of the parties for protocol ρ, as determined in $\text{GS}_{\pi^\rho,\mathcal{A}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)$. (The input values of the corrupted parties are set to 0.) Let z^ρ denote the internal state of $\mathcal{A}$ at round $l_\rho - 1$ in this execution. Similarly, let $\vec{x}^f$ denote the values that the parties are to give to the trusted party for f, as determined in $\text{GS}_{\pi,\mathcal{A}_\pi}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)$, 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 Step 1 that $\vec{x}^\rho = \vec{x}^f|_0$ and $z^\rho = z^f$. (Note that we don't necessarily have $\vec{x}^\rho = \vec{x}^f|_0$.) From the security of ρ and Claim 7 we have that

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

Now, 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:

1. Let $\vec{r}^\rho$ be a random input vector for protocol ρ, and let $\vec{r}$ denote the combination of $\vec{r}^\pi$ and $\vec{r}^\rho$. (That is, $\vec{r}$ is an entire random-input vector for π^ρ.) Then, $\text{GS}_{\pi^\rho,\mathcal{A}}(l_\pi, k, \vec{x}, z, \vec{r})$ can be obtained from $\text{GS}_{\pi^\rho,\mathcal{A}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)$ and $\text{EXEC}_{\rho,\mathcal{A}_\rho}(k, \vec{x}^\rho, z^\rho, \vec{r}^\rho)$ via a (simple, deterministic) procedure. (Essentially, this procedure updates the internal states of the adversary and the parties.[18])

[18]More precisely, this procedure first modifies each internal state $\text{IS}_{\pi^\rho,\mathcal{A}}(l_\rho - 1, k, \vec{x}, z, \vec{r}^\pi)_i$ by adding $\text{EXEC}_{\rho,\mathcal{A}_\rho}(k, \vec{x}^\rho, z^\rho, \vec{r}^\rho)_i$ 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.

2. Let $\vec{r}^f$ be a random input vector for the ideal evaluation process of f, and let $\vec{r}^{\pi,f}$ denote the concatenation of $\vec{r}^\pi$ and $\vec{r}^f$. Then, $\text{GS}_{\pi,\mathcal{A}_\pi}(l_\rho, k, \vec{x}, z, \vec{r}^{(\pi,f)})$ can be 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 procedure as in the previous bullet.*

Now, let $\vec{r}^\pi$, $\vec{r}^\rho$ and $\vec{r}^f$ be randomly chosen in their domains. It follows that:

$$\text{GS}_{\pi^\rho,\mathcal{A}}(l_\pi, k, \vec{x}, z) \stackrel{\text{d}}{=} \text{GS}_{\pi,\mathcal{A}_\pi}(l_\rho, k, \vec{x}, z). \tag{5}$$

Step 3 From the resumption of protocol π until its conclusion, adversary $\mathcal{A}_\pi$ again follows 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 procedure to the corresponding sides of (5).

This completes the proof for the case of a single ideal evaluation call. We remark that the proof demonstrates the following slightly stronger statement. Let $\text{EXEC}_{\pi^\rho,\mathcal{A}}(k, \vec{x}, z)|\vec{r}^\pi$ (resp., $\text{EXEC}^f_{\pi,\mathcal{A}_\pi}(k, \vec{x}, z)|\vec{r}^\pi$) denote the distribution of $\text{EXEC}_{\pi^\rho,\mathcal{A}}(k, \vec{x}, z)$ (resp., of $\text{EXEC}^f_{\pi,\mathcal{A}_\pi}(k, \vec{x}, z)$) when the random input for π is fixed to $\vec{r}^\pi$. It follows that for any value of $\vec{r}^\pi$ we have

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

On multiple ideal evaluation calls. The case of multiple ideal evaluation calls is a straightforward extension of case of a single call. We first observe that the above proof holds even in the case where π^ρ is also a protocol in a hybrid model, and makes additional ideal evaluation calls — as long as no other call is made at the round where the current invocation of ρ occurs.

Next, consider a protocol π that t-securely computes function g in the hybrid model, such that π makes m ideal evaluation calls. Let ρ_i be a protocol that t-securely computes the function evaluated at the ith ideal call. (Neither the value m nor ρ_i need be fixed in advance.) We define $m+1$ intermediary protocols, where the jth protocol replaces each $i \leq j$, the ith ideal call with an invocation of the corresponding protocol, ρ_i. We stress that each of the protocol invocations uses independent, fresh random inputs.

Now, the proof for the case of a single ideal evaluation call can be used to show by induction on j that for all the intermediary protocols t-securely compute g (still in the hybrid model). Note that the mth intermediary protocol does not make any ideal evaluation calls; thus it is in fact a protocol that t-securely computes g in the real-life model.

4.4.3 Extensions of the proof

On the propagation of statistical distance. A somewhat relaxed version of Definition 4 allows the two sides of (1) to have negligible statistical distance. The composition theorem holds even for this case. In fact, it can be readily seen that the statistical distance is preserved in a very tight sense. That is, if the statistical distance between $\text{EXEC}_{\pi^\rho,\mathcal{A}}(k, \vec{x}, z)$ and $\text{IDEAL}_{g,\mathcal{S},}(k, \vec{x}, z)$ is δ (for some $\mathcal{A}$ and $\mathcal{S}$ and some values of k, $\vec{x}$ and z), then there exist values δ_1 and δ_2 such that $\delta_1 + \delta_2 = \delta$, and such that:

- The statistical distance between $\text{EXEC}^f_{\pi,\mathcal{A}_\pi}(k, \vec{x}, z)$ and $\text{IDEAL}_{g,\mathcal{S}_\pi}(k, \vec{x}, z)$ is δ_1.

- There exist values $\vec{x}^\rho, z^\rho, \vec{x}^f, z^f$ such that the statistical distance between $\text{EXEC}_{\rho,\mathcal{A}_\rho}(k, \vec{x}^\rho, z^\rho)$ and $\text{IDEAL}_{f,\mathcal{S}_\rho}(k, \vec{x}^f, z^f)$ is δ_2. (In fact, $\vec{x}^\rho, z^\rho, \vec{x}^f, z^f$ are the values defined in Step 2 of the analysis.

In the case of multiple ideal evaluation calls the above proof shows that the total statistical distance between the output of the composed protocol and the output of an ideal evaluation of g is the sum of the statistical distances that correspond to all the individual protocol invoked by the composed protocol.

A similar statement holds with respect to the adaptive case (Theorem 10).

On computational indistinguishability. An even more relaxed version of Definition 4 allows the two sides of (1) to be only computationally indistinguishable. The composition theorem holds even for this case. We defer the treatment of this case to Section 6.

On black-box simulation. The above proof can be extended to show the following additional result. Assume that the security of protocols π and ρ 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 simulators associated with π and ρ do not rewind the adversary, then the simulator associated with π^ρ does not rewind as well.

While the proof outline remains unchanged, this extension is quite tedious. The main obstacle is that now adversaries $\mathcal{A}_\rho$ and $\mathcal{A}_\pi$ do not have access to the internal state of $\mathcal{A}$. Instead, whenever $\mathcal{A}_\rho$ or $\mathcal{A}_\pi$ wishes to reach a certain internal state of $\mathcal{A}$ it obtains the appropriate prefix of the view of $\mathcal{A}$, and runs $\mathcal{A}$ on this prefix until the desired internal state is reached. (The view of $\mathcal{A}$ contains all the necessary information to uniquely specify a run of $\mathcal{A}$; this includes $\mathcal{A}$'s input, random input and auxiliary input, the security parameter, and all the messages received by $\mathcal{A}$ until the desired round.) We omit further details.

5 Adaptive adversaries

This section defines secure protocols, presents and proves the composition theorem for the case of adaptive adversaries. Adaptive security is a considerably stronger property than non-adaptive security. Also, in real world situations attackers often choose their target hosts in an adaptive way. Remark 1 in Section 5.2 presents an example that highlights the additional power of adaptive security.

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 our definition of secure protocols. All the remarks made in Section 4.2 and in footnotes throughout Section 4.1 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, 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 the same notion of emulation as in the non-adaptive case.

The real-life model. An n-party protocol π is a collection of n interactive, probabilistic Turing machines, where the ith machine is associated with the ith party, P_i. Each P_i has input x_i, random input r_i and auxiliary input z_i, as well as the security parameter k.[19] Each two parties are connected via a private channel. That is, every two parties may communicate so that no other party, nor the adversary, learns the exchanged information. (A more precise description of the computational process follows shortly.)

An adaptive t-limited real-life adversary $\mathcal{A}$ is another interactive (computationally unbounded) Turing machine that starts off with random input and some auxiliary input. It may choose to corrupt parties during the computation (see details below), based on the information known to it, and as long as at most t parties are corrupted altogether. Once a party is corrupted the party's input, random input, auxiliary input and the entire history of the messages sent and received by the party become known to the adversary. see Remark 2 in Section 5.2. 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.

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 more parties as described above, one by one in an adaptive way. 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.

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 $\perp$. In addition, adversary $\mathcal{A}$ outputs some arbitrary function of its view of the computation. The adversary view consists of its auxiliary and random inputs, followed by the corrupted parties' inputs, random inputs, auxiliary 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.) Let the early global output of the execution be the concatenation of the outputs of all parties and of the adversary.

Post-execution corruption. In the non-adaptive and non-concurrent case (i.e., when no two protocols are executed at the same round) auxiliary inputs suffice for capturing the influence of the external environment on the current protocol execution, and in particular suffice for proving the composition theorem. (Auxiliary inputs represent events that occur *before* the current execution started.) In the adaptive case the following additional way for the external environment to influence the current execution must be taken into consideration, even in the non-concurrent case: In general, the external environment can influence the adversary's choice of which parties to corrupt. While inactive during the current protocol execution, the external environment resumes activity as soon as the current execution is completed. From this point on, subsequent corruption requests may depend

[19] In the adaptive case we let each party have its own auxiliary input, rather than letting only the adversary have auxiliary input. This auxiliary input models the information gathered by the adversary when corrupting a party. (Typically, this information consists of data pertaining to other protocols run by the party.) We stress that the protocol run by the uncorrupted parties does not use the auxiliary input in any way. The auxiliary input is there only for the adversary to see upon corruption.

(in an adaptive way) both on external information and on information learnt from the completed execution. In addition, when corrupting a party the adversary sees the internal information from the completed execution. Consequently, the security requirement from an execution must take into account the information gathered by "the environment" from these *post-execution corruptions.*

We model this additional requirement as follows. We introduce an additional entity (representing the "external environment") to the real-life model. This entity, called a **post-execution corruptor** $\mathcal{P}$, is an interactive Turing machine that "kicks into action" when the execution of the protocol is completed. $\mathcal{P}$ starts off with an auxiliary input, some random input, and the security parameter; typically, the auxiliary input will suffice to emulate the entire external environment.[20] In addition $\mathcal{P}$ gets the early global output of the current execution. (That is, $\mathcal{P}$ gets the outputs of *all* parties.) Next, $\mathcal{P}$ and $\mathcal{A}$ interact in rounds, where in each round $\mathcal{P}$ first generates a `corrupt P_i' request for some P_i. Upon receipt of this request, $\mathcal{A}$ hands $\mathcal{P}$ some arbitrary information. (Intuitively, this information is interpreted as P_i's internal data.) The interaction continues until $\mathcal{P}$ halts, with some output. (Without loss of generality, this output can be $\mathcal{P}$'s entire view of its interaction with $\mathcal{A}$.) Once the interaction is completed, $\mathcal{P}$'s output is added to the global output. See further discussion on the role of the post-execution corruptor in Remark 3, Section 5.2.

We use the following notation. Let $\mathrm{POST}_{\pi,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z},\vec{r})$ denote $\mathcal{P}$'s output after interacting with parties running protocol π on input $\vec{x} = x_1 \ldots x_n$, random input $\vec{r} = r_P, r_0 \ldots r_n$, auxiliary input $\vec{z} = z_P, z_0, \ldots z_n$ and security parameter k as described above (z_P and r_P for $\mathcal{P}$; z_0 and r_0 for $\mathcal{A}$; x_i, z_i and r_i for party P_i). Let $\mathrm{ADV}_{\pi,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z},\vec{r})$ denote the output of real-life adversary $\mathcal{A}$ in this execution. Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z},\vec{r})_i$ denote the output of party P_i from this execution. If P_i is uncorrupted then this is the output specified by the protocol; if P_i is corrupted then $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z},\vec{r})_i = \perp$. Let

$$\begin{aligned}&\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z},\vec{r}) = \\ &\mathrm{POST}_{\pi,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z},\vec{r}), \mathrm{ADV}_{\pi,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z},\vec{r}), \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z},\vec{r})_1 \ldots \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z},\vec{r})_n\end{aligned}$$

denote the **global output** of the computation in the real-life model.[21] Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z})$ denote the random variable describing $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z},\vec{r})$ where $\vec{r}$ is uniformly chosen. Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{P}}$ denote the probability ensemble $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z})\}_{k\in\mathbf{N},\langle\vec{x},\vec{z}\rangle\in\{0,1\}^*}$.

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 $x_i \in D$ and auxiliary input z_i; no random input is needed. Recall that the parties wish to compute $f(k,\vec{x},r_f)_1, \ldots, f(k,\vec{x},r_f)_n$, where $r_f \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}^t$ and t is the value determined by the security parameter, and P_i learns $f(k,\vec{x},r_f)_i$. An **adaptive ideal-process-adversary** $\mathcal{S}$ is an interactive Turing machine that has auxiliary input z_0, random input r_0 and security parameter k.[22] In addition, there is an (incorruptible) **trusted party**, T, and a post-execution corruptor $\mathcal{P}$. The ideal process proceeds as follows.

First corruption stage: First, $\mathcal{S}$ proceeds in iterations. 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

[20] As in the case of the real-life adversary, it is not necessary to let $\mathcal{P}$ have random input. Yet, we find it more natural on the definitional level to let $\mathcal{P}$ be randomized.

[21] The difference between the *early global output* and the *global output* of the computation is that the global output contains the view of the post-execution corruptor.

[22] As in the non-adaptive case, It is essential for $\mathcal{S}$ to have random input, even in a non-uniform complexity setting. In particular, $\mathcal{S}$ will have to 'simulate' the randomness used by the uncorrupted parties in the real-life model.

corrupted its internal data (that is, its input and auxiliary input) become known to $\mathcal{S}$. Let B denote the set of corrupted parties at 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 T values chosen by $\mathcal{S}$ (based on the information gathered so far).

Let $\vec{b}$ be the $|B|$-vector of the altered inputs of 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 the vector $\vec{y}$ from the parties (y_i is received from P_i). Next, T chooses $r_f \stackrel{\text{R}}{\leftarrow} \mathcal{R}_f$, 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 $\mathcal{S}$'s random input and the information gathered so far (this information now includes also the values received from the trusted party).

Output: Each uncorrupted party P_i outputs $f(k, \vec{y}, r_f)_i$, and the corrupted parties output $\perp$. 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 auxiliary inputs and the resulting function values $\{f(k, \vec{y}, r_f)_i : P_i \text{ is corrupted}\}$. Let the **early global output** of the ideal process be the concatenation of the outputs of all parties and of the adversary.

Post-execution corruption: Once the outputs are generated, $\mathcal{S}$ engages in an interaction with post-execution corruptor $\mathcal{P}$, similar to the interaction of $\mathcal{A}$ with $\mathcal{P}$ in the real-life model. That is, $\mathcal{P}$ starts off with some auxiliary and random inputs and the early global output of the current execution of the ideal process. Next, $\mathcal{P}$ and $\mathcal{S}$ interact in rounds where in each round $\mathcal{P}$ 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.) The interaction continues until $\mathcal{P}$ halts. Once the interaction is completed, $\mathcal{P}$'s output is added to the global output.

Let $\text{POST}_{f,\mathcal{S},\mathcal{P}}(k, \vec{x}, \vec{z}, \vec{r})$, where $\vec{r} = r_P, r_0, r_f$, denote the view of post-execution corruptor $\mathcal{P}$ (with random input r_P and security parameter k), after interacting as described above with an ideal-process adversary $\mathcal{S}$ and with parties having input $\vec{x} = x_1 \dots x_n$ and auxiliary input $\vec{z} = z_P, z_0, ...z_n$, and with a trusted party for computing f with random input r_f. Let $\text{ADV}_{f,\mathcal{S},\mathcal{P}}(k, \vec{x}, \vec{z}, \vec{r})$, denote the output of adversary $\mathcal{S}$ from this execution. Let the $(n+2)$-vector

$$\text{IDEAL}_{f,\mathcal{S},\mathcal{P}}(k, \vec{x}, \vec{z}, \vec{r}) =$$
$$\text{POST}_{f,\mathcal{S},\mathcal{P}}(k, \vec{x}, \vec{z}, \vec{r}), \text{ADV}_{f,\mathcal{S},\mathcal{P}}(k, \vec{x}, \vec{z}, \vec{r}), \text{IDEAL}_{f,\mathcal{S},\mathcal{P}}(k, \vec{x}, \vec{z}, \vec{r})_1 \dots \text{IDEAL}_{f,\mathcal{S},\mathcal{P}}(k, \vec{x}, \vec{z}, \vec{r})_n$$

denote the **global output** of the ideal process on inputs $\vec{x}, \vec{z}$, adversary $\mathcal{S}$, post-execution corruptor $\mathcal{P}$ and random inputs $\vec{r}$ as described above (party P_i outputs $\text{IDEAL}_{f,\mathcal{S},\mathcal{P}}(k, \vec{x}, \vec{z}, \vec{r})_i$). Let $\text{IDEAL}_{f,\mathcal{S},\mathcal{P}}(k, \vec{x}, \vec{z})$ denote the distribution of $\text{IDEAL}_{f,\mathcal{S},\mathcal{P}}(k, \vec{x}, \vec{z}, \vec{r})$ when $\vec{r}$ is uniformly distributed. Let $\text{IDEAL}_{f,\mathcal{S},\mathcal{P}}$ denote the probability ensemble $\{\text{IDEAL}_{f,\mathcal{S},\mathcal{P}}(k, \vec{x}, \vec{z})\}_{k \in \mathbb{N}, \langle \vec{x}, \vec{z} \rangle \in \{0,1\}^*}$.

Comparing computations in the two models. This part is identical to the non-adaptive case (Section 4.1). In particular, the remarks made there apply here as well.

Definition 8 (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 computes *f if for any adaptive t-limited real-life adversary $\mathcal{A}$, and any post-execution corruptor $\mathcal{P}$, there exists an adaptive ideal-process adversary $\mathcal{S}$ whose running time is polynomial in the running time of $\mathcal{A}$, such that*

$$\text{IDEAL}_{f,\mathcal{S},\mathcal{P}} \stackrel{\mathrm{d}}{=} \text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}. \tag{6}$$

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

Spelled out, Equation (6) means that for any large enough value of the security parameter k, for any input vector $\vec{x}$ and any auxiliary input z, the output vectors $\text{IDEAL}_{f,\mathcal{S},\mathcal{P}}(k, \vec{x}, \vec{z})$ and $\text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(k, \vec{x}, \vec{z})$ should be identically distributed.

5.2 Discussion

Remark 1: Adaptive versus non-adaptive security. It is easy to see that an adaptively secure protocol is also non-adaptively secure. In particular, observe that the definition of non-adaptive security (Definition 4) remains unchanged even if we allow the ideal-process adversary to corrupt parties in an adaptive way (as in the adaptive ideal process). This is so since the real-life adversary corrupts a fixed set of parties, and the distribution of the corrupted parties in the ideal process must be similar to that of the real-life computation.

Yet, non-adaptive security does not imply adaptive security. We demonstrate this point via an example, taken from [9]. 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 $m = \sqrt{t}$ parties, and shares its secret among these parties using an m-out-of-m sharing scheme. In addition D publicizes the set S.* Intuitively, this scheme lacks in security since S is public and $|S| \ll t$. 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 exponentially small probability. Consequently, this protocol is secure in the presence of non-adaptive adversaries (if a small error probability is allowed).

Other examples, omitted here, demonstrate adaptively *in*secure protocols that are non-adaptively secure even without any error probability.

Remark 2: Erasing local data. The amount of information seen by the adversary upon corrupting a party is an important parameter in the definition of adaptive security. In particular, the main technical difficulty in proving adaptive security of protocols is usually to show how the ideal-process adversary can 'fill out' the internal data of corrupted parties for the simulated real-life adversary.

By allowing the adversary access, upon corrupting a party in the real-life model, to all the information the corrupted party *ever knew* rather than only the current contents of the party's memory, we formalize our mistrust in the fulfillment of un-verifiable instructions such as locally erasing past information. Indeed, in a distributed setting where no party is thoroughly trusted it is unrealistic to base the security of a protocol on such instructions. In particular, a party may be reluctant to base its security on the good will of *other* parties to locally erase data. See [7, 9] for a more elaborate discussion on semi-honest parties that appear honest to an external observer but may internally deviate from their protocols. In particular, the [7, 9] definitions incorporate additional structure for better capturing this concern. For sake of simplicity, our definition does not contain this structure.

Yet, in some scenarios putting trust in internal erasures is both reasonable and necessary. An example is the case of threshold cryptography (see, e.g., [17]), and in particular proactive security [36, 10], where the parties are servers controlled by a single administrative authority, and use erasures to maintain security in the face of break-ins by outsiders. In such case we let $\mathcal{A}$ learn only the corrupted party's *current* state.

Remark 3: On post-execution corruption. We make some additional remarks and observations on the notion of post-execution corruption:

1. It is essential that the post-execution corruptor $\mathcal{P}$ be a separate entity from $\mathcal{A}$. In particular, $\mathcal{P}$'s auxiliary input may contain much more information than the auxiliary input of $\mathcal{A}$. (In fact, in the proof of the composition theorem below $\mathcal{P}$'s auxiliary input will include the inputs and random inputs of *all* parties to the entire computation — except for the random inputs to be used in the current execution.)

2. A consequence from the introduction of the post-execution corruptor $\mathcal{P}$ is that now the ideal-process adversary $\mathcal{S}$ is "forced" to generate its output in a somewhat interactive manner. That is, first $\mathcal{S}$ has to generate some output. Next, on each corruption request made by $\mathcal{P}$, $\mathcal{S}$ has to output an additional piece of data (presumably, this piece of data simulates the internal data of the relevant party). This interactivity is essential to the proof of the composition theorem.

3. A seemingly stronger alternative formulation of Definition 8 may require that a single ideal-process adversary $\mathcal{S}$ will satisfy (6) with respect to any post-execution corruptor. Yet, using the same argument as that of Remark 4 (Section 4.2), it can be seen that this alternative formulation is equivalent to the current one.

4. A notion of post-execution corruption appears also in [2]; yet there the formalization is different. In particular, in [2] there is no *algorithmic* post-execution corruptor; instead the security requirement is stated for "all sequences of post-execution corruptions". This formalization does not allow the corruption requests to be chosen adaptively; in addition it is not adequate for the computational setting (see Section 6).[23]

5.3 Modular composition: The adaptive case

We formalize the composition theorem (stated informally in the Introduction) for the case of adaptive adversaries in the secure channels setting. As in the non-adaptive case, we first define the hybrid model; next we describe how an ideal evaluation call is replaced by a subroutine protocol. Finally we state the composition theorems.

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

[23] To avoid confusion, we note that in [2] the term "post-protocol corruptor" refers to a different entity than the the "post-execution corruptor" introduced here. There it is a module within the adversary that receives external corruption requests and corrupts parties accordingly. Here it is an external entity, with different inputs, that interacts with the adversary and *generates* the corruption requests.

computing $f_1, ..., f_m$. 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, the protocol instructs each uncorrupted party which value to hand T. Yet, before activating T the adversary adaptively corrupts parties, learns the internal data of corrupted parties, and determines the values that corrupted parties hand T. (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^t 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.

Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}^{f_1,...,f_m}(k, \vec{x}, \vec{z})$ denote the random variable describing the global output of the computation in the $f_1, ..., f_m$-hybrid model with protocol π, adversary $\mathcal{A}$, security parameter k, post-execution corruptor $\mathcal{P}$, inputs $\vec{x}$ and auxiliary input $\vec{z}$ for the parties, analogously to the definition of $\text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(k, \vec{x}, \vec{z})$ in Section 5.1. (We stress that here π is not a real-life protocol and uses ideal calls to T.) Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}^{f_1,...,f_m}$ denote the probability ensemble $\text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}^{f_1,...,f_m}(k, \vec{x}, \vec{z})\}_{k\in\mathrm{N},\langle\vec{x},\vec{z}\rangle\in\{0,1\}^*}$.

Protocols for securely computing a function g in the $(f_1, ..., f_m)$-hybrid model are defined in the usual way:

Definition 9 *Let $f_1, ..., f_m, g$ be n-party functions and let π be a protocol for n parties in the $(f_1, ..., f_m)$-hybrid model. We say that π* adaptively t-securely computes g in the $(f_1, ..., f_m)$-hybrid model *if for any adaptive t-limited adversary $\mathcal{A}$ (in the f-hybrid model) and any post-execution corruptor $\mathcal{P}$ there exists an adaptive ideal-process adversary $\mathcal{S}$, whose running time is polynomial in the running time of $\mathcal{A}$, such that:*

$$\text{IDEAL}_{g,\mathcal{S},\mathcal{P}} \stackrel{\mathrm{d}}{=} \text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}^{f_1,...,f_m}. \tag{7}$$

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

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), and are not repeated here. 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) *Let $t < n$, and let $f_1, ..., f_m, g$ be n-party functions. Let π be an n-party protocol that adaptively t-securely (resp., t-privately) computes g in the $(f_1, ..., f_m)$-hybrid model, and assume that no more than one ideal evaluation call is made at each round. Let $\rho_1, ..., \rho_m$ be n-party protocols that adaptively t-securely (resp., t-privately) compute $f_1, ..., f_m$, respectively. Then the protocol $\pi^{\rho_1,...,\rho_m}$ adaptively t-securely (resp., t-privately) computes g.*

5.4 Proof of Theorem 10

As in the non-adaptive case, 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 treat only the case where $m = 1$ and 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 and is not repeated.

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 and is not repeated. Here we sketch the additional difficulties arising from the adaptivity of the adversaries and simulators. Full details appear in Section 5.4.2.

1. Adversary $\mathcal{A}_\rho$ has to accommodate corruption requests made by $\mathcal{A}$ during the execution of ρ. This is done as follows: whenever $\mathcal{A}$ requests to corrupt party P, adversary $\mathcal{A}_\rho$ corrupts P in its real-life interaction, and hands P's internal data to $\mathcal{A}$. In addition, $\mathcal{A}$ expects to see P's internal data from the (suspended) execution of protocol π. This data is obtained from P's auxiliary input. That is, $\mathcal{A}_\rho$ expects P's auxiliary input to describe P's internal data from protocol π. If this data is inconsistent or not in the right format then $\mathcal{A}_\rho$ halts.

2. A post-execution corruptor, $\mathcal{P}_\rho$, is constructed. Essentially, $\mathcal{P}_\rho$ mimics the behavior of $\mathcal{A}$ after the execution of $\mathcal{A}_\rho$ is completed, and issues the same corruption requests that $\mathcal{A}$ would. For this, $\mathcal{P}_\rho$'s auxiliary information contains the inputs and random inputs of all parties to the global interaction, except for the random inputs for ρ. When $\mathcal{P}_\rho$ gets the output of $\mathcal{A}_\rho$, it runs $\mathcal{A}$ on this output up to the completion of protocol ρ. Next, $\mathcal{P}_\rho$ continues the run of $\mathcal{A}$ from the completion of ρ up to the completion of π. Whenever $\mathcal{A}$ corrupts some party, $\mathcal{P}_\rho$ issues a matching corruption request for $\mathcal{A}_\rho$.

3. When adversary $\mathcal{A}_\pi$ runs the simulator $\mathcal{S}_\rho$, it has to accommodate $\mathcal{S}_\rho$'s corruption requests made during the ideal process. This is done as follows: whenever $\mathcal{S}_\rho$ requests to corrupt a party P, adversary $\mathcal{A}_\pi$ corrupts P in its hybrid model. P's input in the ideal process is set to the value that P hands its trusted party. P's auxiliary input is set to the internal data of P from protocol π. If P is corrupted *after* the ideal call to the trusted party is made, then P's output value is also given to $\mathcal{S}_\rho$.

4. 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, adversary $\mathcal{A}_\pi$ corrupts P in its hybrid model, and obtains P's internal data from protocol π. In addition, $\mathcal{A}$ expects to see party P's internal data from the execution of ρ. To obtain this data, $\mathcal{A}_\pi$ plays the role of the post-execution corruptor for $\mathcal{S}_\rho$, and asks $\mathcal{S}_\rho$ to corrupt P. Then, $\mathcal{A}_\pi$ combines P's internal data from protocol π and $\mathcal{S}_\rho$'s answer and obtains simulated internal data for P and protocol π^ρ. This data is handed to $\mathcal{A}$.

   In the analysis it is argued that the way in which $\mathcal{A}_\pi$ 'plays the role of the post-execution corruptor for $\mathcal{S}_\rho$' is valid. That is, the interaction between $\mathcal{S}_\rho$ and $\mathcal{A}_\pi$ is identical to an (imaginary) interaction between $\mathcal{S}_\rho$ and $\mathcal{P}_\rho$.

5.4.2 A Detailed proof

Let $\mathcal{A}$ be an adversary and let $\mathcal{P}$ be a post-execution corruptor (interacting with parties running π^ρ). First we present the constructions of $\mathcal{A}_\rho$, $\mathcal{P}_\rho$, and $\mathcal{A}_\pi$, proving Claim 11 along the way. Next we let $\mathcal{S} = \mathcal{S}_\pi$ show that $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}} \stackrel{\text{d}}{=} \text{IDEAL}_{g,\mathcal{S},\mathcal{P}}$.

Terminology. We use the same definitions of executions, internal states, global states, and running an adversary from an internal state as in the non-adaptive case (Section 4.4.2). Yet here these notions of-course refer to the adaptive model. In particular, we modify the notion of global state in two ways:

1. We augment the global state 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 accommodates the fact that, when corrupting a party, the adversary gets access to all the information that the party knew in the past. (See Remark 2 in Section 5.2 for more discussion on this definitional decision.) In particular, it guarantees that the global state of an execution (at any round) uniquely determines the continuation of the execution until its completion.

2. We augment the definition of global state to rounds after the execution of the protocol has been completed. In these rounds, the the global state contains also the internal state of the post-execution corruptor (and, in particular, the corruption requests made by the corruptor and the adversary's answers).

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

Construction of $\mathcal{A}_\rho$. Given adversary $\mathcal{A}$, adversary $\mathcal{A}_\rho$ proceeds as follows. Let l_ρ be the round where protocol π^ρ starts running protocol ρ. (This is the round where π calls T).

1. $\mathcal{A}_\rho$ verifies that its auxiliary input, z_0^ρ, is a valid internal state of $\mathcal{A}$ at round $l_\rho - 1$. If z_0^ρ is not valid then $\mathcal{A}_\rho$ halts with empty output.

2. $\mathcal{A}_\rho$ continues the above run of $\mathcal{A}$ from round l_ρ on, follows $\mathcal{A}$'s instructions, and hands the learned information to $\mathcal{A}$. More precisely:

   (a) Whenever $\mathcal{A}_\rho$ receives a message sent from an uncorrupted party to a corrupted party, it forwards this message to $\mathcal{A}$.

   (b) Whenever $\mathcal{A}$ instructs some corrupted party, P, to send a message to an uncorrupted party, $\mathcal{A}_\rho$ sends this message in the name of P as instructed. (If P is not yet corrupted in $\mathcal{A}_\rho$'s real-life environment then $\mathcal{A}_\rho$ corrupts P, and ignores P's internal history. We call these parties the a-priori corrupted parties.)

   (c) When $\mathcal{A}$ corrupts a new party, P_i, during the execution of protocol ρ, $\mathcal{A}_\rho$ proceeds as follows. First $\mathcal{A}_\rho$ corrupts P_i in its real-life model and obtains P_i's internal history for protocol ρ. In addition, $\mathcal{A}_\rho$ needs to provide $\mathcal{A}$ with P_i's internal history from the execution of protocol π. This information is obtained from P_i's auxiliary input, z_i^ρ. That is, $\mathcal{A}_\rho$ assumes that z_i^ρ describes P_i's internal history pertaining to the execution of protocol π up to round $l-1$. (If z_i^ρ is not consistent with the current run of $\mathcal{A}$ then $\mathcal{A}_\rho$ halts.) Note that P_i's internal history of protocols π and ρ are sufficient for obtaining P_i's internal history of π^ρ. $\mathcal{A}_\rho$ now hands this data to $\mathcal{A}$.

3. Once protocol ρ is completed, $\mathcal{A}_\rho$ outputs the current internal state of the simulated $\mathcal{A}$. Next, $\mathcal{A}_\rho$ is ready to interact with some post-execution corruptor as described in Section 5.1.

Construction of $\mathcal{P}_\rho$. Informally, $\mathcal{P}_\rho$ continues the run of $\mathcal{A}$ after the completion of protocol ρ, and generates the same corruption requests made by $\mathcal{A}$.

1. $\mathcal{P}_\rho$ starts off with a string q that describes the (early) global output of an execution of protocol ρ with adversary $\mathcal{A}_\rho$, and with an auxiliary input $z^\rho_{\mathcal{P}}$ that is expected to describe a global state of an execution of π^ρ on input vector $\vec{x}$ and auxiliary input vector $\vec{z}$ at round $l_\rho - 1$. (If either $z^\rho_{\mathcal{P}}$ or q are not valid then $\mathcal{P}_\rho$ halts with no output.)

2. $\mathcal{P}_\rho$ combines q and $z^\rho_{\mathcal{P}}$ into a string, v, that encodes a global state of an execution at the round, l_π, where protocol ρ is completed and the execution of π resumes. (Specifically, the adversary's internal state in v is taken from q; the internal history of an uncorrupted party P_i is the one from $z^\rho_{\mathcal{P}}$, after removing the random input intended for the execution of ρ, and adding P_i's portion of q. If in q party P_i is corrupted then it is corrupted in v as well.)

3. $\mathcal{P}_\rho$ continues the execution of the protocol with adversary $\mathcal{A}$ from global state v (playing the parts of *all* parties). Whenever adversary $\mathcal{A}$ corrupts a party, P_i, corruptor $\mathcal{P}_\rho$ issues a `corrupt` P_i' request to $\mathcal{A}_\rho$. Upon receipt of P_i's internal history, d_i, corruptor $\mathcal{P}_\rho$ combines d_i with P_i's internal history for π, obtains P_i's internal history for π^ρ, and hands this data to $\mathcal{A}$.

4. When π^ρ is completed and $\mathcal{A}$ is ready to interact with its own post-execution corruptor $\mathcal{P}$, corruptor $\mathcal{P}_\rho$ plays $\mathcal{P}$ for $\mathcal{A}$. (In order to do that, $\mathcal{P}$ has to receive the early global output of the current execution of π^ρ and some auxiliary input. $\mathcal{P}_\rho$ extracts the early global output from the current orchestrated interaction; $\mathcal{P}$'s random input and auxiliary input are part of $\mathcal{P}_\rho$'s auxiliary input. When $\mathcal{A}$ corrupts more parties, $\mathcal{P}_\rho$ proceeds as in the previous step.

5. $\mathcal{P}_\rho$ halts when $\mathcal{A}$ does, and outputs its auxiliary input, plus the entire conversation between $\mathcal{P}_\rho$ and the adversary it interacts with.[24]

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

Note that $\mathcal{A}_\rho$ is deterministic, since all of the randomness used by $\mathcal{A}$ is provided in the auxiliary input z^ρ_0. Yet, the simulator $\mathcal{S}_\rho$ is (inherently) randomized, since it should generate a distribution equal to $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}}(k,\vec{x},\vec{z})$. (In particular, it should mimic the randomness used by the uncorrupted parties running ρ.)

The special structure of $\mathcal{A}_\rho$ implies that $\mathcal{S}_\rho$ has an additional property, described as follows.[25] Note that $\mathcal{A}_\rho$ completely ignores the internal history of the a-priori corrupted parties. 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 $\vec{x}^\rho|_0$

[24]As seen in the analysis of $\mathcal{S}$, it is important that $\mathcal{P}_\rho$ outputs its conversation with the adversary. Instructing $\mathcal{P}_\rho$ to output its auxiliary input is important for the proof in the computational setting (Theorem 14).

[25]This property and the related discussion are very similar to the non-adaptive case. Yet we repeat the presentation in full, with the appropriate modifications to the adaptive case. A reader that is familiar with the non-adaptive case can safely skip to the construction of adversary $\mathcal{A}_\pi$.

and $\vec{z}^\rho|_0$ denote the vectors obtained from $\vec{x}^\rho$ and $\vec{x}^\rho$, respectively, by replacing all the entries that correspond to the a-priori corrupted parties with 0. Then, we have:

Claim 11 *For any input vector $\vec{x}^\rho$, and any auxiliary input vector $\vec{z}^\rho$ we have:*

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

Proof: We have argued above that

$$\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(k,\vec{x}^\rho,\vec{z}^\rho) \stackrel{\mathrm{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(k,\vec{x}^\rho|_0,\vec{z}^\rho|_0).$$

However, $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(k,\vec{x}^\rho,\vec{z}^\rho) \stackrel{\mathrm{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(k,\vec{x}^\rho,\vec{z}^\rho)$, and $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(k,\vec{x}^\rho|_0,\vec{z}^\rho|_0) \stackrel{\mathrm{d}}{=}$ $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(k,\vec{x}^\rho|_0,\vec{z}^\rho|_0)$. The claim follows. □

Construction of $\mathcal{A}_\pi$. Adversary $\mathcal{A}_\pi$ proceeds as follows.

1. $\mathcal{A}_\pi$ invokes $\mathcal{A}$ on its own input, auxiliary input and random input, and follows the instructions of $\mathcal{A}$ up to round $l-1$. (Recall that so far, both in π and in π^ρ the parties run π.) In addition, $\mathcal{A}_\pi$ keeps another piece of its 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 adversary $\mathcal{A}_\pi$ interacts with parties that call a trusted party for ideal evaluation of function f. In order to continue the run of $\mathcal{A}$, adversary $\mathcal{A}_\pi$ invokes simulator $\mathcal{S}_\rho$. The auxiliary input z_0^ρ for $\mathcal{S}_\rho$ is set to the current internal state of $\mathcal{A}$. The random input of $\mathcal{S}_\rho$ is set to r^ρ. Next, running $\mathcal{S}_\rho$ is done as follows.

   (a) When $\mathcal{S}_\rho$ asks to corrupt (in its ideal process) a party that is already corrupted, $\mathcal{S}_\rho$ is given input values 0 for this party. (Recall that these are the a-priori corrupted parties, thus their inputs do not affect the distribution of the output of $\mathcal{S}_\rho$. See Claim 11.)

   (b) When $\mathcal{S}_\rho$ asks to corrupt a party P_i that is not yet corrupted, $\mathcal{A}_\pi$ corrupts P_i in its f-hybrid model; let x_i^f be the value that P_i is about to hand T, the trusted party for f. Then $\mathcal{A}_\pi$ informs $\mathcal{S}_\rho$ that P_i's input is x_i^f. In addition, $\mathcal{A}_\pi$ sets z_i^ρ to contain P_i's internal history pertaining to protocol π, and hands z_i^ρ to $\mathcal{A}_\rho$ as P_i's auxiliary input.

   (c) When $\mathcal{S}_\rho$ hands its trusted party the inputs of the corrupted parties and asks for the values of f, then $\mathcal{A}_\pi$ invokes its own trusted party, T, with the same input values for the corrupted parties, and hands the answer back to $\mathcal{S}_\rho$.

   (d) If P_i is broken into after Step 2c then $\mathcal{S}_\rho$ is given also P_i's output for f.

3. Let v denote the (early) output of $\mathcal{S}_\rho$, before it starts interacting with a post-execution corruptor. Recall that v is an internal state of $\mathcal{A}$ at the round, l_π, where the execution of π resumes. $\mathcal{A}_\pi$ continues the current run of $\mathcal{A}$ from internal state v until the completion of protocol π, and follows $\mathcal{A}$'s instructions. When $\mathcal{A}$ corrupts a party P at this stage, $\mathcal{A}_\pi$ proceeds as follows.

   (a) $\mathcal{A}_\pi$ corrupts P in its f-hybrid model and obtains P's internal history pertaining to protocol π.

   (b) $\mathcal{A}_\pi$ plays the role of the post-execution corruptor for $\mathcal{S}_\rho$, and requests corruption of P. It then obtains the (simulated) internal history of P pertaining to protocol ρ. (In the process $\mathcal{S}_\rho$ may corrupt P in its ideal process. In this case $\mathcal{A}_\pi$ hands $\mathcal{S}_\rho$ the input and auxiliary input for P as described in Steps 2b and 2d.)

(c) $\mathcal{A}_\pi$ combines the data from the previous two steps to obtain the internal history of P pertaining to protocol π^ρ, and hands this data to $\mathcal{A}$.[26]

4. Once protocol π is completed, $\mathcal{A}_\pi$ outputs whatever $\mathcal{A}$ outputs, and is now ready to interact with a post-execution corruptor.

We are now guaranteed that there exists a simulator $\mathcal{S}_\pi$ such that $\text{IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}} \stackrel{\text{d}}{=} \text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}$. (Here $\mathcal{P}$ is the post-execution corruptor for π^ρ, fixed together with adversary $\mathcal{A}$.) We let $\mathcal{S} = \mathcal{S}_\pi$.

Analysis of $\mathcal{S}$. Fix an input vector $\vec{x}$, a post-execution corruptor $\mathcal{P}$, and auxiliary input vector $\vec{z}$, as well as some large enough value of the security parameter. We show that

$$\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(k,\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(k,\vec{x},\vec{z}). \tag{8}$$

The rightmost equality follows from the security of π in the f-hybrid model. The leftmost equality is shown in three steps, as follows. (Steps 2 and 3 are partitioned differently than the non-adaptive case.)

Step 1. Recall that l_ρ is the round where protocol π makes the ideal evaluation call, and protocol π^ρ invokes ρ. Then, until round $l_\rho - 1$ protocols π and π^ρ behave the same. That is, let $\vec{r}^\pi$ be some fixed random-input vector, shortened to include only the portion that is used for protocol π (both before and after the execution of ρ.) We have:

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

Step 2. Similarly to the non-adaptive case, we proceed to define the inputs and auxiliary inputs of the parties to protocol ρ (resp., to the ideal evaluation of f) in the above executions. Here, however, these definitions require a bit more care.

Let C be the set of a-priori corrupted parties, determined by $\vec{r}^\pi$. (That is, C is the set of parties that are corrupted by the onset of round l_ρ.) The set C is identical in the two executions. Next:

DEFINING PARAMETERS FOR THE EXECUTION OF ρ: Let $\vec{x}^\rho$ denote the input values that the parties have for protocol ρ, as described in $\text{GS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(l_\rho - 1, k, \vec{x}, \vec{z}, \vec{r}^\pi)$. Let z_i^ρ denote the internal history of party P_i at the end of round $l_\rho - 1$ of this execution. The inputs and internal histories of the parties in C are set to 0. Let z_0^ρ denote $\mathcal{A}$'s internal state at round $l_\rho - 1$. Let z_P^ρ, the auxiliary input of $\mathcal{P}_\rho$, be set to $z_P^\rho = z_0^\rho, z_1^\rho, ..., z_n^\rho$. Let $\vec{z}^\rho = z_P^\rho, z_0^\rho, ..., z_n^\rho$.

DEFINING PARAMETERS FOR THE IDEAL EVALUATION OF f: Let $\vec{x}^f$ denote the input values that the parties prepare for the trusted party for f, as described in $\text{GS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(l_\rho - 1, k, \vec{x}, \vec{z}, \vec{r}^\pi)$. Let z_i^f denote the internal history of party P_i at the end of round $l_\rho - 1$ in this interaction.

Let z_0^f denote $\mathcal{A}$'s internal state at the end of round $l_\rho - 1$. Set z_P^f, the auxiliary input of corruptor $\mathcal{P}_\rho$, to $z_P^f = z_0^f, z_1^f, ..., z_n^f$. Let $\vec{z}^f = z_P^f, z_0^f, ..., z_n^f$.

CONCLUSION: It follows from the above description that $\vec{x}^\rho = \vec{x}^f|_0$, and that $\vec{z}^\rho = \vec{z}^f|_0$. It thus follows from the validity of $\mathcal{S}_\rho$ and from Claim 11 that

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

[26] It can be be seen that the interaction between $\mathcal{A}_\pi$ and $\mathcal{S}_\rho$ during Step 3 is identical to the interaction between $\mathcal{P}_\rho$ and $\mathcal{S}_\rho$, given the appropriate auxiliary input for $\mathcal{P}_\rho$.

Step 3. 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 $\mathcal{A}_\rho$, $\mathcal{P}_\rho$, and $\mathcal{A}_\pi$ that:

1. Let $\vec{r^\rho}$ be a random input vector for protocol ρ, and let $\vec{r}$ denote the combination of $\vec{r^\pi}$ and $\vec{r^\rho}$. (That is, $\vec{r}$ is an entire random-input vector for π^ρ.) Let $\hat{\text{GS}}_{\pi^\rho,\mathcal{A},\mathcal{P}}(l,k,\vec{x},\vec{z},\vec{r})$ be the vector $\text{GS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(l,k,\vec{x},\vec{z},\vec{r})$ after removing, for each uncorrupted party, all the internal states pertaining to protocol ρ (except for the output from ρ). Then, $\hat{\text{GS}}_{\pi^\rho,\mathcal{A},\mathcal{P}}(l_\pi,k,\vec{x},\vec{z},\vec{r})$ can be obtained from $\text{GS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(l_\rho-1,k,\vec{x},\vec{z},\vec{r^\pi})$ and $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(k,\vec{x^\rho},\vec{z^\rho},\vec{r^\rho})$ via a (simple, deterministic) procedure. (This procedure is similar to the one in the non-adaptive case: it essentially updates the internal histories of the parties and the internal state of the adversary.[27])

   Furthermore, for each round $l > l_\pi$ the vector $\hat{\text{GS}}_{\pi^\rho,\mathcal{A},\mathcal{P}}(l,k,\vec{x},\vec{z},\vec{r})$ can be obtained from $\hat{\text{GS}}_{\pi^\rho,\mathcal{A},\mathcal{P}}(l-1,k,\vec{x},\vec{z},\vec{r^\pi})$ and $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(k,\vec{x^\rho},\vec{z^\rho},\vec{r^\rho})$ via the following procedure: Continue the execution for one round from the global state described in $\hat{\text{GS}}_{\pi^\rho,\mathcal{A},\mathcal{P}}(l-1,k,\vec{x},\vec{z},\vec{r^\pi})$. If no new corruption occurs in this round then the goal is reached. In case that $\mathcal{A}$ corrupts a new party, P_i, take the internal history of P_i pertaining to protocol ρ from the output of $\mathcal{P}_\rho$ in $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(k,\vec{x^\rho},\vec{z^\rho},\vec{r^\rho})$. (It is guaranteed that $\mathcal{P}_\rho$ has corrupted P_i in this execution, since $\mathcal{P}_\rho$ orchestrates an identical execution with the same inputs and random inputs.)

   Even further, $\mathcal{A}$'s interaction with any post-execution corruptor $\mathcal{P}$ at the completion of the execution of π^ρ is determined by $\hat{\text{GS}}_{\pi^\rho,\mathcal{A},\mathcal{P}}(l_\pi,k,\vec{x},\vec{z},\vec{r^\pi})$ and $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(k,\vec{x^\rho},\vec{z^\rho},\vec{r^\rho})$ via a similar procedure.

2. Let $\vec{r^f}$ be a random input vector for the ideal evaluation process of f, and let $\vec{r^{\pi,f}}$ denote the combination of $\vec{r^\pi}$ and $\vec{r^f}$. ($\vec{r^{\pi,f}}$ is an entire random-input vector for π in the f-hybrid model.) Then, $\text{GS}_{\pi,\mathcal{A}_\pi}(l_\rho,k,\vec{x},z,\vec{r^{\pi,f}})$ 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 procedure as in the previous bullet (the real-life execution).*

   Furthermore, for each round $l > l_\pi$ the procedure for determining $\hat{\text{GS}}_{\pi,\mathcal{A}_\pi,\mathcal{P}}(l,k,\vec{x},\vec{z},\vec{r^{\pi,f}})$ from $\hat{\text{GS}}_{\pi,\mathcal{A}_\pi,\mathcal{P}_\rho}(l-1,k,\vec{x},\vec{z},\vec{r^{\pi,f}})$ and $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(k,\vec{x^\rho},\vec{z^\rho},\vec{r^\rho})$ is identical to the one in the the real-life execution. The interaction with post-execution corruptor $\mathcal{P}$ at the completion of the execution of π is also determined in the same way as there.

Let $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z})|\vec{r^\pi}$ (resp., $\text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(k,\vec{x},\vec{z})|\vec{r^\pi}$) denote the distribution of $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z})$ (resp., of $\text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(k,\vec{x},\vec{z})$) when the random input for π is fixed to $\vec{r^\pi}$. It follows that for any value of $\vec{r^\pi}$ we have

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

Equation (8) follows.

6 The computational setting

In this section we define secure protocols and prove the composition theorem in the computational setting, where the adversary sees all the communication among the parties but 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.

[27] More precisely, this procedure first modifies the internal history of each uncorrupted party P_i by adding $\text{EXEC}_{\rho,\mathcal{A}_\rho}(k,\vec{x^\rho},z^\rho,\vec{r^\rho})_i$ 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 histories of the uncorrupted parties.

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. The definition is very similar to that of the secure channels setting (Section 5.1). Executing a protocol π in the real-life scenario, as well as the notation $\text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}$, are the same as in the secure channels setting, with the following exceptions:

1. The real-life adversary, $\mathcal{A}$, and the post-execution corruptor $\mathcal{P}$ are probabilistic polynomial time (PPT).[28]
2. $\mathcal{A}$ sees all the communication between the uncorrupted parties.[29] Consequently, when a party gets corrupted the new data learnt by the adversary is only the party's input and random input.

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 referred to in the ideal-vs-real emulation requirement, made below.) The notation $\text{IDEAL}_{f,\mathcal{S},\mathcal{P}}$ 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 12 (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 computes f in the computational setting, *if for any* PPT *t-limited real-life adversary $\mathcal{A}$ and any* PPT *post-execution corruptor $\mathcal{P}$ there exists a* PPT *ideal-process adversary $\mathcal{S}$, such that*

$$\text{IDEAL}_{f,\mathcal{S},\mathcal{P}} \stackrel{c}{\approx} \text{EXEC}_{\pi,\mathcal{A},\mathcal{P}} \tag{9}$$

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

6.2 Discussion

We present further discussion on Definition 12.

Remark 1: On the complexity of $\mathcal{P}$. We stress that Definition 12 quantifies only over all post-execution corruptors $\mathcal{P}$ that are PPT. This is so since in the computational setting we assume that all involved entities (including the "environment" represented by $\mathcal{P}$) are PPT. Furthermore, a definition that allows $\mathcal{P}$ more computational power will be hard to satisfy, since an over-powerful $\mathcal{P}$ 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 $\mathcal{P}$ access to the communication among the parties, via $\mathcal{A}$'s view.)

[28] The running time of the adversary (as well as that of all other entities involved) is measured against 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.

[29] We assume that the links are *ideally authenticated*, namely the adversary cannot *alter* the communication.

Remark 2: On computational correctness. It may appear at first glance that Definition 12 (security in the computational setting) has the weakness that it requires only "computational correctness" from protocols. An argument to this effect may proceed as follows: Assume that the parties wish to compute a function that is the value of some pseudorandom generator g on a randomly chosen input, and consider the protocol that outputs a totally random value. This protocol would be considered secure since the definition requires only that the output of the protocol be *computationally indistinguishable* from the output of the ideal process; yet we may want to compute g precisely.

This argument, however, is misleading. In particular, it lacks the distinction between the case where the evaluated function is deterministic and the case where it is probabilistic. Let us elaborate, using above example as a reference point. Consider first the scenario where g is evaluated as a deterministic function: All parties have a common input x and the function value is $g(x)$. (Alternatively, the input x can be shared among the parties using some standard secret sharing method.) In this case the definition requires the protocol to output the correct value, $g(x)$, on *every* input x. (Recall that the distinguisher between the two sides of (9) sees the inputs and auxiliary inputs of all parties.) Thus in this case the function g must be computed precisely, and the fact that it happens to be a pseudorandom generator is irrelevant. In particular, outputting a random value regardless of x is not considered secure.

Next, consider an alternative scenario: the parties have empty inputs, and wish to jointly evaluate g on a random input. That is, they wish to compute a *probabilistic* function f defined as $f() = g(x)$, where x is a random value. Here x is part of the 'intrinsic randomness' of f. That is, in the ideal process x is chosen by the trusted party. Now, a protocol that outputs a random value is considered secure, and is arguably so, since we know (by the pseudorandomness of g) that in the computational setting a random value is just as good as a the output of a pseudorandom generator *on a random and unknown input.*[30]

In other words, Definition 12 gives the designer of the task at hand the *freedom* to choose whether to require absolute correctness (by choosing to evaluate a deterministic function) or computational correctness (by choosing to evaluate a probabilistic function).

6.3 Modular composition: The computational case

We state and prove the composition theorem 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 $\text{EXEC}^{f_1,...,f_m}_{\pi,\mathcal{A},\mathcal{P}}$ remains unchanged (yet here it applies to the computational setting). Protocols for securely computing a function g in the computational $(f_1, ..., f_m)$-hybrid model are defined in the usual way:

[30] Alternatively, one may want a protocol that securely computes the above probabilistic function $f()$, with the additional requirement that the outputs of the uncorrupted parties will be distributed *identically* to $g(x)$ for a randomly chosen x. This more stringent security requirement, while somewhat less motivated, can be expressed as follows. A protocol π satisfies this requirement if there exists a protocol π' such that:
(a). π' securely computes the deterministic function $g(x)$ (on every input).
(b). Protocol π consists of having each party first randomly choose an input for π' from the appropriate domain, and then execute π' on the chosen input.
It is easy to see that such a protocol π has this additional security property.

Definition 13 *Let $f_1, ..., f_m, g$ be n-party functions and let π be a protocol for n parties in the computational $(f_1, ..., f_m)$-hybrid model. We say that π* adaptively t-securely computes g in the computational $(f_1, ..., f_m)$-hybrid model *if for any* PPT *adaptive t-limited adversary $\mathcal{A}$ (in the f-hybrid model) and every* PPT *post-execution corruptor $\mathcal{P}$, there exists a* PPT *adaptive ideal-process adversary $\mathcal{S}$ such that:*

$$\text{IDEAL}_{g,\mathcal{S},\mathcal{P}} \stackrel{c}{\approx} \text{EXEC}^{f_1,...,f_m}_{\pi,\mathcal{A},\mathcal{P}}. \tag{10}$$

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

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 case of secure channels.

Theorem 14 (adaptive modular composition in the computational setting) *Let $t < n$, and let $f_1, ..., f_m, g$ be n-party functions. Let π be an n-party protocol that adaptively t-securely (resp., t-privately) computes g in the computational $(f_1, ..., f_m)$-hybrid model, and assume that no more than one ideal evaluation call is made at each round. Let $\rho_1, ..., \rho_m$ be n-party protocols that adaptively t-securely (resp., t-privately) compute $f_1, ..., f_m$, respectively, in the computational setting. Then protocol $\pi^{\rho_1,...,\rho_m}$ adaptively t-securely (resp., t-privately) computes g in the computational setting.*

Proof: As in the case of adaptive security with secure channels, we restrict the presentation to active adversaries and to protocols where $m = 1$ and the trusted party is called only once. The cases of passive adversaries and $m > 1$ follow as there.

The constructions of $\mathcal{A}_\rho$, $\mathcal{P}_\rho$, $\mathcal{A}_\pi$ and $\mathcal{S}$ 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 $\mathcal{A}_\rho$, $\mathcal{P}_\rho$ are linear in the complexity of $\mathcal{A}$; the complexity of $\mathcal{A}_\pi$ is linear in the complexities of $\mathcal{A}$ and $\mathcal{S}_\rho$, and the complexity of $\mathcal{S}$ is identical to that of $\mathcal{S}_\pi$. Analyzing $\mathcal{S}$, we show:

$$\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}} \stackrel{c}{\approx} \text{EXEC}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{P}} \stackrel{c}{\approx} \text{IDEAL}_{f,\mathcal{S},\mathcal{P}}. \tag{11}$$

The rightmost equality follows from the security of π in the f-hybrid model. The leftmost equality is shown similarly to the secure channels case. Essentially, the only difference from the proof there is in Step 2, namely that $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}$ and $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}$ are only guaranteed to be computationally indistinguishable; but this suffices to show (11).

More precisely, given a distinguisher $\mathcal{D}$ between $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}}$ and $\text{EXEC}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{P}}$, construct a distinguisher $\mathcal{D}'$ between $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}$ and $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}$. On input k and an $(n+2)$ vector $\vec{v} = v_P, v_0, ...v_n$, distinguisher $\mathcal{D}'$ orchestrates an execution of π^ρ with adversary $\mathcal{A}$ and corruptor $\mathcal{P}$, on the inputs and random inputs appearing in v_P, and using the data in $\vec{v}$ for the parties' outputs from ρ. (Recall that $\vec{v}_P$ is the output of $\mathcal{P}_\rho$). Once the global output $\vec{v}'$ from this execution is generated, D' runs $\mathcal{D}$ on $(k, \vec{v}')$ and outputs whatever $\mathcal{D}$ outputs.

Using the same arguments as in the secure channels case, it is seen that if $\vec{v}$ has the distribution of $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(k, \vec{x}, \vec{z})$ for some $\vec{x}, \vec{z}$ then $\vec{v}'$ has the distribution of $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}}(k, \vec{x}, \vec{z})$. similarly, if $\vec{v}$ has the distribution of $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(k, \vec{x}, \vec{z})$ then $\vec{v}'$ has the distribution of $\text{EXEC}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{P}}(k, \vec{x}, \vec{z})$. □

Non-adaptive security in the computational setting. We remark that in the above proof the distinguisher $\mathcal{D}'$ does not use its auxiliary information, w, provided in Definition 3 (unless distinguisher $\mathcal{D}$ does). Thus, Theorem 14 holds even under a weaker version of Definition 3, where the distinguisher has no auxiliary information.

The distinguisher's auxiliary information becomes necessary in the case of non-adaptive adversaries in the computational setting. There, the distinguisher $\mathcal{D}'$ does not have access to the various inputs of the parties for π^ρ. Instead, it will receive the inputs, auxiliary inputs, and random inputs of the parties for π^ρ in its own auxiliary input, w.

On the propagation of the computational distance. It can be seen that the composition does not 'waste' the security of the protocols. That is, if there exists an adversary $\mathcal{A}$, a post-execution corruptor $\mathcal{P}$, and a distinguisher $\mathcal{D}$ that distinguishes between $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}}(k,\vec{x},\vec{z})$ and $\text{IDEAL}_{g,\mathcal{S},\mathcal{P}}(k,\vec{x},\vec{z})$ with probability δ (for some values of k, $\vec{x}$ and $\vec{z}$), then there exist values δ_1 and δ_2 such that $\delta_1+\delta_2=\delta$, and distinguishers $\mathcal{D}_1$ and $\mathcal{D}_2$, such that:

- $\mathcal{D}_1$ distinguishes between $\text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(k,\vec{x},\vec{z})$ and $\text{IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(k,\vec{x},\vec{z})$ with probability δ_1.
- $\mathcal{D}_2$ distinguishes between $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(k,\vec{x}^\rho,\vec{z}^\rho)$ and $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(k,\vec{x}^f,\vec{z}^f)$ with probability δ_2, where $\vec{x}^\rho,\vec{z}^\rho,\vec{x}^f,\vec{z}^f$ are the values defined in Step 2 of the proof of Theorem 10.

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.

I 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.

References

[1] D. Beaver, "Foundations of Secure Interactive Computing", *CRYPTO,* 1991.

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

[3] D. Beaver and S. Goldwasser, "Multi-party computation with faulty majority", *30th FOCS,* 1989, pp. 468-473.

[4] M. Bellare, R. Canetti and H. Krawczyk, "A modular approach to the design and analysis of authentication and key-exchange protocols", *30th STOC,* 1998.

[5] M. Ben-Or, R. Canetti and O. Goldreich, "Asynchronous Secure Computations", *25th STOC,* 1993, pp. 52-61.

[6] M. Ben-Or, S. Goldwasser and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation", *20th STOC,* 1988, pp. 1-10.

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

[8] R. Canetti, "Modular composition of multi-party cryptographic protocols: The concurrent case", in preparation.

[9] R. Canetti, U. Feige, O. Goldreich and M. Naor, "Adaptively Secure Computation", *28th STOC,* 1996. Fuller version in MIT-LCS-TR #682, 1996.

[10] R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In *Proceedings of CRYPTO'94*.

[11] R. Canetti and R. Gennaro, Incoercible multi-party computation, *37th FOCS*, 1996.

[12] R. Canetti, S. Halevi and A. Herzberg, "How to Maintain Authenticated Communication", Available at the Theory of Cryptography Library, `http://theory.lcs.mit.edu/~tcryptol/`, 1998. Preliminary version at *16th PODC,* 15-25, 1997.

[13] R. Canetti, E. Kushilevitz, R. Ostrovsky and A. Rosen, "Randomness vs. Fault-Tolerance", Available at the Theory of Cryptography Library, `http://theory.lcs.mit.edu/~tcryptol/`, 1998. Preliminary version at *16th PODC,* 35-45, 1997.

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

[15] B. Chor and E. Kushilevitz, "a Zero-One law for boolean privacy", SIAM J. Disc. Math. 4 (1991) 36-47. Preliminary version in STOC 21 (1989) 62-72.

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

[17] Y. Desmedt and Y. Frankel, "Threshold cryptosystems", In G. Brassard, editor, *Advances in Cryptology — Crypto '89*, pages 307–315, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science No. 435.

[18] D. Dolev, C. Dwork and M. Naor, Non-malleable cryptography, *23rd STOC,* 1991.

[19] P. Feldman, "A practical scheme for non-interactive Verifiable Secret Sharing", *28th FOCS,* 1987, pp. 427-437.

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

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

[22] O. Goldreich, *"Foundations of Cryptography (Fragments of a book)"*, Weizmann Inst. of Science, 1995. (Avaliable at `http://theory.lcs.mit.edu/~tcryptol/`)

[23] O. Goldreich. *"Secure Multi-Party Computation"*, 1998. First draft available at `http://theory.lcs.mit.edu/~oded`

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

[25] O. Goldreich, S. Micali and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design", *27th FOCS,* 1986, pp. 174-187.

[26] O. Goldreich, S. Micali and A. Wigderson, "How to Play any Mental Game", *19th STOC,* 1987, pp. 218-229.

[27] O. Goldreich and Y. Oren, "On the cunning power of cheating verifiers: Some observations about Zero-Knowledge proofs", in preparation. Preliminary version by Y. Oren in *28th FOCS,* 1987.

[28] S. Goldwasser, and L. Levin, "Fair Computation of General Functions in Presence of Immoral Majority", *CRYPTO,* 1990.

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

[30] 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.

[31] M. Hirt and U. Maurer, "Complete characterization of adversaries tolerable in secure multi-party computation", *16th PODC,* 1997, pp. 25–34.

[32] E. Kushilevitz, "Privacy and communication complexity", SIAM Jour. disc. Math. Vol. 5. No. 2 (1992), 273-284. Preliminary version in 29th FOCS (1989).

[33] J. Kilian, E. Kushilevitz, S. Micali, R. Ostrovsky, "Reducibility and Completeness in Private Computations", manuscript, 1997. Preliminary versions in 23rd STOC (1991) by Kilian and in 35th FOCS (1994) by Kushilevits. Micali and Ostrovsky.

[34] S. Micali and P. Rogaway, "Secure Computation", unpublished manuscript, 1992. Preliminary version in *CRYPTO 91.*

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

[36] 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, Montreal, Quebec, Canada*, pages 51–59, 1991.

[37] T. Rabin, "A Simplified Approach to Threshold and Proactive RSA" *CRYPTO '98,* 1998.

[38] T. Rabin and M. Ben-Or, "Verifiable Secret Sharing and Multi-party Protocols with Honest Majority", *21st STOC,* 1989, pp. 73-85.

[39] A. Yao, 'theory and applications of trapdoor functions', In *Proc. 23rd Annual Symp. on Foundations of Computer Science*, pages 80–91. IEEE, 1982.

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

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

A Extensions of the definitions

We have seen how the general framework of section 2 is used to capture security in several settings (the secure channels setting, the computational setting, in the presence of non-adaptive, adaptive, passive, active adversaries). Below we sketch how this framework can provide a sound basis for extensions needed to deal with several other, related scenarios.

Asynchronous networks. Here we modify both the real-life and the ideal-process. The real-life computation is modified to reflect total asynchrony: first, we consider message-driven protocols (i.e., upon the receipt of a message the party performs local computation, sends some messages, and then waits for the next received message). Plus, the adversary has the additional power of delivering messages with arbitrary order and delays. The ideal process is modified to reflect 'the most we can hope for' in an asynchronous network; in particular it reflects the fact that up to t parties may never join the computation. That is, the trusted party waits to receive the inputs of only $n-t$ parties, where t is the maximum number of corruptions. It then computes the outputs based on the received $n-t$ values. The ideal-process adversary has the additional power of deciding the identities of these $n-t$ parties. Other than these modifications, the definition remains unchanged. See details in [7, 5].

Coercion. This extension deals with resilience to coercion, or to adversaries that try to learn the parties' inputs by requiring attacked parties to provide 'convincing' internal data (rather then by a full-fledged corruption where the true internal data of the corrupted party is revealed 'by physical means'). Also here both the real-life and the ideal-process computations are modified to reflect the new concern, while keeping the overall structure unchanged. See details in [11].

Authentication and key exchange. This extension consists of two parts. The first part deals with formalizing the notion of authentication protocols (or authenticators). For this purpose, the real-life model is *unauthenticated*. That is, the adversary can delete, inject and modify messages at wish. We also don't impose any timing constraints on the adversary, thus allowing asynchronous message delivery. Instead of comparing the real-life computation to an ideal process, here we compare it to another real-life computation — one in which the links are ideally authenticated. That is, we define an ideally authenticated model, where it is guaranteed that any message received was indeed sent and wasn't modified or duplicated. (Yet, messages can be deleted even in the ideally-authenticated model.) An authenticator is a "protocol complier" is a good that, given a protocol π in the ideally-authenticated model, generates a protocol π' that runs in the real-life (unauthenticated) model, yet emulates protocol π (in the usual sense).

The second part of this extension defines secure key-exchange protocols, and demonstrates that using key-exchange protocols in the standard way results in good authenticators. Key exchange protocols are defined as those that emulate a specially tailored "ideal key exchange process". Interestingly, this approach also provides a modular methodology for the design of key exchange protocols. See details in [4].

Distributed signatures. This extension captures a functionality that is somewhat different than secure function evaluation, namely that of secure distributed signatures. Here a set of signers share the capability to generate signatures on messages. The main security concern is making sure that an external (possibly adversarial) entity that interacts with the system will be able to obtain a valid signature on a message if and only if a "legal" subset of the signers have agreed to sign this message.

To capture these requirements a special-purpose ideal process is defined. In the ideal process there are no signing and verification keys; instead, a trusted party maintains a publically available "database" of "signed messages". A message is added to the database if a legal subset of the signing parties wishes to do so. Verifying whether a message is "signed" is done by querying the database. A distributed signature scheme is secure if it emulates this ideal process in the standard sense. See [12] for more details.

On-line security. In order to prove that modular composition preserves security of protocols even when several subroutine calls may run at the same time, a stronger notion of security is required. We call this stronger notion on-line security. (The intuition behind the name is that the ideal-process adversary has to generate its output in an "on-line" fashion, reacting to new auxiliary inputs that arrive periodically.) On-line security is studied in [8].

To obtain on-line security the current real-life model and ideal process are modified essentially as follows. We replace the post-execution corruptor with an arbitrary environment machine, $\mathcal{Z}$. This machine starts off with some auxiliary information (typically, this information will include the inputs of *all* parties to some global computation), and interacts with the adversary in an arbitrary way. The view of $\mathcal{Z}$ is added to the global output of the computation in the two models; the definition of security now requires that the global outputs be similarly distributed for *any* $\mathcal{Z}$.

Note the introduction of machine $\mathcal{Z}$ simplifies the presentation somewhat, since now the parties and the adversary no longer need auxiliary inputs, and in addition the post-execution corruptor $\mathcal{P}$ is no longer necessary. However, the resulting security requirement is stronger.

Reactive systems. This work deals with the special case of evaluating a function on inputs that are fixed and known before the computation starts. A more general case is that of on-going, reactive computations where the inputs are being learned as the computation proceeds (and may depend on previous computational steps). This extra generality may be captured in a way that is reminiscent of the definition of on-line security (see the previous paragraph). The real-life model is modified so that the parties receive a new set of inputs at arbitrary times during the computation (say, at at the onset of each communictaion round). In addition, the protocol may specify an output at arbitrary times. The inputs are generated by an input adversary $\mathcal{I}$ that starts off with some arbitrary input and in addition sees the global output of the computation until this round. (Essentially, the difference between $\mathcal{I}$ and the auxiliary information machine $\mathcal{Z}$ from the previous paragraph is that $\mathcal{I}$ interacts also with the uncorrupted parties.) The global output of the computation is defined in rounds: at each round it contains the uncorrupted parties' outputs together with the adversary output up to this point.

The ideal process is modified similarly to the real-life model. That is, the adversary interacts with $\mathcal{I}$; in addition $\mathcal{I}$ periodically hands new inputs to the parties. The trusted party receives the inputs of the parties in rounds; in each round it hands the current outputs to the parties, according to the pre-defined task. The global output of each round are defined in the natural way.

The emulation requirement (say, Definition 8) is extended as follows. The "for all inputs $\vec{x}$" requirement is replaced with "for all input adversaries $\mathcal{I}$". Furthermore, it is now required that for all rounds, the global outputs of the ideal process be distributed identically to the global output of the real-life computation.

B Other definitions

We briefly review some approaches to defining secure multi-party computation. These are the approaches of Micali and Rogaway [34], Goldwasser and Levin [28], Beaver [1, 2], and Canetti, Feige, Goldreich and Naor [7, 9]. 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. It seems that the most rigorously presented of these definitions is the one in [34].

Micali and Rogaway [34] also envision an ideal process, similar to the one here, for secure function evaluation. However, the ideal process remains as a motivating intuition behind their definition, and is not explicitly used as here. Instead they specify a list of requirements from a protocol and postulate that these requirements amount to emulation of the ideal-process computation by a real-life computation. A sketch of these requirements follows. 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. In any execution of the protocol the outputs of the uncorrupted parties, as determined by the output awareness function, should equal the value of the computed function applied to the contributed inputs (as determined by the input awareness function).[31] Next it is required 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 a certain type of simulation (eg, 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: only when a party is corrupted by the adversary does the simulator receive the corresponding information.

We remark that the [34] definition applies only to the 'secure channels' setting where the adversary is computationally unbounded and does not learn the communication among uncorrupted parties. Also, the current version deals only with protocols that evaluate a *deterministic* function.

Goldwasser and Levin take a different approach. First they extract 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 computed (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.[32]

[31] Existence of the input awareness and output awareness functions is a strong requirement. In particular, it implies that protocols where parts of the computation are done locally without interaction (eg, a trivial protocol where no communication takes place and each party computes its output locally) are considered insecure. It is advocated by [34] that this restiction is an essential part of the notion of secure function evaluation.

[32] 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 difference between the [28] 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 [28]), one claims that executing the protocol in the idealized model is equivalent to an ideal evaluation process of a specific function.

Beaver's definition [1, 2] take a very similar approach to the [9, 7] definitions and the ones here. We sketch this approach in the [2] terminology. First a general notion of comparing security of protocols is formulated, as follows. Consider two protocols α and β for computing 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. ([2] is somewhat ambiguous regarding the specifics 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 process similar to the one here.

Disallowing rewinding is a considerable limitation, especially in the computational setting. (See Remark 3 in Section 4.2). An additional weakness of this defintion is that, unlike here, $\mathcal{A}'$ is not required to be as efficient at $\mathcal{A}$. (See Remark 1 in Section 4.2).

In addition, to allow secure sequential composition, the definition requires the protocol to be also post-protocol corruptible in a way that is reminiscent of our notion of post-execution corruptibility. Yet, the [2] notion does not define a post-execution corruptor; instead the security requirement is made with respect to "any sequence of post-execution corruption requests". This formalization 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. (See Remark 2 in Section 5.2.)

The [7, 9] approach is essentially the same as the one here, except for the following differences. First, their definition requires 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 a post-execution corruptor. Finally, their definition has 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 an uncorrupted party 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.