Security and Composition of Multi-party Cryptographic Protocols

Ran Canetti*

June 5, 1998

Abstract

We present general definitions of security for multi-party cryptographic protocols and show that, using these definitions, security is preserved under a natural composition method.

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 method is essentially the natural 'subroutine substitution' method suggested by Micali and Rogaway.

We first present the general definitional approach. Next we consider several settings for multi-party protocols. These include the cases of *non-adaptive* and *adaptive* adversaries, as well as the *information-theoretic* and the *computational* models.

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 maintain their *security* in the presence of malicious attacks is one of the central tasks in cryptography. Intuitively, security is understood as guaranteeing a set of correctness properties of the outputs, together with a set of secrecy requirements regarding the local data. Yet, capturing the security requirements of a cryptographic 'protocol problem' in a way that is both precise and workable is not a trivial task.

Indeed, although several general definitions of secure protocols have been proposed in the past (e.g. [18, 22, 28, 2, 7]), numerous constructions of multi-party cryptographic protocols, and in particular general constructions for 'solving any cryptographic protocol problem' (e.g., [31, 21, 6, 14, 30, 22]) have never been rigorously proven secure.

This paper aims at improving our ability to prove security of cryptographic protocols. We build on a known definitional approach [2, 7] that seems workable and flexible. (In particular, this approach has been used in a number of quite varied settings, e.g. [2, 5, 7, 9, 11, 4, 12, 13].) As a first step, we somewhat modify and simplify these definitions, and present them in a clear and concise way.

Next, we consider composition of protocols. A useful property of a workable definition of secure protocols is that the security be preserved under natural methods of *composing* simpler protocols into more complex ones. This is desirable 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 proving security of the above constructions, and others.

We show that our notion of security is preserved under a natural and modular composition method, in several important settings. We believe that the results and techniques presented here will be instrumental in proving security of protocols. For instance, our composition theorem already plays a central role in the security proof of [13]. Other candidates include [21, 6, 9, 4] and more.

Defining secure protocols. In a nutshell, the definitional approach of [2, 7] is sketched as follows: First formulate an 'ideal process' that captures the desired characteristics of the task at hand and, in particular, rules out any unwanted behavior. The ideal process involves adding an imaginary 'trusted party' to the network; this party receives data from the other parties, locally computes the desired outcome and hands it back to them. Next, say that a protocol securely performs the task if it 'emulates' the ideal process, in the following way: For *any* 'real-life adversary' attacking the protocol, there should *exist* a (highly constrained) 'ideal process adversary' that manages to achieve essentially the same effect on the ideal process. Since the ideal process guarantees that no undesirable behavior occurs, we are assured that no such behavior occurs when running the protocol in 'real-life'. (Capturing the 'effect of an adversary on a protocol' is a key part in the definition; we elaborate within.)

Although the approach is very attractive, both the [2] and the [7] formalizations suffer from shortcomings and unnecessary complexities that are 'cleaned up' here. One such no-longer-necessary complexity is the requirement that the 'ideal process adversary' operates via *black-box simulation* of 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.)

Another modification, relevant to adaptive adversaries that corrupt parties during the computation, is the treatment of corruption requests that occur *after* the protocol is completed. (Such requests may originate with adversaries interacting with other protocols run by the parties; accommodating these requests is essential for secure composition.) We introduce the notion of an *algorithmic* 'post-protocol corruptor' that interacts with the adversary after the protocol terminates. This allows us to prove, for the first time, a composition theorem for the case of computationally bounded adaptive adversaries.

Other modifications are described within. In addition, we elaborate on the [2, 7] definitions, as well as on other ones ([28, 22]) in Appendix B.

Several standard 'adversary models' are considered. The adversary can be passive (if it only 'gathers information' and does not modify the behavior of corrupted parties) or active (if the corrupted parties execute some arbitrary, malicious code). It can be non-adaptive (if the identities of corrupted parties are arbitrary but fixed before the computation starts) or adaptive (if the parties to be corrupted are chosen during the computation). We also consider the secure channels setting where channels are absolutely secure and the adversary has unlimited computational powers, and the computational setting where the adversary learns all the communication among the parties, and is restricted to probabilistic polynomial time. In Appendix A we sketch how the definition can be extended to deal with several other settings and concerns.

The paper concentrates on the case of honest majority, where strictly less than half of the parties are corrupted at any time. The presentation is also limited to the task of evaluating a (randomized or deterministic) function of the parties' inputs. We elaborate within.

Modular composition. When designing a protocol for some task, we want to be able to break down the task to several partial (presumably simpler) tasks, design protocols for these partial tasks, and then use the already designed protocols as subroutines in the solution for the given task. Furthermore, when proving the security of the combined protocol we do not want to re-prove the security of the subroutine protocols from scratch. In other words, we want to support the following methodology for designing protocols:
(1) Design a 'high-level' protocol for the given task assuming that other, simpler tasks can be carried out securely.
(2) Design protocols that securely carry out these simpler 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 method of combining protocols modular composition. (The first to consider modular composition of protocols in this context were Micali and Rogaway [28]. We elaborate below.) 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 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 using subroutine calls for ideal evaluation of $f_1 ... f_m$. Then the protocol $\pi^{\rho_1 \cdots \rho_m}$ that is identical to protocol π with the exception that every subroutine call for ideal evaluation of f_i is replaced by an invocation of protocol ρ_i, securely computes g from scratch.*

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, is dealt with in [8], and requires a stronger notion of security of protocols. (A sketch of this stronger notion appears in Appendix A.)

Several other composition methods for protocols are considered in the literature. For instance, 'sequential composition' usually means simply running two (secure) protocols one after the other, and 'parallel composition' means running them in parallel at the same time. These composition methods can be regarded as special cases of modular composition with the appropriate "high level" protocol. Yet, sometimes (i.e., when the high-level protocol is insecure, or in settings where modular composition does not hold) these methods do not preserve security [28, 2]. Consequently we regard modular composition as the main general tool for modular protocol design.

Some related work. Micali and Rogaway present another definition of secure function evaluation, for the case of deterministic functions in the secure channels setting [28]. In this setting, they define modular composition of protocols (there is is called reducibility of protocols), and sketch a proof of a composition theorem similar to the one here. The [28] definition involves some additional requirements from secure protocols. It was previously believed that some of these requirements were *necessary* for proving that modular composition preserves security. Here we show that modular composition preserves the security of protocols even under our more permissive definition.

On top of a general definition of security of protocols, [2] states a 'sequential composition' theorem. That is, if secure protocols are invoked sequentially, 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. In addition, the it seems to make some implicit limiting assumptions on the behavior of the adversary.

Another definitional approach is taken in [22]. We briefly discuss the [28, 2, 22, 7] definitions in Appendix B.

An information-theoretic notion of *privacy* for the case of non-adaptive, passive adversaries in the secure channels setting is studied in [15, 26]. Reducibility of protocols w.r.t. the [15, 26] notion of privacy is discussed in [27]. Yet the [27] notion of reducibility is different than the one here in that there the parties cannot communicate except for jointly invoking 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 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 in particular a definition of the 'semi-ideal' model of computation, with access to ideal evaluation of a given function), and a full proof.

Section 4 generalizes the treatment of Section 3 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 price of some repetition.

Section 5 deals with adaptive adversaries in the *computational* setting. Since the treatment is very similar to that of Section 4, this section is *not* self-contained, and should be read in conjunction with Section 4. The case of non-adaptive adversaries in the computational setting can be inferred quite easily. (It is also addressed in [20].)

Throughout Section 3, 4, 5, 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

This informal section presents the general approach taken by the [2, 7] definitions, as well as by ours.

First, we remark that the definitions apply only to case of honest majority. That is, they are realizable only if the adversary controls less than half of the parties. If the adversary controls half of the parties or more then our requirements have to be weakened somewhat. (Yet, the general approach will remain largely unchanged.) See [3, 22] for discussions and protocols for this dishonest majority case. See [20] for a treatment of the important special case where only two parties are involved, one of which is corrupted.

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 the inputs. This is a very general formalization that captures a large class of cryptographic tasks. Yet, some on-going tasks require a more general formalization — one that allows inputs to arrive at different times during the computation, and where new inputs depend on previous behavior of the network. In Appendix A we sketch how the definition can be extended to capture this setting.

Recall that the paper deals with eight basic adversary models: The adversary may be either passive or active, as well as either non-adaptive of adaptive. In addition, the setting may be either that of secure channels or a computational one.[1] Yet, the definitional approach is the same for all models, with 'local modifications' made to accommodate specific scenarios. In the rest of this section we motivate and sketch this general approach.

Some basic requirements. Two basic requirements come to mind when trying to capture the notion of secure function evaluation. The first is *correctness,* or in other words that the "good" parties (i.e., the parties that are not corrupted by the adversary), output "a correct" value of the function evaluated at the parties' inputs. This requirement is somewhat complicated by the fact that the function may be probabilistic (thus the output should be taken from some distribution), and more importantly by the fact that if the adversary is active then the corrupted parties cannot, in general, be prevented from changing their inputs to the function in an arbitrary way.

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 learn. This requirement seems to call for a definition based on some notion of simulation (as in the case of probabilistic encryption or zero-knowledge [23, 24]), 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, it can already be seen from the above description that these two requirements are "intertwined" to some extent: 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 corrupted parties.

Further inspection of the problem reveals some additional, "more subtle" requirements. For

[1] Other variations of these adversary models 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* it learns only messages sent to corrupted parties is often convenient for designing protocols. The definitions can be easily 'ported' to these adversary models.

instance, in the case of probabilistic functions we also want to make sure that the adversary be unable to affect the distribution of the outputs. In the case of *adaptive* adversaries, we don't want the identities of the corrupted parties to depend on "illegal information" (such as the local inputs of the uncorrupted parties).

One may try to list and formalize all these requirements (and others), and define a protocol as secure if it satisfies all. However, this approach is error-prone, since we may never be certain that we have listed all requirements; in addition, simply making each requirement separately may not be strong enough. (Remark 3, Section 3.1.1 contains an example of such a case.) Consequently, our definition takes a different approach. 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, all the above requirements). Then we say that a computation is secure if it "emulates" the ideal process, in some well-defined manner. In a way, this approach is a direct generalization of the 'simulation approach' of [23, 24] to the context of multi-party protocols. Let us elaborate.

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, and in the presence of a real-life adversary that controls a set of corrupted parties.[2] At the end of the computation the uncorrupted parties output whatever is specified in their protocol. The corrupted parties output nothing. In addition we envision the adversary as having its own output of the computation, representing the gathered information.

Next an ideal process for multi-party function evaluation is formulated, as follows. (The ideal process is meant to capture our requirements from this evaluation.) 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 the adversary is active then it 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 computes the expected outputs (i.e., it evaluates the given function at the given inputs) and hands each party its designated output. The evaluated function can be probabilistic; i.e., it may have an additional argument, taken to be the random input. In this case the trusted party tosses the necessary coins and uses the outcome to evaluate the function. Finally, the uncorrupted parties output whatever they receive from the trusted party. The corrupted parties output nothing. The adversary outputs some arbitrary string, representing the information gathered during the computation. (In the ideal process this information consists of the identities of the corrupted parties, their inputs, random inputs, and the values they received from the trusted party.)

Finally, 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 adversary can be also achieved in the ideal process by an ideal-process adversary. This requirement is captured as follows. We first define the *global output* of a computation on a given input. This is a random variable that contains the outputs of all parties and of the adversary. Next we require that for *any* real-life adversary $\mathcal{A}$ attacking a secure protocol π there *exists* an ideal-process adversary $\mathcal{S}$ 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 $\mathcal{S}$. (Here the notion of similarity can be either "equal distribution" or "statistical closeness" or "computational indistinguishability"[3].)

[2] The specifics of the interaction depend on the specific adversary model, say out of the eight models mentioned above.

[3] It may appear at first glance that allowing the global output of the ideal process to be only computationally

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, as well as the other requirements mentioned above, 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.

A bit more abstractly, this approach may be viewed as follows. A real-life multi-party computation is regarded as a stochastic process that, given a protocol, a real-life adversary and inputs to the parties, generates some distribution on the parties' outputs. The ideal process is another stochastic process defined analogously. A protocol is secure if for any real-life adversary there exists an ideal-process adversary such that, for any input, the resulting two processes generate similar output distributions.

3 Non-adaptive adversaries

In this section we define secure protocols, present and prove the composition theorem, for the case of non-adaptive adversaries. Adaptive adversaries are dealt with in Section 4.

Although non-adaptive security (i.e., security against non-adaptive adversaries) is considerably weaker than than adaptive security (see Remark 3 in Section 4.1.1), we choose to first present the non-adaptive case in full. This is for two main 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 reader. Second, some protocols in the literature (e.g., [21, 17, 25]) have security proofs that work only against non-adaptive adversaries. 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 5 we show how the treatment is adapted to settings where the adversary is restricted to probabilistic polynomial time.

Section 3.1 contains the definition of secure protocols. Section 3.2 presents the composition theorem, to be proven in Section 3.3.

3.1 Non-adaptive security of protocols

We 'flesh out' the approach outlined in Section 2 for the non-adaptive case. The definitions for the cases of active and passive adversaries are developed side by side, noting the differences throughout the presentation. This form of presentation has the advantage of having a uniform treatment of the two cases.[4]

We first describe the real-life model; next we describe the ideal process; next the notion of emulation (and the definition) is presented. Finally, Section 3.1.1 holds several remarks on the definition.

indistinguishable from the global output of the real-life computation results in an overly weak correctness requirement. We elaborate on this point in Remark 2, Section 4.1.1.

[4]Security for the special case of *passive, non-adaptive* adversaries can be captured using somewhat simpler definitions than the ones presented here. (Basically, this is so since in this case the correctness and the secrecy requirements can be stated separately wthout losing in security.) Yet, we choose not to use this simpler formalization since it does not generalize, neither to active adversaries nor to adaptive ones. This simpler formalization can be found in [20].

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 and random input r_i. 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, and some auxiliary input.[5]

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.[6] 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 outputs the messages 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.[7]

At the end of the computation (say, at some predetermined round) all parties locally generate their outputs. The uncorrupted parties output whatever is specified in the protocol. The corrupted parties output a special symbol specifying that they are corrupted. (Note that, technically speaking, 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 and random inputs, followed by the corrupted parties' inputs, random inputs, and all the messages sent and received by the corrupted parties during the computation. (Without loss of generality, we can imagine that the adversary's output consists of its entire view.)

We use the following notation. Let $\text{ADV}_{\pi,\mathcal{A}}(\vec{x}, z, \vec{r})$ denote the output of real-life adversary $\mathcal{A}$ with auxiliary input z and when interacting with parties running protocol π on input $\vec{x} = x_1 \dots x_n$ and random input $\vec{r} = r_1 \dots r_n$ as described above (x_i and r_i for party P_i). Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z, \vec{r})_i$ denote the output of party P_i after running protocol π on input $\vec{x}$, random input $\vec{r}$, and with a real life adversary $\mathcal{A}$ having auxiliary input z. If P_i is uncorrupted then this is the output specified by the protocol; if P_i is corrupted then $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z, \vec{r})_i = \perp$. Let

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

Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z)$ denote the random variable describing $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z, \vec{r})$ where $\vec{r}$ is uniformly chosen.

The ideal process. The model is parameterized by a probabilistic function $f : D^n \times \mathcal{R}_f \to E^n$, where $\mathcal{R}_f$ is taken to be the random input. (We call such functions n-party functions.) Each

[5]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.

[6]The description here captures a *fully connected, synchronous* network *with rushing.* It should be kept in mind that different models are possible, representing different real-life communication settings and network topologies.

In particular, when one is not concerned with efficiency it is possible to let the parties talk in a 'round robin', where in each communication round only a single party sends messages. This model somewhat simplifies presenting feasibility results.

[7]In the case of passive adversaries the ordering of events within a round is of no importance.

party P_i has input $x_i \in D$; no random input is needed. (Intuitively, the parties wish to compute $f(\vec{x}, r_f)_1, \ldots, f(\vec{x}, r_f)_n$, where $r_f \stackrel{R}{\leftarrow} \mathcal{R}_f$ and P_i learns $f(\vec{x}, r_f)_i$.[8]) 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.[9] Also here we usually do not distinguish between $\hat{\mathcal{S}}$ and $\mathcal{S}$. Adversary $\mathcal{S}$ starts off with the inputs of the corrupted parties, random input and some auxiliary input.[10] In addition, there is an (incorruptible) trusted party, T. The ideal process proceeds as follows.

Input substitution: The ideal-process-adversary $\mathcal{S}$ 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 $r_f \stackrel{R}{\leftarrow} \mathcal{R}_f$, and hands each P_i the value $f(\vec{y}, r_f)_i$.[11]

Output: Each uncorrupted party P_i outputs $f(\vec{y}, r_f)_i$, 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 $\{f(\vec{y}, r_f)_i : P_i \text{ is corrupted}\}$.

Let $\text{ADV}_{f,\mathcal{S}}(\vec{x}, z, \vec{r})$, where $\vec{r} = \langle f_f, r \rangle$, denote the output of ideal process adversary $\mathcal{S}$ on random input r 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

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

denote the outputs of the parties on inputs $\vec{x}$, adversary $\mathcal{S}$, and random inputs $\vec{r}$ as described above (party P_i outputs $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, z, \vec{r})_i$). Let $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, z)$ denote the distribution of $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, z, \vec{r})$ when $\vec{r}$ is uniformly distributed.

Comparing computations in the two models. Finally we require that protocol π emulates the process of evaluating f in the ideal process, in the following sense. For any real-life adversary $\mathcal{A}$ there should exist an ideal-process adversary $\mathcal{S}$, such that for any input vector $\vec{x}$ and any auxiliary input z, the output vectors $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, z)$ and $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z)$ are identically distributed.[12]

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

[8] $e \stackrel{R}{\leftarrow} D$ means that element e is drawn uniformly at random from domain D.

[9] As seen in the sequel, there is no need to explicitly limit the size of C in the ideal model. In particular, 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.

[10] It is essential that $\mathcal{S}$ be randomized. This holds even in a non-uniform complexity setting where no generality is lost by assuming that the real-life adversary (and the corrupted parties) are deterministic.

[11] 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).

[12] In [28, 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 1 in Section 3.1.1

glance. However, a second inspection will verify that this requirement is inevitable. See Remark 2 in Section 3.1.1.

For measuring complexity, we assume that the protocol π, the simulator $\mathcal{S}$ and the adversary $\mathcal{A}$ have 1^k as part of their input, where k is a security parameter that tends to infinity. (The n-party function f is now potentially an ensemble of functions, where a function corresponds to each value of k. Similarly, the auxiliary input z may depend on k.) Complexity is measured in terms of k. For simplicity, we leave k implicit in the notation.

Definition 1 (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) 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 $\mathcal{A}$, such that for every input vector $\vec{x}$ and every auxiliary input vector $\vec{z}$,*

$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x}, z) \stackrel{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z) \tag{1}$$

where $\stackrel{\text{d}}{=}$ stands for "identically distributed".[13]

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

We stress that in (1) the two *entire* $(n+1)$-vectors describing the global output of the computation must be identically distributed. See Remark 3 in Section 3.1.1 for more discussion.

3.1.1 Discussion

Remark 1: On the notion of emulation. In [28, 2, 7] the notion of emulation is more restrictive. Specifically, it was 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'; also, the simulator corrupts parties only when the adversary corrupts them. 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 inherently requires rewinding the adversary.)

It was speculated in [7, 9] (and, implicitly, also in [28, 2] that this is needed in order for the composition theorem to work. Here we show that the modular composition theorem holds even for simulators that do not operate via black-box simulation. Furthermore, it can be seen from the proof that if the security of the original protocols is proven via black-box simulation then the security of the composed protocol is also proven via black-box simulation. Also, if the simulations for the original protocols are of the restricted type described above then so is the simulator for the composed protocol.

Remark 2: On the complexity of the ideal-process adversary. We elaborate on the requirement that the complexity of the ideal-process adversary $\mathcal{S}$ be comparable to the complexity of the real-life adversary $\mathcal{A}$. The ideal-process adversary is an imaginary concept whose purpose is to formalize the following statement: "whatever the adversary learns from interacting with parties running π, could have also been learnt in the ideal process *in comparable computational effort.*" Capturing this requirement involves limiting the computational power of the ideal process adversary. In particular, if we do not limit the computational power of the ideal process adversary, we end up with a much weaker notion of security.

[13] A somewhat weaker definition allows the statistical distance (i.e., the total variation distance) between the two sides of (1) to be negligible.

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 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, where the adversary is computationally bounded. A convenient design paradigm for secure protocols in the computational setting proceeds as follows: first design a secure protocol π in the secure channels setting. Then, let protocol π' be identical to π except that each message is now encrypted. Indeed, if π is secure in the secure channels setting according to the definition here (and an appropriate encryption mechanism 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 3: Combining correctness and secrecy. We stress that in Equation (1) (Definition 1) the two *entire* $(n+1)$-vectors describing the output of the computation must be identically distributed. This captures the requirement that the adversary view and the outputs of the uncorrupted parties should be considered *together*. Let us elaborate.

Some previous definitions of secure computation (e.g., [21]) partitioned the security requirement into two separate requirements: (a) secrecy: the adversary's view of the computation is simulatable based on the inputs and specified outputs of the corrupted parties, and (b) correctness: the output of the uncorrupted parties is equal to their output in the ideal process. However, as discussed in Section 2 (and pointed out in [28]), separately requiring correctness and secrecy does not guarantee secure computation against active adversaries. Essentially, the problem with separately requiring secrecy and correctness is that is does not capture the influence that the adversary is "allowed" to have on the output of the uncorrupted parties. In particular, this approach does not put any limitations on the process by which the corrupted parties choose which input values to contribute to the computation.

For self-containment we sketch the [28] example that highlights this problem. Let $x = x_0x_1$. Let the computed function be $f(x,y) = x_1$ if $x_0 = 0$ and $f(x,y) = y$ otherwise, let party A hold x and let party B hold y. Consider the protocol where B first announces y, next A announces x, next all parties locally compute the function value. Intuitively (and also according to our definition), this protocol is *insecure* since it allows a corrupted A to choose x only after y is known. This way A can both learn y *and* decide on the output value, whereas in an ideal computation A could achieve only one of these two goals in any execution. Yet, this protocol is certainly correct (based on the input that A contributes to the computation), and also private since A can always learn y by choosing $x_0 = 1$.

In contrast, the approach taken here regards the inputs as fixed external values that are given "from above". The ideal process specifies the allowed process by which the contributed inputs of the corrupted parties are chosen; a secure protocol must guarantee that the real-life process is equivalent. Here the outputs of the uncorrupted parties and of the adversary are two aspects of the same process.

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-model adversary: the one that satisfies Definition 1 with respect to $\mathcal{U}$.

Another consequence of this observation follows. One may wish to strengthen Definition 1 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.

3.2 Modular composition: The non-adaptive case

Recall that we want to break down a given task to several partial tasks, design protocols for these partial tasks, and then use the already designed protocols as subroutines in the 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. Here we concentrate on non-adaptive adversaries in the secure channels setting.

The semi-ideal 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 semi-ideal model with ideal access to $f_1, ..., f_m$ (or in short the $(f_1, ..., f_m)$-ideal model), is obtained as follows. We start with the real-life model of Section 3.1. 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_j (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_j-inputs to T (party P_i hands x_i^f), and are handed back their respective outputs (P_i learns $f_j(x_1^f..x_n^f, r_{f_j})_i$). 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 $\text{EXEC}_{\pi,\mathcal{A}}^{f_1,...,f_m}(\vec{x}, z)$ denote the random variable describing the output of the computation in the $f_1, ..., f_m$-ideal model with protocol π, adversary $\mathcal{A}$, inputs $\vec{x}$ and auxiliary input z for the adversary, analogously to the definition of $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, z)$ in Section 3.1. (We stress that here π is not a real-life protocol and uses ideal calls to T.) Protocols for securely computing a function g in the $(f_1, ..., f_m)$-ideal model are defined via the usual comparison to the ideal process for g:

Definition 2 *Let $f_1, ..., f_m, g$ be n-party functions and let π be a protocol for n parties in the $(f_1, ..., f_m)$-ideal model. We say that π* non-adaptively t-securely computes g in the $(f_1, ..., f_m)$-ideal model *if for any non-adaptive t-limited adversary $\mathcal{A}$ (in the f-ideal 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 for every input vector $\vec{x}$ and every auxiliary input z,*

$$\text{IDEAL}_{g,\mathcal{S}}(\vec{x}, z) \stackrel{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A}}^{f_1,...,f_m}(\vec{x}, z). \tag{2}$$

If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries then we say that π non-adaptively t-privately computes g in the $(f_1, ..., f_m)$-ideal 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 a 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 input to ρ; in addition, ρ is given 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.[14] Let $\pi^{\rho_1 \cdots \rho_m}$ denote protocol π (originally designed for the $(f_1 ... f_m)$-ideal model) where each ideal evaluation call to f_i is replaced by a call to ρ_i.

We stress that the internal state of protocol $\pi^{\rho_1 \cdots \rho_m}$ (within some 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. In particular, protocol $\pi^{\rho_1 \cdots \rho_m}$ does not 'keep track' of the number of rounds that took ρ_i to terminate. This of course does not prevent the party from keeping track of round numbers in means external to protocol $\pi^{\rho_1 \cdots \rho_m}$, nor does it prevent the adversary from keeping track of round numbers.

Theorem 3 (non-adaptive modular composition for active adversaries) *Let $f_1, ..., f_m, g$ be n-party functions. Let π be an n-party protocol that non-adaptively t-securely computes g in the $(f_1, ..., f_m)$-ideal 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 (non-adaptively) t-securely compute $f_1, ..., f_m$, respectively. Then the protocol $\pi^{\rho_1, \cdots, \rho_m}$ non-adaptively t-securely computes g.*

The proof appears in Section 3.3 below. A similar (in fact, a considerably simpler) proof applies to the case of *passive* adversaries. We leave the proof as an easy exercise:

Theorem 4 (non-adaptive modular composition for passive adversaries) *Let $f_1, ..., f_m, g$ be n-party functions. Let π be an n-party protocol that non-adaptively t-privately computes g in the $(f_1, ..., f_m)$-ideal 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 (non-adaptively) t-privately computes $f_1, ..., f_m$, respectively. Then the protocol $\pi^{\rho_1, \cdots, \rho_m}$ non-adaptively t-privately computes g.*

3.3 Proof of Theorem 3

For simplicity we restrict the presentation to 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 straightforward (as long as at most one call to T is made at each round).

We first present an informal overview. Full details appear in Section 3.3.1 below. 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

$$\forall \vec{x}, z \quad \text{EXEC}_{\pi^\rho, \mathcal{A}}(\vec{x}, z) \stackrel{\text{d}}{=} \text{IDEAL}_{g, \mathcal{S}}(\vec{x}, z) \tag{3}$$

Intuitively, if $\mathcal{A}$ was partitioned into two separate "modules" — one module that operates solely against protocol ρ, and another module that operates solely against π in the f-ideal model — then we could use the security of ρ and the security of π to obtain ideal-process adversaries

[14] The assumption that all parties terminate at the same round is a real one. That is, it is possible to construct protocols where not all parties terminate at the same round, and where the modular composition theorem below is violated.

that (separately) simulate each one of these modules. If we could then make some simplifying assumptions on the behavior of the simulators (say, something like "one pass black box simulation"), then we would combine the outputs of these two simulators in the natural way, and obtain a simulator that (hopefully) 'properly simulates' $\mathcal{A}$.

Clearly, one cannot make such an assumption on $\mathcal{A}$ and the simulators. We will, nevertheless, do something quite similar to the above plan, without making any assumptions on $\mathcal{A}$ or the simulators. We proceed in four steps as follows:

1. We construct out of $\mathcal{A}$ real-life adversary, denoted $\mathcal{A}_\rho$, that operates against protocol ρ alone. We are guaranteed that $\mathcal{A}_\rho$ has a simulator (ie, an ideal-model adversary), $\mathcal{S}_\rho$, such that for all inputs $\vec{x}^\rho$ (to protocol ρ) and any auxiliary input z^ρ, we have $\text{EXEC}_{\rho,\mathcal{A}_\rho}(\vec{x}^\rho, z^\rho) \stackrel{\text{d}}{=} \text{IDEAL}_{f,\mathcal{S}_\rho}(\vec{x}^\rho, z^\rho)$.

2. Out of $\mathcal{A}$ and $\mathcal{S}_\rho$ we construct an adversary, $\mathcal{A}_\pi$, that operates against protocol π in the f-ideal model. We are guaranteed that this adversary has a simulator $\mathcal{S}_\pi$ such that for all inputs $\vec{x}$ (to protocol π) and any auxiliary input z, we have $\text{EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z) \stackrel{\text{d}}{=} \text{IDEAL}_{g,\mathcal{S}_\pi}(\vec{x}, z)$.

3. We combine $\mathcal{S}_\pi$ and $\mathcal{S}_\rho$ into one simulator, $\mathcal{S}$, that simulates the interaction of $\mathcal{A}$ with parties running π^ρ.

4. We analyze $\mathcal{S}$, showing (3).

Before getting into the details, let us sketch the above four steps. $\mathcal{A}_\rho$ control the same set of parties that $\mathcal{A}$ controls. It expects its auxiliary input to describe a prefix of an execution of $\mathcal{A}$ against protocol π^ρ up to the round, l, where ρ is invoked. (If the auxiliary input is inconsistent then $\mathcal{A}_\rho$ halts.) Next, $\mathcal{A}_\rho$ continues the run of $\mathcal{A}$ and follows $\mathcal{A}$'s instructions. We will show how $\mathcal{A}_\rho$ is able to 'fill up' all the information that $\mathcal{A}$ expects to see. At the end of its interaction (with parties running ρ), adversary $\mathcal{A}_\rho$ outputs its entire view of the computation.

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. 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 $\mathcal{S}_\rho$ (playing the trusted party for f). The auxiliary input of $\mathcal{S}_\rho$ will describe the *current* interaction between $\mathcal{A}_\pi$ and its network, up to round l. Recall that $\mathcal{S}_\rho$ outputs the entire view of an interaction between $\mathcal{A}$ and parties running ρ; for the duration of protocol ρ adversary $\mathcal{A}_\pi$ runs $\mathcal{A}$ on the output of $\mathcal{S}_\rho$. We will be guaranteed that the output of $\mathcal{S}_\rho$ indeed 'matches' the execution of $\mathcal{A}$ so far. Once protocol ρ completes its execution and the parties return to running π, adversary $\mathcal{A}_\pi$ returns to following the instructions of $\mathcal{A}$. Finally, $\mathcal{A}_\pi$ outputs its entire view of the computation.

It remains to describe the construction of $\mathcal{S}$. In a way, we are now 'undoing' the de-construction of $\mathcal{A}$ to $\mathcal{A}_\rho$ and $\mathcal{A}_\pi$, performed the previous two steps. That is, $\mathcal{S}$ starts by running $\mathcal{S}_\pi$ and following $\mathcal{S}_\pi$'s instructions. When $\mathcal{S}_\pi$ invokes its trusted party (for g), $\mathcal{S}$ invokes its own trusted party for g. Once $\mathcal{S}_\pi$ generates its output (this output is a simulated view of an interaction of $\mathcal{A}_\pi$), simulator $\mathcal{S}$ invokes $\mathcal{S}_\rho$, in a way that mimics the way $\mathcal{A}_\pi$ invoked $\mathcal{S}_\rho$ (the data for running $\mathcal{S}_\rho$ is taken from $\mathcal{S}_\pi$'s output in the natural way). Finally, $\mathcal{S}$ replaces, in the output of $\mathcal{S}_\pi$, the description of round l with the output of $\mathcal{S}_\rho$ (and makes some other obvious modifications). The result is $\mathcal{S}$'s output.[15]

Before sketching our analysis of $\mathcal{S}$ let us address one subtle 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

[15]Here we assume that $\mathcal{A}$ outputs its entire view. Otherwise, $\mathcal{S}$ runs $\mathcal{A}$ on the obtained view and outputs whatever $\mathcal{A}$ outputs.

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 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. We formalize and elaborate on this argument in the full proof below.

Showing (3) is done as follows. Fix an input vector $\vec{x}$ and auxiliary input z. First we define two 'hybrid' distributions. The first, denoted $\text{MOD-EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z)$, is essentially constructed from $\text{EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z)$ by replacing, in $\mathcal{A}_\pi$'s output, the description of round l with the output of the invocation of $\mathcal{S}_\rho$ made during that run of $\mathcal{A}_\pi$. (This is a purely 'syntactic' modification that can be done by inspecting $\mathcal{A}_\pi$'s output alone.) The second hybrid distribution, denoted $\text{MOD-IDEAL}_{g,\mathcal{S}_\pi}(\vec{x},z)$, is obtained by applying the same procedure to $\text{IDEAL}_{g,\mathcal{S}_\pi}(\vec{x},z)$. Next, we show that:

$$\text{EXEC}_{\pi^\rho,\mathcal{A}}(\vec{x},z) \stackrel{\text{d}}{=} \text{MOD-EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z) \stackrel{\text{d}}{=} \text{MOD-IDEAL}_{g,\mathcal{S}_\pi}(\vec{x},z) \stackrel{\text{d}}{=} \text{IDEAL}_{g,\mathcal{S}}(\vec{x},z) \tag{4}$$

The rightmost equality follows immediately from the construction of $\mathcal{S}$. The middle equality follows from the security of π in the f-ideal model. The leftmost equality follows (after some work) from the security of protocol ρ.

3.3.1 A Detailed proof

We flesh out the details of the proof outline presented above. As there, we restrict the presentation to the case where $m = 1$ and the trusted party is called only once.

Let $\mathcal{A}$ be an adversary (interacting with parties running π^ρ). First we present the constructions of adversaries $\mathcal{A}_\rho$, $\mathcal{A}_\pi$ and of simulator $\mathcal{S}$, proving two claims (Claims 5 and 6) along the way. Next we analyze $\mathcal{S}$, showing (3).

We remark that the analysis of $\mathcal{S}$ does not directly use Claims 5 and 6, thus they may be omitted without harming the proof. Yet these claims are instrumental in highlighting crucial points and in gaining confidence in the construction en-route.

Some inevitable terminology. Although some of the notions defined here are used only towards the end of the proof, we hope it is clearer to define them together at this early stage.

An execution of a protocol (either in the real-life or in the f-ideal model) is the process of running the protocol in the appropriate model with the appropriate parameters. It is determined by the adversary as well as by the inputs, random inputs, and auxiliary inputs of the participating entities. (In the f-deal model it is determined also by the trusted party for f.) The transcript of an execution contains all the relevant information pertaining to an execution. This includes the inputs, random inputs and auxiliary input of the parties and adversary, all the messages exchanged during the protocol, and the outputs of the parties and the adversary.[16] We often do not distinguish between an execution and its transcript. A prefix of a transcript up to some round, ℓ, is the string

[16]The information in the transcript is arranged as follows. First appear the inputs of all parties, their random inputs (as well as the random input of the trusted party for f, in the f-ideal model), and the adversary's auxiliary input. These are followed by a sequence of round configurations. The ℓth round configuration describes, in some standard format, the instantaneous descriptions of all the parties and the adversary at the end of round ℓ. This includes the internal data of the uncorrupted parties and of the adversary, and in addition all the messages sent in this round.

obtained from the full transcript by removing all round configurations describing rounds $\ell+1$ and onwards. In particular, the prefix contains even the random inputs that are to be used *after* round ℓ.

Let $\text{TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x},z,\vec{r})$ denote the transcript of an execution of protocol π^ρ in the real-life model with adversary $\mathcal{A}$, inputs $\vec{x}$, random inputs $\vec{r}$ and auxiliary input z. Let $\text{TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z,\vec{r})$ be defined similarly with respect to protocol π and adversary $\mathcal{A}_\pi$ in the f-ideal model.

The adversary view of some execution is the part of the transcript that is available to the adversary. This consists of the adversary auxiliary and random inputs, followed by the corrupted parties' inputs, random inputs, and all the messages sent and received by the corrupted parties during the computation.

A string s is a valid view of $\mathcal{A}$ if s contains a 'consistent' view of *some* execution in which $\mathcal{A}$ participates. (We sometimes say that 's is consistent with $\mathcal{A}$'.) That is, s first holds $\mathcal{A}$'s auxiliary input z, as well as the inputs $\vec{x}_C$ and random inputs $\vec{r}_C$ of the set C of corrupted parties. next s describes some information that $\mathcal{A}$ receives in the first communication round (presumably this information consists of messages sent to the corrupted parties); Next s holds the correct messages that $\mathcal{A}$ sends in the first round of the interaction on $z,\vec{x}_C,\vec{r}_C$ and having received the above messages, and so on until the end of the interaction. If s is a valid view with the exception that it describes an interaction only until some round, ℓ, then we say that s is a valid prefix of $\mathcal{A}$'s view. Also here we assume that s contains all of $\vec{r}_C$, including the random bits to be used after round ℓ is completed.

"Running $\mathcal{A}$ on view s" means invoking $\mathcal{A}$ on the inputs, random inputs and auxiliary inputs contained in the view s; whenever $\mathcal{A}$ expects to receive information from the network it is given the appropriate next piece of s, according to some standard encoding.

In the sequel we will often not be strictly formal in treating the way in which transcripts and adversary views are encoded into strings. (For instance, the same string may sometimes encode a prefix of an adversary view, and at other times encode an auxiliary input describing such a prefix.) A formal treatment can be extracted from the current one.

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 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 prefix of $\mathcal{A}$'s view, ending at round $l-1$. This is done by using z^ρ to simulate a run of $\mathcal{A}$ up to this round. If z^ρ is not a valid prefix (i.e., if at some point $\mathcal{A}$ answers differently from the answer specified in z^ρ) then $\mathcal{A}_\rho$ halts with no output.

3. $\mathcal{A}_\rho$ continues the above run of $\mathcal{A}$ from round l on, follows $\mathcal{A}$'s instructions, and hands the learned information to $\mathcal{A}$. Following $\mathcal{A}$'s instruction consists of:

   (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 its entire view of the computation.

It now follows from the security of protocol ρ that there exists an ideal-process adversary $\mathcal{S}_\rho$ such that for all inputs $\vec{x}$ and auxiliary input $\vec{z}$, $\text{IDEAL}_{f,\mathcal{S}_\rho}(\vec{x},z) \stackrel{\text{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho}(\vec{x},z)$.

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 equal to $\text{EXEC}_{\rho,\mathcal{A}_\rho}(\vec{x},z)$. 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. Recall that $\mathcal{A}_\rho$ ignores the inputs of the corrupted parties. Consequently, the simulator $\mathcal{S}_\rho$ 'does not care about' the input values for these parties. That is, given some output v of $\mathcal{A}_\rho$, let the corresponding **reduced output** v' be the string obtained from v by deleting the inputs of the corrupted parties. Let the reduced output of simulator $\mathcal{S}_\rho$ be defined in the same way. Informally, it now follows from the definition of $\mathcal{A}_\rho$ that the distribution of the reduced output of $\mathcal{S}_\rho$ remains unchanged regardless of the input values of the corrupted parties. In particular, the distribution of the reduced output of $\mathcal{S}_\rho$ remains unchanged if we set the input values of the corrupted parties to 0.

We formalize this discussion as follows. Let $\text{ADV}_{f,\mathcal{S}_\rho}(\vec{x}^\rho,z^\rho)$ denote the output distribution of $\mathcal{S}_\rho$ on inputs $\vec{x}^\rho = x_1^\rho,...,x_n^\rho$ for the parties and auxiliary input z^ρ. Let $\text{RED-ADV}_{f,\mathcal{S}_\rho}(\vec{x}^\rho,z^\rho)$ denote the distribution of the *reduced* output of $\mathcal{S}_\rho$ on $\vec{x}^\rho$ and z^ρ. 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,

Claim 5 *For any input vector $\vec{x}^\rho$ and auxiliary input z^ρ we have*

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

Proof: If z^ρ is not a valid prefix of $\mathcal{A}$ then $\mathcal{A}_\rho$ halts with no output and the claim holds. Otherwise, notice that $\mathcal{A}_\rho$ is instructed to ignore the inputs of the corrupted parties. (Instead, $\mathcal{A}_\rho$ continues following $\mathcal{A}$'s instructions regarding those parties.) Thus, in any execution of $\mathcal{A}_\rho$ the reduced output does not depend on the inputs of the corrupted parties in any way. Therefore,

$$\text{RED-ADV}_{\rho,\mathcal{A}_\rho}(\vec{x}^\rho,z^\rho) \stackrel{\text{d}}{=} \text{RED-ADV}_{\rho,\mathcal{A}_\rho}(\vec{x}^\rho|_0,z^\rho),$$

where $\text{RED-ADV}_{\rho,\mathcal{A}_\rho}(\vec{x}^\rho,z^\rho)$ denotes the distribution of the reduced output of $\mathcal{A}_\rho$ interacting with parties running ρ on inputs $\vec{x}^\rho$ and auxiliary inputs z^ρ. However, $\text{RED-ADV}_{f,\mathcal{S}_\rho}(\vec{x}^\rho,z^\rho) \stackrel{\text{d}}{=}$ $\text{RED-ADV}_{\rho,\mathcal{A}_\rho}(\vec{x}^\rho,z^\rho)$, and $\text{RED-ADV}_{f,\mathcal{S}_\rho}(\vec{x}^\rho|_0,z^\rho) \stackrel{\text{d}}{=} \text{RED-ADV}_{\rho,\mathcal{A}_\rho}(\vec{x}^\rho|_0,z^\rho)$. 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, $\mathcal{A}$ 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 prefix of $\mathcal{A}$'s view of the current run up to round $l-1$. 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 reduced output of $\mathcal{S}_\rho$. See Claim 5.)

(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 with the same input values for the corrupted parties, and hands the answer back to $\mathcal{S}_\rho$.

3. Once $\mathcal{S}_\rho$ has produced its output, $\mathcal{A}_\pi$ removes from this output the input values of the corrupted parties, and obtains the reduced output of $\mathcal{S}_\rho$. Let v' denote the obtained value.

4. $\mathcal{A}_\pi$ uses v' to continue the current run of $\mathcal{A}$ from round l until the completion of protocol ρ. If v' is not consistent with $\mathcal{A}$ (that is, if at any stage $\mathcal{A}$ generates a different value from the value described in v) then $\mathcal{A}_\pi$ halts. (See the discussion and Claim 6 below.)

5. Once $\mathcal{A}$ has completed the simulated part of its interaction and the parties have returned to running π, adversary $\mathcal{A}_\pi$ returns to following $\mathcal{A}$'s instructions until the completion of protocol π.

6. Once protocol π is completed, $\mathcal{A}_\pi$ outputs its entire view of the interaction.

We are guaranteed that there exists a simulator $\mathcal{S}_\pi$ such that for every input $\vec{x}$ and every auxiliary input z we have $\text{IDEAL}_{g,\mathcal{S}_\pi}(\vec{x},z) \stackrel{\text{d}}{=} \text{EXEC}^{f}_{\pi,\mathcal{A}_\pi}(\vec{x},z)$.

Notice that, in the construction of $\mathcal{A}_\pi$, the output of $\mathcal{S}_\rho$ is a valid continuation of the current run of $\mathcal{A}$. This is formalized as follows. Consider an execution of $\mathcal{A}_\pi$, and let w denote $\mathcal{A}_\pi$'s view of this execution. Let v' denote the reduced output of $\mathcal{S}_\rho$ within this execution. Obtain v'' by removing the auxiliary information z^ρ from v'. Next, take w and replace the part describing $\mathcal{A}_\pi$'s view of round l with v''. Furthermore, delete the random bits used by $\mathcal{A}_\pi$ to run $\mathcal{S}_\rho$ from w. Call the result the **modified view of $\mathcal{A}_\pi$**. The following claim is implied by Claim 9 below. Yet, we present a special-purpose proof. (Of course, this proof can be skipped.)

Claim 6 *Let w' be the modified view of $\mathcal{A}_\pi$ in some execution of $\mathcal{A}_\pi$. Then w' is a valid view of $\mathcal{A}$.*

Proof: Consider a run of protocol $\mathcal{A}_\rho$, and assume that the auxiliary input, z^ρ, is a valid prefix of $\mathcal{A}$'s view up to round l of some real interaction between $\mathcal{A}$ and parties running π^ρ. In this case, $\mathcal{A}_\rho$ continues the run of $\mathcal{A}$ whose prefix is described in z^ρ. Therefore, the reduced output of $\mathcal{A}_\rho$ is a valid prefix of $\mathcal{A}$'s view up to the conclusion of protocol ρ. (The reduced output of $\mathcal{A}_\rho$ is obtained from the output of $\mathcal{A}_\rho$ by applying the same procedure used to obtain the reduced output of $\mathcal{S}_\rho$ from the output of $\mathcal{S}_\rho$.) [17]

We have that $\mathcal{A}_\rho$ is deterministic, and that the reduced output of $\mathcal{S}_\rho$ is distributed identically to the reduced output of $\mathcal{A}_\rho$. Furthermore, $\mathcal{A}_\pi$ sets $\mathcal{S}_\rho$'s auxiliary input to a valid prefix of $\mathcal{A}$'s view of the current interaction with the parties running π. It follows that the prefix of w' describing the interaction up to the conclusion of round l is a valid prefix of $\mathcal{A}$'s view up to this point; this holds for *any* random input of $\mathcal{S}_\rho$.

From the conclusion of round l and onwards $\mathcal{A}_\pi$ follows $\mathcal{A}$'s instructions, thus the rest of v' remains consistent with $\mathcal{A}$'s program. □

Construction of $\mathcal{S}$. Informally, $\mathcal{S}$ 'reproduces' a simulated transcript of the interaction between $\mathcal{A}_\pi$ and $\mathcal{A}$, by combining the outputs of $\mathcal{S}_\rho$ and $\mathcal{S}_\pi$. This is done as follows.

[17] In fact, some trivial representational modifications need to be applied to the reduced output of $\mathcal{A}_\rho$ to make it fit the syntax expected from a valid prefix of $\mathcal{A}$'s view. In particular, the string z^ρ should no longer be regarded as an auxiliary input, but rather as an actual view of $\mathcal{A}$.

1. $\mathcal{S}$ first runs $\mathcal{S}_\pi$ on its ($\mathcal{S}$'s) own auxiliary input and random input, and with its own inputs and random inputs for the corrupted parties.

2. When $\mathcal{S}_\pi$ hands its trusted party the inputs of the corrupted parties and asks for the function (ie, g) values, $\mathcal{S}$ acts accordingly: it invokes its own trusted party with the same input values for the corrupted parties, and hands the function values back to $\mathcal{S}_\pi$.

3. When $\mathcal{S}_\pi$ generates its output, $\mathcal{S}$ records this output in a variable w. Recall that w is a valid view of an execution of $\mathcal{A}_\pi$.

4. Next $\mathcal{S}$ runs simulator $\mathcal{S}_\rho$, in the same way that $\mathcal{A}_\pi$ would, having view w. That is:

   (a) $\mathcal{S}_\rho$'s auxiliary input, z^ρ, is taken from the part of w describing the prefix of $\mathcal{A}$'s view up to round $l-1$.

   (b) $\mathcal{S}_\rho$'s random input is taken from the part of w describing the random input that $\mathcal{A}_\pi$ uses to run $\mathcal{S}_\rho$.

   (c) $\mathcal{S}_\rho$ is given input values 0 to the corrupted parties.

   (d) When $\mathcal{S}_\rho$ hands the inputs of the corrupted parties to its trusted party and asks for the function (ie, f) values, $\mathcal{S}$ looks up w to find the interaction between the simulated $\mathcal{A}_\pi$ and T at round l of the computation. We are guaranteed that in that interaction $\mathcal{A}_\pi$ hands T the same input values that $\mathcal{S}_\rho$ hands its trusted party. Now $\mathcal{S}$ finds in w the values that T hands $\mathcal{A}_\pi$ and hands those values to $\mathcal{S}_\rho$ as the values given by the trusted party.

5. When $\mathcal{S}_\rho$ generates its output, v, $\mathcal{S}$ computes the corresponding *reduced output,* v', in the same way that $\mathcal{A}_\pi$ would do having view w.

6. $\mathcal{S}$ combines v' and w in order to obtain a full view of $\mathcal{A}$. That is, first $\mathcal{S}$ removes from w the part of $\mathcal{A}_\pi$'s random input that is used to run $\mathcal{S}_\rho$. Next $\mathcal{S}$ removes from w the part that describes round l, and replaces it with v'. Let y denote the obtained string. It follows from Claim 6 and the description of $\mathcal{A}_\pi$ that y is a valid view of $\mathcal{A}$.

7. Finally, $\mathcal{S}$ runs $\mathcal{A}$ on the view y, and outputs whatever $\mathcal{A}$ outputs. (If $\mathcal{A}$ outputs its entire view then this step is not necessary and $\mathcal{S}$ outputs y.)

Analysis of $\mathcal{S}$. Fix an input vector $\vec{x}$ and auxiliary input z for the parties and adversary, as well as some value of the security parameter. We show that $\text{IDEAL}_{g,\mathcal{S}}(\vec{x},z) \stackrel{\text{d}}{=} \text{EXEC}_{\pi^\rho,\mathcal{A}}(\vec{x},z)$. Recall that $\text{EXEC}_{\pi^\rho,\mathcal{A}}(\vec{x},z)$ is the random variable describing the outputs of all parties running π^ρ in the real-life model, plus the adversary's output. Similarly, the random variable $\text{IDEAL}_{g,\mathcal{S}}(\vec{x},z)$ describes the outputs of all parties of the ideal process for computing g, plus the output of $\mathcal{S}$. The random variable $\text{EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z)$ has a similar meaning with respect to protocol π in the f-ideal model and adversary $\mathcal{A}_\pi$. We first define the following two 'hybrid' distributions.

- Let $\text{MOD-EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z)$ be obtained from the random variable $\text{EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z)$, by replacing the output of $\mathcal{A}_\pi$ (i.e., its view) with its *modified* view. (The modified view is defined in the preamble to Claim 6.) The outputs of the uncorrupted parties are not modified.

- Let $\text{MOD-IDEAL}_{g,\mathcal{S}_\pi}(\vec{x},z)$ denote the random variable obtained by applying the same procedure to $\text{IDEAL}_{g,\mathcal{S}_\pi}(\vec{x},z)$ (with respect to the output of $\mathcal{S}_\pi$).

We complete the proof via the following three claims.

Claim 7 $\text{MOD-IDEAL}_{g,\mathcal{S}_\pi}(\vec{x}, z) \stackrel{\rm d}{=} \text{IDEAL}_{g,\mathcal{S}}(\vec{x}, z)$.

Proof: This claim is immediate from the definition of $\mathcal{S}$. □

Claim 8 $\text{MOD-EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z) \stackrel{\rm d}{=} \text{MOD-IDEAL}_{g,\mathcal{S}_\pi}(\vec{x}, z)$.

Proof: Recall that $\text{EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z) \stackrel{\rm d}{=} \text{IDEAL}_{g,\mathcal{S}_\pi}(\vec{x}, z)$. The modified variables $\text{MOD-EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z)$ and $\text{MOD-IDEAL}_{g,\mathcal{S}_\pi}(\vec{x}, z)$ are obtained from the unmodified ones by applying the same (deterministic) procedure. □

Claim 9 $\text{EXEC}_{\pi^\rho,\mathcal{A}}(\vec{x}, z) \stackrel{\rm d}{=} \text{MOD-EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z)$.

Proof: This proof contains the "essence" of the analysis. We start by defining the following notions.

- Recall the definition of the transcript of π^ρ in the real-life model, $\text{TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x}, z, \vec{r})$. Let the **reduced transcript of** π^ρ, denoted $\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x}, z, \vec{r})$, be constructed from $\text{TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x}, z, \vec{r})$ by removing the all the data contained in the round configurations pertaining to the execution of protocol ρ, except for the inputs and outputs of the parties and the information appearing in $\mathcal{A}$'s view of this execution. That is, first remove from $\text{TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x}, z, \vec{r})$ the random inputs and internal data of the uncorrupted parties pertaining to protocol ρ. Then remove the messages sent among the uncorrupted parties during the execution of ρ. The local inputs and outputs of the uncorrupted parties from protocol ρ are *not* removed.
- Recall the definition of the transcript of π in the f-ideal model, $\text{TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z, \vec{r})$. Let the **modified transcript of** π, denoted $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z, \vec{r})$, be constructed from $\text{TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z, \vec{r})$ by replacing $\mathcal{A}_\pi$'s view with $\mathcal{A}_\pi$'s modified view (as defined in the preamble to Claim 6).

In all the above constructs, if the random input $\vec{r}$ is not specified, then the notation describes the random variable obtained by choosing $\vec{r}$ at random.

Note that, for each $\vec{r}$, the value $\text{EXEC}_{\pi^\rho,\mathcal{A}}(\vec{x}, z, \vec{r})$ is obtained from $\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x}, z, \vec{r})$ via a (simple) deterministic procedure. (This procedure extracts the outputs of the uncorrupted parties and the adversary view from $\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x}, z, \vec{r})$.) Furthermore, $\text{MOD-EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z, \vec{r})$ is obtained from $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z, \vec{r})$ by applying the *same deterministic procedure*. Consequently, it suffices to show that

$$\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x}, z) \stackrel{\rm d}{=} \text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z). \tag{5}$$

To show (5), define the following prefixes of $\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x}, z, \vec{r})$ and of $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z, \vec{r})$. The first prefixes, denoted $\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x}, z, \vec{r})^{(1)}$ and $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z, \vec{r})^{(1)}$, are the transcripts of the corresponding computations until and excluding round l.

The second prefixes, denoted $\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x}, z, \vec{r})^{(2)}$ (resp., $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z, \vec{r})^{(2)}$), are obtained by removing from $\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x}, z, \vec{r})$ (resp., from $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x}, z, \vec{r})$) all the round configurations describing the interaction from the point where protocol ρ completes its execution (resp., from round $l + 1$) and on. We proceed in three steps:

1. Let $\vec{r}^{(1)}$ denote the part of $\vec{r}$ that is used until (and excluding) round l. Note that, although the random inputs to the two executions are of different lengths, the part pertaining to the executions until round l are of equal length. Furthermore, for each value of $\vec{r}^{(1)}$ we have $\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x},z,\vec{r}^{(1)})^{(1)} = \text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z,\vec{r}^{(1)})^{(1)}$. This holds since until round l protocols π^ρ and π are identical, and adversary $\mathcal{A}_\pi$ follows $\mathcal{A}$'s instructions.

2. Fix a value for $\vec{r}^{(1)}$, and let $\vec{x}^\rho$ denote the input values of the parties for protocol ρ in $\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x},z,\vec{r}^{(1)})^{(1)}$. (The input values of the corrupted parties are set to 0.) Let z^ρ denote $\mathcal{A}$'s view of this interaction up to round l. Similarly, let $\vec{x}^f$ denote the input values of the uncorrupted parties for the trusted party for f in $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z,\vec{r}^{(1)})^{(1)}$ (inputs of the corrupted parties set to 0), and let z^f denote $\mathcal{A}$'s view of its interaction with $\mathcal{A}_\pi$ (within $\mathcal{A}_\pi$'s code). Then, it follows from Step 1 that $\vec{x}^\rho = \vec{x}^f$ and $z^\rho = z^f$. Consequently, $\text{EXEC}_{\rho,\mathcal{A}_\rho}(\vec{x}^\rho,z^\rho) \stackrel{\text{d}}{=} \text{IDEAL}_{f,\mathcal{S}_\rho}(\vec{x}^f,z^f)$.

   Notice that the part of $\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x},z,\vec{r})^{(2)}$ that pertains to protocol ρ is in fact the *global output* of an interaction of parties running ρ with adversary $\mathcal{A}_\rho$ on inputs $\vec{x}^\rho$ for the uncorrupted parties, and auxiliary input z^ρ. Similarly, the part of $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z,\vec{r})^{(2)}$ that pertains to round l is in fact the *global output* of the ideal process, on inputs $\vec{x}^f$ for the uncorrupted parties and after interacting with ideal-process adversary $\mathcal{S}_\rho$ having auxiliary input z^f.[18] It follows that, for each fixed value of $\vec{r}^{(1)}$, the random variables $\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x},z)^{(2)}$ and $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z)^{(2)}$ are identically distributed. Consequently,

$$\text{RED-TRANS}_{\pi^\rho,\mathcal{A}}(\vec{x},z)^{(2)} \stackrel{\text{d}}{=} \text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z)^{(2)}. \tag{6}$$

3. From the resumption of protocol π until its conclusion, adversary $\mathcal{A}_\pi$ again follows the instructions of $\mathcal{A}$. Equation (5) follows since the two sides of (5) are obtained by applying the same (randomized) procedure to the corresponding two sides of (6).

The claim follows. $\square$

Remark: On the propagation of statistical distance. A somewhat relaxed version of Definition 1 allows the two sides of (1) to have negligible statistical distance. The composition theorem holds even for this case. In addition, it can be readily seen that the statistical distance is preserved in a very tight sense. That is, if the statistical distance between $\text{EXEC}_{\pi^\rho,\mathcal{A}}(\vec{x},z)$ and $\text{IDEAL}_{g,\mathcal{S},}(\vec{x},z)$ is δ (for some $\mathcal{A}$ and some values of $\vec{x}$ 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 $\text{EXEC}^f_{\pi,\mathcal{A}_\pi}(\vec{x},z)$ and $\text{IDEAL}_{g,\mathcal{S}_\pi}(\vec{x},z)$ is δ_1.
- The statistical distance between $\text{EXEC}_{\rho,\mathcal{A}_\rho}(\vec{x}^\rho,z^\rho)$ and $\text{IDEAL}_{f,\mathcal{S}_\rho}(\vec{x}^f,z^f)$ is δ_2, where $\vec{x}^\rho,z^\rho,\vec{x}^f,z^f$ are the values defined in Step 2 of Claim 9.

A similar statement holds with respect to the adaptive case (Theorem 12).

[18] Here we use the fact that $\mathcal{A}_\rho$ disregards the inputs of the corrupted parties. While Claim 5 is proven using similar arguments, it is not directly necessary here.

4 Adaptive adversaries

This section defines secure protocols, presents and proves the composition theorem for the case of adaptive adversaries. Remark 1 in Section 4.1.1 presents

a motivating example that highlights the additional power of adaptive security on top of non-adaptive security.

This section attempts to be as self-contained as possible, at the price of being somewhat repetitious. Still, in cases where the text is very similar to the non-adaptive case with immediate modifications we refer the reader to the corresponding parts of Section 3. It is thus recommended to first get familiarized with the non-adaptive case (Section 3).

Throughout this section we restrict the presentation to the secure channels setting. The computational setting is dealt with in the Section 5. Section 4.1 contains the definition of secure protocols. Section 4.2 presents the composition theorem, to be proven in Section 4.3.

4.1 Adaptive security of protocols

We flesh out the approach outlined in Section 2 for the case of adaptive adversaries. 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; next the notion of emulation (and the definition) is presented.

All the remarks made in Section 3.1.1 and in footnotes throughout Section 3.1 are relevant here as well, but are not repeated. In addition, Section 4.1.1 holds several remarks specific to the 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.[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.

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 4.1.1.) From this point on $\mathcal{A}$ learns all the messages received by the party. If $\mathcal{A}$ is passive then even 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 that was not yet *activated* in this round and activates it. Upon activation, the party sends its messages for this round (based on its internal state and

[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 auxiliary information gathered by the adversary when corrupting a party. We stress that the protocol run by the party does not use the auxiliary input. It is there only for the adversary to see upon corruption.

the messages that were sent to it in the previous round), and the next mini-round begins. $\mathcal{A}$ learns the messages sent 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. Finally, all uncorrupted parties receive the messages sent to them in this round.

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, $\mathcal{A}$ 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 current execution be the concatenation of the outputs of all parties and of the adversary.

Post-protocol corruption. To be able to prove the composition theorem in the adaptive case (and following [2]), we introduce an additional structure to the adversary. This structure models the fact that if the parties are running also other protocols together with the current one, then a party can get corrupted even long after the current execution is completed. The corruption requests can depend both on the current execution and on the other executions; in particular, they cannot be predicted beforehand. Once an adversary makes such a "late"corruption request, it expects to see the corrupted party's internal data pertaining to all protocol invocations run by the party, and in particular the data pertaining to the current execution.

The real-life model is augmented as follows. Once the execution of the protocol is complete, we let the adversary $\mathcal{A}$ interact with a **post-protocol corruptor** $\mathcal{P}$. Corruptor $\mathcal{P}$ is another (computationally unbounded) interactive Turing machine, that starts off with an auxiliary input and some random input; 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, $\mathcal{A}$ proceeds as follows. If P_i is already corrupted, or t parties are already corrupted, then the request is ignored. Otherwise, $\mathcal{A}$ corrupts P_i and answers with the internal data of P_i. The interaction continues until $\mathcal{P}$ halts. Once the interaction is completed the view of $\mathcal{P}$, including $\mathcal{P}$'s auxiliary and random inputs, as well as all corruption requests and answers, is added to the global output. See further discussion in Remark 3, Section 4.1.1.

We use the following notation. Let $\text{POST}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})$ denote $\mathcal{P}$'s view of the interaction with an adversary $\mathcal{A}$ that has interacted with parties running protocol π on input $\vec{x} = x_1 \dots x_n$, random input $\vec{r} = r_P, r_0 \dots r_n$ and auxiliary input $\vec{z} = z_P, z_0, \dots z_n$ 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 $\text{ADV}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})$ denote the output of real-life adversary $\mathcal{A}$ in this execution.[20] Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})_i$ 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 $\text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})_i = \perp$. Let

$$\text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r}) = \text{POST}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r}), \text{ADV}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r}), \text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})_1 \dots \text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})_n$$

denote the **global output** of the computation in the real-life model.[21] Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z})$ denote the random variable describing $\text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})$ where $\vec{r}$ is uniformly chosen.

[20] Although it is not necessary for computationally unbounded adversaries (and in our non-uniform complexity model it does not even necessarily lower the complexity of the adversary), we let the adversary have random input. This may make the model more 'natural' and easier to envision.

[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-protocol corruptor.

The ideal process. The model is parameterized by a probabilistic function $f : D^n \times \mathcal{R}_f \to E^n$, where $\mathcal{R}_f$ is taken to be the random input. (We call such functions n-party functions.) Each party P_i has input $x_i \in D$ and auxiliary input z_i; no random input is needed. An adaptive ideal-process-adversary $\mathcal{S}$ is an interactive Turing machine that has auxiliary input z_0 and random input r_0.[22] In addition, there is an (incorruptible) trusted party, T. 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 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 $\vec{y}$ be the n-vector constructed from the input vector $\vec{x}$ by substituting the entries of the corrupted parties by the corresponding entries in $\vec{b}$. That is, T receives the vector $\vec{y}$ from the parties (y_i is received from P_i). Next, T chooses $r_f \stackrel{R}{\leftarrow} \mathcal{R}_f$, and hands each P_i the value $f(\vec{y}, r_f)_i$.

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 $f(\vec{y}, r_f)_i$, 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 $\{f(\vec{y}, r_f)_i : \ P_i \text{ is corrupted}\}$. Let the early global output of the ideal process be the concatenation of the outputs of all parties and of the adversary.

Post-protocol corruption: Once the outputs are generated, $\mathcal{S}$ engages in an interaction with post-protocol corruptor $\mathcal{P}$, similar to the interaction of $\mathcal{A}$ with $\mathcal{P}$ in the real-life model. Here $\mathcal{P}$ 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, the view of $\mathcal{P}$ is added to the global output.

Let $\text{POST}_{f,\mathcal{S},\mathcal{P}}(\vec{x}, \vec{z}, \vec{r})$, where $\vec{r} = r_P, r_0, r_f$, denote the view of post-protocol corruptor $\mathcal{P}$ (with random input r_P), after interacting with an ideal-process adversary $\mathcal{S}$; prior to the interaction with $\mathcal{P}$, adversary $\mathcal{S}$ has interacted with parties having input $\vec{x} = x_1 \ldots x_n$ and auxiliary input $\vec{z} = z_P, z_0, \ldots z_n$, and with a trusted party for computing f with random input r_f. Let $\text{ADV}_{f,\mathcal{S},\mathcal{P}}(\vec{x}, \vec{z}, \vec{r})$, denote the output of adversary $\mathcal{S}$ from this execution. Let the $(n+1)$-vector

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

[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.

denote the global output of the ideal process on inputs $\vec{x}, \vec{z}$, adversary $\mathcal{S}$, post-protocol corruptor $\mathcal{P}$ and random inputs $\vec{r}$ as described above (party P_i outputs $\text{IDEAL}_{f,\mathcal{S},\mathcal{P}}(\vec{x}, \vec{z}, \vec{r})_i$). Let $\text{IDEAL}_{f,\mathcal{S},\mathcal{P}}(\vec{x}, \vec{z})$ denote the distribution of $\text{IDEAL}_{f,\mathcal{S},\mathcal{P}}(\vec{x}, \vec{z}, \vec{r})$ when $\vec{r}$ is uniformly distributed.

Comparing computations in the two models. This part is identical to the non-adaptive case (Section 3.1). In particular, the remarks of Section 3.1.1 apply here as well.

Definition 10 (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 real-life adversary $\mathcal{A}$, and any post-protocol 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 for every input vector $\vec{x}$ and every auxiliary input vector $\vec{z}$,*

$$\text{IDEAL}_{f,\mathcal{S},\mathcal{P}}(\vec{x}, \vec{z}) \stackrel{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x}, \vec{z}). \qquad (7)$$

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

4.1.1 Discussion

Remark 1: Adaptive versus non-adaptive security. It is easy to see that an adaptively secure protocol is also non-adaptively secure. Yet, the converse does not hold. We demonstrate the difference between adaptive and non-adaptive security of protocols 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. By letting the adversary know, upon corrupting a party in the real-life model, 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 unverifiable 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 of 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 reasonable and necessary. An example is the case of proactive security [29, 10], where the parties are servers controlled by a single authority, and use erasures to maintain security in the face of repeated break-ins. In such case $\mathcal{A}$ learns only the corrupted party's *current* state.

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

Remark 3: On post-protocol corruption. The post-protocol corruptor $\mathcal{P}$ models the corruption requests that result from a "larger" interaction, of which the current protocol execution is only a part. Recall that the input of $\mathcal{P}$ consists of the (early) global output of the current execution (plus some additional auxiliary information). This models the "legal" information that the larger interaction gets from the current execution. In particular, note that $\mathcal{P}$ is *not* given the random inputs of the parties.

The interaction with $\mathcal{P}$ "forces" the ideal-process adversary $\mathcal{S}$ to generate its output in a somewhat interactive manner. That is, first $\mathcal{S}$ has to generate its output. Next, on each corruption request, $\mathcal{S}$ has to output an additional piece of data (presumably, this piece of data simulates the internal data of the relevant party). This interactivity will be essential in the proof of the composition theorem.

Since our composition theorems are restricted to the case where at most one subroutine call is being performed at each round, it is sufficient to activate $\mathcal{P}$ only *after* the interaction is completed. A natural generalization of the definition, dealing with the case where several subroutine calls are running concurrently, would allow $\mathcal{P}$ to interact with the adversary (both in real-life and in the ideal process) also during the run of the protocol. This generalization is addressed in [8].

Note that Definition 10 lets the ideal-process adversary $\mathcal{S}$ depend on the post-protocol corruptor $\mathcal{P}$. As noted in Remark 4 (Section 3.1.1), this apparent relaxation is of no technical consequence. Yet, it may help prove security of protocols.

Finally, we remark that a notion of post-protocol corruption appears also in [2]; yet there the formalization is somewhat different. In particular, the [2] there is no *algorithmic* post-protocol corruptor; instead the security requirement is stated for "all sequences of post-protocol corruptions". This formalization does not allow the corruptions to be chosen adaptively; in addition it is not readily adaptable to the computational setting (see Section 5).[23]

4.2 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 semi-ideal model; next we describe how an ideal evaluation call is replaced by a subroutine protocol. Finally we state the composition theorems.

The semi-ideal model. The (adaptive) semi-ideal model with ideal access to $f_1, ..., f_m$ (or in short the $(f_1, ..., f_m)$-ideal model), is defined analogously to the non-adaptive case. 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_j (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_j-inputs to T (party P_i hands x_i^f), and are handed back their respective outputs (P_i learns

[23] In [2] the term "postprotocol corruptor" refers to a different entity than the one here. There it is a module within the adversary that receives external corruption requests and corrupts parties accordingly. Here this module is an integral part of the adversary.

$f_j(x_1^f..x_n^f, r_{f_j})_i)$. As in the ideal process, the adversary can adaptively corrupt parties immediately before the call to T, 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 the adversary is ready, the trusted party is invoked and hands the outputs to the parties. Next the adversary can again adaptively corrupt parties as usual.

Let $\text{EXEC}^{f_1,...,f_m}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z})$ denote the random variable describing the global output of the computation in the $f_1, ..., f_m$-ideal model with protocol π, adversary $\mathcal{A}$, post-protocol corruptor $\mathcal{P}$, inputs $\vec{x}$ and auxiliary input $\vec{z}$ for the parties, analogously to the definition of $\text{EXEC}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z})$ in Section 4.1. (We stress that here π is not a real-life protocol and uses ideal calls to T.) Protocols for securely computing a function g in the $(f_1, ..., f_m)$-ideal model are defined in the usual way:

Definition 11 *Let $f_1, ..., f_m, g$ be n-party functions and let π be a protocol for n parties in the $(f_1, ..., f_m)$-ideal model. We say that π* adaptively t-securely computes g in the $(f_1, ..., f_m)$-ideal model *if for any adaptive t-limited adversary $\mathcal{A}$ (in the f-ideal model) and any post-protocol 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 for every input vector $\vec{x}$, and every auxiliary input vector $\vec{z}$,*

$$\text{IDEAL}_{g,\mathcal{S},\mathcal{P}}(\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{EXEC}^{f_1,...,f_m}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z}). \tag{8}$$

If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries then we say that π adaptively t-privately computes g in the $(f_1, ..., f_m)$-ideal 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 3.2), 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 12 (adaptive modular composition for active adversaries) *Let $f_1, ..., f_m, g$ be n-party functions. Let π be an n-party protocol that adaptively t-securely computes g in the $(f_1, ..., f_m)$-ideal 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 compute $f_1, ..., f_m$, respectively. Then the protocol $\pi^{\rho_1,...,\rho_m}$ adaptively t-securely computes g.*

The proof appears in Section 4.3 below. A considerably simpler proof applies to the case of *passive* adversaries. We leave the proof as an easy exercise:

Theorem 13 (Adaptive modular composition for passive adversaries) *Let $f_1, ..., f_m, g$ be n-party functions. Let π be an n-party protocol that adaptively t-privately computes g in the $(f_1, ..., f_m)$-ideal 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-privately computes $f_1, ..., f_m$, respectively. Then the protocol $\pi^{\rho_1,...,\rho_m}$ adaptively t-privately computes g.*

4.3 Proof of Theorem 12

As in the non-adaptive case, we restrict the presentation to protocols 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 straightforward (as long as at most one call to T is made at each round).

The proof outline is the same as in the non-adaptive case (Section 3.3) and is not repeated. Yet here the proof is considerably more involved. We sketch the additional difficulties arising from the adaptivity of the adversaries and simulators. Full details appear in Section 4.3.1 below.

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-protocol 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. That is, 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. In addition, $\mathcal{P}_\rho$ gets sufficient information to continue the run of $\mathcal{A}$ from this point 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 semi-ideal 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 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 semi-ideal 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-protocol corruptor $\mathcal{P}$ 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 P's (simulated) internal data from ρ 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 $\mathcal{P}$ for $\mathcal{S}_\rho$' is a 'good' one. That is, the interaction between $\mathcal{S}_\rho$ and $\mathcal{A}_\pi$ is identical to an (imaginary) interaction between $\mathcal{S}_\rho$ and $\mathcal{P}_\rho$.

4.3.1 A Detailed proof

We flesh out the details of the proof. Let $\mathcal{A}$ be an adversary (interacting with parties running π^ρ), and let $\mathcal{P}$ be a post-protocol corruptor. First we present the constructions of $\mathcal{A}_\rho$, $\mathcal{P}_\rho$, $\mathcal{A}_\pi$ and $\mathcal{S}$, proving two claims along the way. (As in the non-adaptive case, these claims are not technically necessary for the analysis of $\mathcal{S}$ and the proof; yet they may assist in understanding the construction.) Next we show that $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{IDEAL}_{g,\mathcal{S},\mathcal{P}}(\vec{x},\vec{z})$.

Terminology. We use the same definitions of executions, transcripts, adversary views, valid views, valid prefixes, and running an adversary on its view, as in the non-adaptive case (Section 3.3.1). Here of course these notions refer to the adaptive model. In particular, here the transcript contains the view of the post-protocol corruptor (i.e., its auxiliary and random inputs, and its interaction with the adversary.)

Let $\text{TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},z,\vec{r})$ denote the transcript of an execution of protocol π^ρ in the real-life model with adversary $\mathcal{A}$, post-protocol corruptor $\mathcal{P}$, inputs $\vec{x}$, random inputs $\vec{r}$ and auxiliary input z. Let $\text{TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},z,\vec{r})$ be similarly defined with respect to protocol π and adversary $\mathcal{A}_\pi$ in the f-ideal model.

Construction of $\mathcal{A}_\rho$. Given adversary $\mathcal{A}$, adversary $\mathcal{A}_\rho$ proceeds as follows. Let l 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 prefix of $\mathcal{A}$'s view, ending at round $l-1$. Verifying that $\vec{z}_0^\rho$ is a valid prefix is done by running $\mathcal{A}$ on z_0^ρ up to this round. If $\vec{z}_0^\rho$ is not a valid prefix then $\mathcal{A}_\rho$ halts.

2. $\mathcal{A}_\rho$ corrupts, in its real-life environment, the parties that are already corrupted in the run described in $\vec{z}_0^\rho$. ($\mathcal{A}_\rho$ ignores the input values of these parties.)

3. $\mathcal{A}_\rho$ continues the above run of $\mathcal{A}$ from round l on, follows $\mathcal{A}$'s instructions, and hands the learned information to $\mathcal{A}$. Following $\mathcal{A}$'s instruction consists of:

   (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.

   (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 data for protocol ρ. In addition, $\mathcal{A}_\rho$ needs to provide $\mathcal{A}$ with P_i's internal data from the execution of protocol π. This data 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 data 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 data of protocols π and ρ are sufficient for obtaining P_i's internal data of π^ρ. $\mathcal{A}_\rho$ now hands this data to $\mathcal{A}$.

4. Once protocol ρ is completed, $\mathcal{A}_\rho$ outputs its view of the computation. Next, $\mathcal{A}_\rho$ is ready to interact with some post-protocol corruptor $\mathcal{P}$, as described in Section 4.1.

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. Recall that $\mathcal{P}_\rho$ starts off with an auxiliary input, z_P^ρ, random input r_P^ρ, and with the (early) global output of some execution of protocol ρ with adversary $\mathcal{A}_\rho$. Corruptor $\mathcal{P}_\rho$ expects z_P^ρ to describe an input vector $\vec{x}$, an auxiliary input vector $\vec{z}$, and a random input vector $\vec{r}$ for protocol π^ρ. ($\vec{r}$ will be used only until round $l-1$.)

2. $\mathcal{P}_\rho$ "orchestrates" an execution, *until round* $l-1$, of parties running π^ρ with adversary $\mathcal{A}$ on input x_i, auxiliary input z_i and random input r_i for P_i (z_0 and r_0 for $\mathcal{A}$).

3. Next $\mathcal{P}_\rho$ lets v denote the part of the early global output that describes $\mathcal{A}_\rho$'s view of the execution. $\mathcal{P}_\rho$ uses v to continue the above run of $\mathcal{A}$ until the completion of ρ. If v is inconsistent with the current run of $\mathcal{A}$ (that is, if $\mathcal{A}$ ever responds differently from the response described in v) then $\mathcal{P}_\rho$ halts.

4. Once protocol ρ is completed (and the parties return to running π), corruptor $\mathcal{P}_\rho$ continues the "orchestration" of the current execution of $\mathcal{A}$ with parties running π^ρ. In order to continue running π, the uncorrupted parties need their outputs from protocol ρ. These outputs are taken from the early global view. In addition, $\mathcal{P}_\rho$ uses its random input r_P^ρ as the random inputs of the uncorrupted parties during this part of the execution.

5. Whenever $\mathcal{A}$ 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 data, d_i, corruptor $\mathcal{P}_\rho$ combines d_i with P_i's internal data for π, obtains P_i's internal data for π^ρ, and hands this data to $\mathcal{A}$.

6. When π^ρ is completed and $\mathcal{A}$ is ready to interact with its own post-protocol 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.

7. $\mathcal{P}_\rho$ halts when $\mathcal{A}$ does.

It now follows from the security of protocol ρ that there exists an ideal-process adversary $\mathcal{S}_\rho$ such that for all inputs $\vec{x}$ and all auxiliary input $\vec{z}$, we have $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(\vec{x},\vec{z})$.

Note that $\mathcal{A}_\rho$ is deterministic, since all of the randomness used by $\mathcal{A}$ is provided in the auxiliary input z_0^ρ. Yet, the simulator $\mathcal{S}_\rho$ is (inherently) randomized, since it should generate a distribution equal to $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}}(\vec{x},\vec{z})$. (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.[24] Note that $\mathcal{A}_\rho$ completely ignores the internal data received from the parties that were already corrupted in the prefix of view described in z_0^ρ. (Call these the **early corrupted** parties.) Consequently, the simulator $\mathcal{S}_\rho$ 'does not care about' the input values for these parties. That is, given some output v of $\mathcal{A}_\rho$, let the corresponding **reduced output** v' of $\mathcal{A}_\rho$ be the string string obtained from v by deleting the internal data of the early corrupted parties in the execution described in v. Let the reduced output of simulator $\mathcal{S}_\rho$ be defined in the same way. Informally, it now follows from the definition of $\mathcal{A}_\rho$ that the distribution of the reduced output of $\mathcal{S}_\rho$ remains unchanged regardless of the input values of the early corrupted parties. In particular, the distribution of the reduced output of $\mathcal{S}_\rho$ remains unchanged if we set the input values of the early corrupted parties to 0.

We formalize this discussion as follows. Let $\text{ADV}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho,\vec{z}^\rho)$ denote the output distribution of $\mathcal{S}_\rho$ on inputs $\vec{x}^\rho = x_1^\rho,...,x_n^\rho$, post-protocol corruptor $\mathcal{P}_\rho$, and auxiliary inputs $\vec{z}^\rho = z_P^\rho, z_0^\rho,...,z_n^\rho$. Let $\text{RED-ADV}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho,\vec{z}^\rho)$ denote the distribution of the *reduced* output of $\mathcal{S}_\rho$ on $\vec{x}, \mathcal{P}_\rho, \vec{z}$. Let $\vec{x}^\rho|_0$ denote the vector obtained from $\vec{x}^\rho$ by replacing all the inputs of the early corrupted parties with 0. (Recall that the identities of the early corrupted parties can be deduced from z_0^ρ.) Then,

Claim 14 *For any input vector $\vec{x}^\rho$, and any auxiliary input vector $\vec{z}^\rho$ we have:*

$$\text{RED-ADV}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho,\vec{z}^\rho) \stackrel{\text{d}}{=} \text{RED-ADV}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho|_0,\vec{z}^\rho)$$

Proof: If z_0^ρ, the auxiliary input of $\mathcal{S}_\rho$, is not a valid of $\mathcal{A}$'s view then $\mathcal{A}_\rho$ halts with no output and the claim holds. Otherwise, notice that $\mathcal{A}_\rho$ is instructed to ignore the inputs of the early corrupted parties. (Instead, $\mathcal{A}_\rho$ continues following $\mathcal{A}$'s instructions regarding those parties.) Thus, in any execution of $\mathcal{A}_\rho$ the reduced output does not depend on the inputs of the early corrupted parties in any way. Thus,

$$\text{RED-ADV}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho,\vec{z}^\rho) \stackrel{\text{d}}{=} \text{RED-ADV}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho|_0,\vec{z}^\rho),$$

where $\text{RED-ADV}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho,\vec{z}^\rho)$ denotes the distribution of the reduced output of $\mathcal{A}_\rho$ interacting with parties running ρ on inputs $\vec{x}^\rho$, post-protocol corruptor $\mathcal{P}_\rho$, and auxiliary inputs $\vec{z}^\rho$. However,

[24]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 $\mathcal{A}_\pi$.

$\text{RED-ADV}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho,\vec{z}^\rho) \stackrel{\rm d}{=} \text{RED-ADV}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho,\vec{z}^\rho)$, and $\text{RED-ADV}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho|_0,\vec{z}^\rho) \stackrel{\rm d}{=} \text{RED-ADV}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho|_0,\vec{z}^\rho)$. 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, $\mathcal{A}$ 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 prefix of $\mathcal{A}$'s view of the current run up to round $l-1$. 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 early corrupted parties, thus their inputs do not affect the distribution of the reduced output of $\mathcal{S}_\rho$. See Claim 14.)
   (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-ideal 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 data 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. Once $\mathcal{S}_\rho$ has produced its output (before interacting with a post-protocol corruptor), $\mathcal{A}_\pi$ removes from this output the internal data of the early corrupted parties, as well as the auxiliary input z_0^ρ. Next $\mathcal{A}_\pi$ adds, in the appropriate places, the internal data pertaining to protocol π of the parties corrupted during the execution of ρ. (This data is obtained from the suspended current interaction of $\mathcal{A}_\pi$ in the f-ideal model.) Call this the **modified output** of $\mathcal{S}_\rho$.
4. $\mathcal{A}_\pi$ uses the modified output of $\mathcal{S}_\rho$, denoted v', to continue the current run of $\mathcal{A}$ from round l until the completion of protocol ρ. If v' is not consistent with $\mathcal{A}$ (that is, if at any stage $\mathcal{A}$ generates a different value from the corresponding value described in v) then $\mathcal{A}_\pi$ halts. (See the discussion and Claim 15 below.)
5. Once $\mathcal{A}$ has completed the simulated part of its interaction and the parties have returned to running π, adversary $\mathcal{A}_\pi$ returns to following $\mathcal{A}$'s instructions until the completion of protocol π. 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-ideal model and obtains P's internal data pertaining to protocol π.
   (b) $\mathcal{A}_\pi$ plays the role of the post-protocol corruptor for $\mathcal{S}_\rho$, and requests corruption of P. It then obtains the (simulated) internal data of P pertaining to protocol ρ. (In the process $\mathcal{S}_\rho$ will probably corrupt P in its ideal model. 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 data of P pertaining to protocol π^ρ, and hands this data to $\mathcal{A}$.[25]

6. Once protocol π is completed, $\mathcal{A}_\pi$ outputs its entire view of the interaction, and is ready to interact with a post-protocol corruptor.

We are now guaranteed that there exists a simulator $\mathcal{S}_\pi$ such that for every input $\vec{x}$, and every auxiliary input $\vec{z}$ we have $\text{IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z})$. (Here we use the post-protocol corruptor $\mathcal{P}$ that is given together with adversary $\mathcal{A}$.)

Notice that, in the construction of $\mathcal{A}_\pi$, the output of $\mathcal{S}_\rho$ is a valid continuation of the current run of $\mathcal{A}$. This is formalized as follows. Consider an execution of $\mathcal{A}_\pi$, and let w denote $\mathcal{A}_\pi$'s view of this execution. Let v' denote the modified output of $\mathcal{S}_\rho$ within this execution. Next, take w and replace the part describing $\mathcal{A}_\pi$'s view of round l with v'. Furthermore, for each party P, corrupted after protocol ρ is completed, add P's simulated internal data from protocol ρ (obtained from the response of $\mathcal{S}_\rho$ to $\mathcal{A}_\pi$'s request to corrupt P) to the internal data of P. Finally delete r^ρ, the random bits used by $\mathcal{A}_\pi$ to run $\mathcal{S}_\rho$, from w. Call the result the **modified view of $\mathcal{A}_\pi$**. The following claim is implied by Claim 18 below; we thus omit the proof here.

Claim 15 *Let w' be the modified view of $\mathcal{A}_\pi$ in some execution of $\mathcal{A}_\pi$. Then w' is a valid view of $\mathcal{A}$ up to the conclusion of protocol π.*

Construction of $\mathcal{S}$. Informally, $\mathcal{S}$ 'reproduces' a simulated transcript of the above interaction between $\mathcal{A}_\pi$ and $\mathcal{A}$, by combining the outputs of $\mathcal{S}_\rho$ and $\mathcal{S}_\pi$. This is done as follows, given simulators $\mathcal{S}_\pi$ and $\mathcal{S}_\rho$.

1. $\mathcal{S}$ first runs $\mathcal{S}_\pi$ on its ($\mathcal{S}$'s) own auxiliary input and random input. Whenever $\mathcal{S}_\pi$ asks to corrupt a party, $\mathcal{S}$ corrupts this party (in the ideal process) and hands the received input and auxiliary input back to $\mathcal{S}_\pi$.

2. When $\mathcal{S}_\pi$ hands its trusted party the inputs of the corrupted parties and asks for the function (ie, g) values, $\mathcal{S}$ acts accordingly: it invokes its own trusted party with the same input values for the corrupted parties, and hands the function values back to $\mathcal{S}_\pi$. If a party is corrupted after the function values are known then $\mathcal{S}_\pi$ is given also the party's output value.

3. When $\mathcal{S}_\pi$ generates its output (before the interaction with the post-protocol corruptor), $\mathcal{S}$ records this output in a variable w. Recall that w is a valid view of an execution of $\mathcal{A}_\pi$.

4. Next $\mathcal{S}$ runs simulator $\mathcal{S}_\rho$, in the same way that $\mathcal{A}_\pi$ would with view w. That is:

   (a) $\mathcal{S}_\rho$'s auxiliary input, z_0^ρ, is taken from the part of w describing the prefix of $\mathcal{A}$'s view up to (and excluding) round l.

   (b) $\mathcal{S}_\rho$'s random input is taken from the part of w describing the random input r^ρ that $\mathcal{A}_\pi$ uses to run $\mathcal{S}_\rho$.

   (c) Dealing with requests of $\mathcal{S}_\rho$ to corrupt parties is done as follows. First notice that all the parties that $\mathcal{S}_\rho$ corrupts already appear as corrupted in w. This follows from the fact that that the current execution of $\mathcal{S}_\rho$ is identical to the execution of $\mathcal{S}_\rho$ that is implicit in w. Thus:

[25] Informally, the interaction between $\mathcal{A}_\pi$ and $\mathcal{S}_\rho$ during Step 4 is "the same as" the interaction between $\mathcal{P}_\rho$ and $\mathcal{S}_\rho$, given the appropriate auxiliary input for $\mathcal{P}_\rho$. This claim is made more precise in the sequel.

i. When $\mathcal{S}_\rho$ corrupts a party which was corrupted before round l, then $\mathcal{S}$ sets the input of this party to 0. (Recall that these are the early corrupted parties.) The auxiliary inputs of these parties are taken from w. (Each auxiliary input is set to the internal data of the said party from the execution of π up to round l.)

ii. When $\mathcal{S}_\rho$ corrupts a party which was corrupted (in w) during round l, then $\mathcal{S}$ takes both the input and the auxiliary input of this party from w, and hands this data to $\mathcal{S}_\rho$.

(d) When $\mathcal{S}_\rho$ hands the inputs of the corrupted parties to its trusted party and asks for the function (ie, f) values, $\mathcal{S}$ looks up w to find the interaction between the simulated $\mathcal{A}_\pi$ and the trusted party T at round l of the computation. We are guaranteed that in that interaction $\mathcal{A}_\pi$ hands T the same input values that $\mathcal{S}_\rho$ hands its trusted party. Now $\mathcal{S}$ finds in w the values that T hands $\mathcal{A}_\pi$ and hands those values to $\mathcal{S}_\rho$ as the values given by the trusted party.

5. When $\mathcal{S}_\rho$ generates its output (before the interaction with the post-protocol corruptor), $\mathcal{S}$ records this output in variable v. Next $\mathcal{S}$ computes the corresponding *modified output*, v', in the same way that $\mathcal{A}_\pi$ would do on view w. (See Step 3 in the description of $\mathcal{A}_\pi$.)

6. Next, $\mathcal{S}$ obtains from $\mathcal{S}_\rho$ the internal data pertaining to protocol ρ of the parties that were corrupted after ρ is completed, in the same way that $\mathcal{A}_\pi$ would on view w. That is, whenever w describes a corruption of some party P after ρ is completed, $\mathcal{S}$ plays the post-protocol corruptor for $\mathcal{S}_\rho$, asks $\mathcal{S}_\rho$ to corrupt P and records the answer.

7. $\mathcal{S}$ combines w and v' in order to obtain a full view of $\mathcal{A}$. That is, first $\mathcal{S}$ removes from w the part of $\mathcal{A}_\pi$'s random input that is used to run $\mathcal{S}_\rho$. Next $\mathcal{S}$ removes from w the part that describes round l, and replaces it with v'. Finally, $\mathcal{S}$ incorporates the internal data of the post-protocol corrupted parties (as reported by $\mathcal{S}_\rho$ in the previous step) within w, in the natural way. Let y denote the obtained string.

8. It now follows from Claim 15 and the description of $\mathcal{A}_\pi$ that y is a valid view of $\mathcal{A}$. Next, $\mathcal{S}$ runs $\mathcal{A}$ on its view y, and outputs whatever $\mathcal{A}$ outputs. (If $\mathcal{A}$'s output is its entire view then this step is not necessary and $\mathcal{S}$ outputs y.)

9. Now $\mathcal{S}$ is ready to interact with post-protocol corruptor $\mathcal{P}$. Whenever the corruptor asks to corrupt some party P, then $\mathcal{S}$ plays the post-protocol corruptor for both $\mathcal{S}_\pi$ and $\mathcal{S}_\rho$, asking each of them to corrupt P. Now, $\mathcal{S}_\pi$ returns P's internal data pertaining to protocol π, and $\mathcal{S}_\rho$ returns P's internal data pertaining to protocol ρ. Simulator $\mathcal{S}$ combines the two parts of P's data and obtains P's internal data for protocol π^ρ.

Analysis of $\mathcal{S}$. The analysis follows the general outline of the analysis for the non-adaptive case, with some necessary local additions. Fix an input vector $\vec{x}$, a post-protocol corruptor $\mathcal{P}$, and auxiliary input vector $\vec{z}$, as well as some value of the security parameter. We show that $\text{IDEAL}_{g,\mathcal{S},\mathcal{P}}(\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z})$. (For definitions of these random variables see Section 4.1. In addition recall the definition of $\text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z})$ in Section 4.2.) We first define the following two 'hybrid' distributions.

- Let $\text{MOD-EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z})$ be obtained from the random variable $\text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z})$, by replacing the output of $\mathcal{A}_\pi$ (i.e., its view) with its *modified* view, as defined in the preamble to Claim 15. The outputs of the uncorrupted parties and $\mathcal{P}$'s view are not modified.

- Let $\text{MOD-IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(\vec{x},\vec{z})$ denote the random variable obtained by applying the same procedure to $\text{IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(\vec{x},\vec{z})$ (with respect to the output of $\mathcal{S}_\pi$).

We complete the proof via the following three claims.

Claim 16 $\text{MOD-IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{IDEAL}_{g,\mathcal{S},\mathcal{P}}(\vec{x},\vec{z})$.

Proof: This claim is immediate from the definition of $\mathcal{S}$. □

Claim 17 $\text{MOD-EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{MOD-IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(\vec{x},\vec{z})$.

Proof: Recall that $\text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(\vec{x},\vec{z})$. The modified variables $\text{MOD-EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z})$ and $\text{MOD-IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(\vec{x},\vec{z})$ are obtained from the unmodified ones by applying the same (deterministic) procedure. □

Claim 18 $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{MOD-EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z})$.

Proof:

The proof outline is quite similar to the non-adaptive case. We start by defining the following notions.

- Recall the definition of the transcript $\text{TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})$. Let the **reduced transcript of** π^ρ, denoted $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})$, be constructed from $\text{TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})$ by removing all the information pertaining to the execution of protocol ρ, except for the inputs and outputs of the parties and the information that appears in $\mathcal{A}$'s view of this execution. That is, first remove from $\text{TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})$ the internal data of the *un*corrupted parties pertaining to protocol ρ. Then remove the messages sent among the uncorrupted parties during the execution of ρ. The local inputs and outputs of the uncorrupted parties from protocol ρ are *not* removed.
- Recall the definition of $\text{TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r})$. Let the **augmented transcript of** π, denoted $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r})$, be constructed from $\text{TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r})$ by replacing $\mathcal{A}_\pi$'s view with $\mathcal{A}_\pi$'s *modified* view (as defined in the preamble to Claim 15). In particular, this means that if party P is corrupted during or after round l, then the internal data of P is modified to include the information pertaining to protocol ρ (obtained from the output of $\mathcal{S}_\rho$ within $\mathcal{A}_\pi$).

In all the above constructs, if the random input $\vec{r}$ is not specified, then the notation describes the random variable obtained by choosing $\vec{r}$ at random.

Note that, for each $\vec{r}$, the value $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})$ is obtained from $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})$ via a deterministic procedure. (This procedure extracts the outputs of the uncorrupted parties and the adversary view from $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})$.) Furthermore, $\text{MOD-EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r})$ is obtained from $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r})$ by applying the *same deterministic procedure.* Consequently, it suffices to show that

$$\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z}) \stackrel{\text{d}}{=} \text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z}). \tag{9}$$

To show (9), define the following random variables. The first variables, denoted $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})^{(1)}$ and $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r})^{(1)}$, are the prefixes of the transcripts of $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})$ and $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r})$, respectively, until round $l-1$.

The second variables, denoted $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})^{(2)}$ (resp., $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r})^{(2)}$), are obtained in two steps: First remove from $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})$ (resp., from $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r})$) all the information pertaining to the interaction from the point where protocol ρ completes its execution (resp., from round $l+1$) and on. Next put back in the internal data pertaining to protocol ρ (resp., the simulated internal data pertaining to ρ) of the parties corrupted after ρ is completed. Finally add the parts of the random inputs of the parties that are used in protocol π.[26]

We complete the proof of the Claim in three steps:

Step 1. Let $\vec{r}^{(1)}$ denote the part of the random input $\vec{r}$ that is used until (and excluding) round l. Note that, although the random inputs to the two executions are of different lengths, the part pertaining to the executions until round l are of equal length. Furthermore, for each value of $\vec{r}^{(1)}$ we have $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r}^{(1)})^{(1)} = \text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r}^{(1)})^{(1)}$. This holds since until round l protocols π^ρ and π are identical, and adversary $\mathcal{A}_\pi$ follows $\mathcal{A}$'s instructions.

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 more care.

Fix a value for $\vec{r}^{(1)}$. Let E be the resulting set of early corrupted parties (i.e., E is the set of parties that are corrupted by the onset of round l.) The set E 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 $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r}^{(1)})^{(1)}$. (The inputs of the parties in E are set to 0.) Let z_i^ρ denote the internal data of party P_i at the onset of round l. (We assume that z_i^ρ includes z_i, where z_i is P_i's auxiliary input to π^ρ.) Let z_0^ρ denote $\mathcal{A}$'s view of the interaction up to round l. Let z_P^ρ, the auxiliary input of $\mathcal{P}_\rho$, be set to $z_P = \vec{x},\vec{z},\vec{r}^{(1)}$. 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 $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r}^{(1)})^{(1)}$. (The inputs of the parties in E are set to 0.) Let z_i^f denote the internal data of party P_i at the onset of round l in this interaction. (We assume that z_i^f includes z_i, where z_i is P_i's auxiliary input to π.) Let z_0^f denote $\mathcal{A}$'s view of the interaction up to round l. Set z_P^f, the auxiliary input of corruptor $\mathcal{P}_\rho$, is set to $z_P^f = \vec{x},\vec{z},\vec{r}^{(1)}$. Let $\vec{z}^f = z_P^f, z_0^f, ..., z_n^f$.

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

$$\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho,\vec{z}^\rho) \stackrel{\text{d}}{=} \text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(\vec{x}^f,\vec{z}^f) \tag{10}$$

It remains to observe that:

- The random variable $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r})^{(2)}$ is obtained by merging, via a simple 'syntactic' procedure, the fixed value $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z},\vec{r}^{(1)})^{(1)}$ with the random variable

[26] In fact, it suffices to add only the random inputs to be used during the part of π that is executed after ρ has returned. Furthermore, only the random inputs of the parties that are uncorrupted at this stage are needed. Yet, for simplicity we include all the random inputs used in π.

$\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho, \vec{z}^\rho)$. (This procedure concatenates the two values, and then performs some obvious rearrangements and deletions of data that appears multiple times, such as z_0^ρ and z_P. In particular, the random inputs of the uncorrupted parties for the part of π that is executed after ρ is completed are obtained from the random input of $\mathcal{P}_\rho$.) [27]

- The random variable $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r})^{(2)}$ is obtained by merging the fixed value $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z},\vec{r}^{(1)})^{(1)}$ with the random variable $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(\vec{x}^f,\vec{z}^f)$, via *the same procedure* as in the previous bullet. (Here we notice that the interaction between $\mathcal{P}_\rho$ and $\mathcal{S}_\rho$ is distributed identically to the interaction between $\mathcal{A}_\pi$ and $\mathcal{S}_\rho$.)

It follows that

$$\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z})^{(2)} \stackrel{\text{d}}{=} \text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z})^{(2)}. \tag{11}$$

Step 3. The following holds.

- The random variable $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z})$ is obtained from $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z})^{(2)}$ by applying a deterministic procedure. This procedure consists of continuing the current execution of π^ρ with adversary $\mathcal{A}$, from the completion of protocol ρ onwards. (Note that all the parameters for uniquely specifying this part of the execution are determined in $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z})^{(2)}$. In particular, the random inputs of the uncorrupted parties for the current part of π are taken from $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z})^{(2)}$. Also, whenever a party P is corrupted, the internal data of P pertaining to ρ is taken from $\text{RED-TRANS}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z})^{(2)}$.)

- Recall that from the resumption of protocol π until its conclusion, adversary $\mathcal{A}_\pi$ returns to following the instructions of $\mathcal{A}$, with the exception that whenever $\mathcal{A}$ corrupts a party, P, then $\mathcal{A}_\pi$ requests $\mathcal{S}_\rho$ to corrupt P and incorporates $\mathcal{S}_\rho$'s answer in P's internal data before feeding this data to $\mathcal{A}$. Consequently, the random variable $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z})$ is obtained from $\text{AUG-TRANS}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z})^{(2)}$ by applying the *same deterministic procedure* of the previous bullet.

Equation (9) follows. $\square$

5 The computational setting

In this section we define secure protocols, present 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.

The treatment is quite similar to the case of adaptive adversaries in the secure channels setting (Section 4). Therefore, this section is *not* self-contained; we assume familiarity with Section 4 and only highlight the differences. Section 5.1 contains definitions of secure protocols. Section 5.2 presents and proves the composition theorem. All the remarks from Sections 3.1.1 and 4.1.1 are relevant here. Additional remarks on the definition in the computational setting appear at the end of Section 5.1.

[27] Here and in the next bullet we repeat the argument made in the proof of Claim 14, stating that the output of $\mathcal{A}_\rho$ (and, consequently, the output of $\mathcal{S}_\rho$) continues the execution of $\mathcal{A}$, regardless of the inputs of the parties in E.
In addition, we notice that $\mathcal{P}_\rho$ is defined so that it mimics the corruption requests made by $\mathcal{A}$ after the completion of ρ.

5.1 Adaptive security of protocols in the computational setting

We define adaptively secure multi-party computation in the computational setting. The definition is very similar to that of the secure channels setting (Section 4.1). Executing a protocol π in the real-life scenario, as well as the notation $\textsc{exec}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z})$, are the same as in the secure channels setting, with the following exceptions:

1. The real-life adversary, $\mathcal{A}$, and the post-protocol corruptor $\mathcal{P}$ are PPT.[28] See Remark 1 below.
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, with the exception that the ideal-model adversary $\mathcal{S}$ is PPT. The notation $\textsc{ideal}_{f,\mathcal{S},\mathcal{P}}(\vec{x},\vec{z})$ remains unchanged.

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

Definition 19 *Let $\mathcal{X} = \{X_\kappa\}_{\kappa\in\{0,1\}^*}$ and $\mathcal{Y} = \{Y_\kappa\}_{\kappa\in\{0,1\}^*}$ be two ensembles of probability distributions. We say that $\mathcal{X}$ and $\mathcal{Y}$ are* computationally indistinguishable *(and write "$\mathcal{X} \stackrel{c}{\approx} \mathcal{Y}$") if for every positive polynomial p, for every probabilistic polynomial-time algorithm D and for all κ where $k = |\kappa|$ is sufficiently large,*

$$|\mathrm{Prob}(D(X_\kappa) = 1) - \mathrm{Prob}(D(Y_\kappa) = 1)| < \frac{1}{p(k)}.$$

Definition 20 (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-protocol corruptor $\mathcal{P}$ there exists a PPT ideal-process adversary $\mathcal{S}$, such that for any input vector $\vec{x}$ and any auxiliary input vector $\vec{z}$ we have*

$$\textsc{ideal}_{f,\mathcal{S},\mathcal{P}}(\vec{x},\vec{z}) \stackrel{c}{\approx} \textsc{exec}_{\pi,\mathcal{A}}(\vec{x},\vec{z}) \tag{12}$$

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

Remark 1: On the complexity of $\mathcal{P}$. We stress that Definition 20 quantifies only over all post-protocol corruptors $\mathcal{P}$ that are PPT. A definition that allows $\mathcal{P}$ more computational power will be hard to satisfy, since an over-powerful $\mathcal{P}$ can break in an "illegal" way cryptographic primitives used by the parties. (This is so, since our model allows $\mathcal{P}$ access to the communication among the parties, via $\mathcal{A}$'s view. It can then encode "illegal information" in its corruption requests.)[30]

It is seen below that in the computational setting the composition theorems can be proven even when $\mathcal{P}$ is restricted to PPT.

[28] A Turing machine is PPT if it runs in probabilistic polynomial time. As in the secure channels case, security is measured in terms of the security parameter k.

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

[30] ¿From a purely technical point of view $\mathcal{P}$ cannot do "too much harm" since it outputs at most poly(n) bits, where n is a constant, whereas the security parameter k tends to infinity. Yet, this is an "unnatural" way to restrict the power of $\mathcal{P}$. In particular, in the closely related model where n grows with k an over-powerful $\mathcal{P}$ can make the definition practically unsatisfiable.

Remark 2: On computational correctness. It may appear at first glance that Definition 20 (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 protocol's output be *computationally indistinguishable* form the output of the ideal process.

This argument, however, is misleading. In particular, it is fuzzy about exactly how the function to be computed is specified. Let us consider two options: the first is that 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 (i.e., $g(x)$) on *any* input x. Thus outputting a random value, regardless of x, will not be considered secure. (Figuratively, in this case the distinguisher between the two sides of (12) 'knows x'.)

Another option is to let the parties have empty inputs, and require them 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.*

In other words, Definition 20 gives the protocol designer the *freedom* to choose whether to require absolute correctness (as in the first option) or computational correctness (as in the second option).

5.2 Modular composition: The computational case

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

The computational semi-ideal model. The (computational, adaptive) $(f_1, ..., f_m)$-ideal model is defined identically to the secure channels, adaptive case (Section 4.2), with the exception that we start from the computational real-life model, rather than from the secure-channels real-life model. The notation $\text{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z})$ remains unchanged (yet here is applies to the computational setting). Protocols for securely computing a function g in the computational $(f_1, ..., f_m)$-ideal model are defined in the usual way:

Definition 21 *Let $f_1, ..., f_m, g$ be n-party functions and let π be a protocol for n parties in the computational $(f_1, ..., f_m)$-ideal model. We say that π* adaptively t-securely computes g in the computational $(f_1, ..., f_m)$-ideal model *if for any* PPT *adaptive t-limited adversary $\mathcal{A}$ (in the f-ideal model) and every* PPT *post-protocol corruptor $\mathcal{P}$, there exists a* PPT *adaptive ideal-process adversary $\mathcal{S}$, and such that for every input vector $\vec{x}$ and every auxiliary input vector $\vec{z}$,*

$$\text{IDEAL}_{g,\mathcal{S},\mathcal{P}}(\vec{x},\vec{z}) \stackrel{c}{\approx} \text{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},\mathcal{P}}(\vec{x},\vec{z}). \tag{13}$$

If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries then we say that π adaptively t-privately computes g in the computationally $(f_1, ..., f_m)$-ideal 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 22 (adaptive, active modular composition in the computational setting) *Let $f_1, ..., f_m, g$ be n-party functions. Let π be an n-party protocol that adaptively t-securely computes g in the computational $(f_1, ..., f_m)$-ideal 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 compute $f_1, ..., f_m$, respectively, in the computational setting. Then protocol $\pi^{\rho_1, \ldots, \rho_m}$ adaptively t-securely computes g in the computational setting.*

The proof is very similar to the proof of Theorem 12. The differences appear below. The case of passive adversaries is proven similarly:

Theorem 23 (adaptive, passive modular composition in the computational setting) *Let $f_1, ..., f_m, g$ be n-party functions. Let π be an n-party protocol that adaptively t-privately computes g in the computational $(f_1, ..., f_m)$-ideal 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-privately computes $f_1, ..., f_m$, respectively, in the computational setting. Then protocol $\pi^{\rho_1, \ldots, \rho_m}$ adaptively t-privately computes g in the computational setting.*

Proof of Theorem 22: As in the case of secure channels, we restrict the presentation to protocols where $m = 1$ and the trusted party is called only once.

The constructions of $\mathcal{A}_\rho$, $\mathcal{P}_\rho$, $\mathcal{A}_\pi$ and $\mathcal{S}$ are identical to those of Section 4.3 (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 linear in the complexities of $\mathcal{A}$, $\mathcal{S}_\rho$ and $\mathcal{S}_\pi$.

The analysis of $\mathcal{S}$ is very similar to the secure channels case. The 'hybrid' random variables $\text{MOD-EXEC}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x}, \vec{z})$ and $\text{MOD-IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(\vec{x}, \vec{z})$ are defined as there. Claim 16 remains valid. Claims 17 and 18 are modified as follows.

Claim 24 $\text{MOD-EXEC}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x}, \vec{z}) \stackrel{c}{\approx} \text{MOD-IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(\vec{x}, \vec{z})$.

Proof: We have $\text{EXEC}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x}, \vec{z}) \stackrel{c}{\approx} \text{IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(\vec{x}, \vec{z})$. The modified variables $\text{MOD-EXEC}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x}, \vec{z})$ and $\text{MOD-IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(\vec{x}, \vec{z})$ are obtained from the unmodified ones by applying the same (deterministic) procedure. □

Claim 25 $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x}, \vec{z}) \stackrel{c}{\approx} \text{MOD-EXEC}^{f}_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x}, \vec{z})$.

Proof: The only difference from the proof of Claim 18 is in Step 2, namely that the two sides of Equation (10) are only guaranteed to be computationally indistinguishable (rather than equally distributed). Consequently, the two sides of Equation (11) are computationally indistinguishable and the claim follows. (Formally, given a distinguisher between the two sides of (11), one constructs a distinguisher between the two sides of (10), with the same success probability.) □

Furthermore, for any value of the security parameter 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-protocol corruptor $\mathcal{P}$ and a distinguisher D that distinguishes between $\text{EXEC}_{\pi^\rho,\mathcal{A},\mathcal{P}}(\vec{x}, \vec{z})$ and $\text{IDEAL}_{g,\mathcal{S},\mathcal{P}}(\vec{x}, \vec{z})$ with probability δ (for some values of $\vec{x}$ and $\vec{z}$), then there exist values δ_1 and δ_2 such that $\delta_1 + \delta_2 = \delta$, and distinguishers D_1 and D_2, such that:

- D_1 distinguishes between $\text{EXEC}^f_{\pi,\mathcal{A}_\pi,\mathcal{P}}(\vec{x},\vec{z})$ and $\text{IDEAL}_{g,\mathcal{S}_\pi,\mathcal{P}}(\vec{x},\vec{z})$ with probability δ_1.
- D_2 distinguishes between $\text{EXEC}_{\rho,\mathcal{A}_\rho,\mathcal{P}_\rho}(\vec{x}^\rho,\vec{z}^\rho)$ and $\text{IDEAL}_{f,\mathcal{S}_\rho,\mathcal{P}_\rho}(\vec{x}^f,\vec{z}^f)$ 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 Claim 18.)

□

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

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

[18] 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.

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

[20] O. Goldreich. *"Secure Multi-Party Computation",* In preparation, 1998.

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

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

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

[24] 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.

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

[26] E. Kushilevitz, "Privacy and communication complexity", SIAM Jour. disc. Math. Vol. 29th FOCS (1989).

[27] 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.

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

[29] 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.

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

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

A Extensions of the definitions

We have seen how the general framework of section 2 is adapted 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 four 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: the protocol is message-driven (i.e., upon the receipt of a message the party performs local computation, sends some messages, and then waits for the next received message). 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. Here 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. The ideal process is modified to reflect the fact we cannot prevent loss of messages. It only guarantees that any message received was indeed sent and wasn't modified or duplicated. In the case of key-exchange the ideal process reflects the specific requirement from this task. See details in [4].

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 introduced in [8].

To obtain on-line security the current real-life model and ideal process are modified essentially as follows. We replace the post-protocol corruptor with an arbitrary auxiliary information machine, $\mathcal{Z}$, that starts off with some auxiliary information. (Typically, this information will include the inputs of *all* parties.) Both the real-life adversary and the ideal-process adversary can interact with $\mathcal{Z}$ in an arbitrary way throughout the execution of the protocol. The view of $\mathcal{Z}$ is added to the global output of the computation; 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-protocol corruptor $\mathcal{P}$ is no longer necessary. However, the resulting security requirement is stronger.

Interactive inputs. The above settings deal 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 computations where the inputs are being learned as the computation proceeds (and may depend on previous computational steps). This extra generality is captured in a somewhat similar way to 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. 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 adversary view 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 $\mathcal{I}$ gives data also to 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 10) 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 will 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 [28], Goldwasser and Levin [22], Beaver [1, 2], and Canetti, Feige, Goldreich and Naor [7, 9]. These definitions vary in their level of restrictiveness. It seems that the most restrictive of these definitions is the [28] one.

Micali and Rogaway 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 are called input awareness and output awareness functions. 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 a simulated conversation between the (real-life) adversary and the 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 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.

These requirements seem to imply our definition; in fact, they seem to be considerably more restrictive. In particular, we don't require that the contributed inputs and outputs be determined based on the party's communications alone. (Our ideal-process adversary does imply input and output awareness functions for the corrupted parties, but these functions are allowed to depend on the entire adversary view.) Also, we do not restrict the type of simulation done by the ideal-process adversary (and in particular do not require black-box simulation).

We remark that the current version of the [28] definition applies only to the 'secure channels' setting where the adversary is computationally unbounded and does not learn the communication among uncorrupted parties. Also, this version deals only with protocols that evaluate a *deterministic* function.[32]

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.[33]

Beaver's definition [1, 2] can be regarded as the basis for the [9, 7] definitions and the ones here. We sketch the approach and 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 black-box and 'one-shot', in the sense that rewinding the adversary is not allowed. (Some parameters of the interaction between the interface

[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.

[32] The [28] manuscript contains many important and enlightening observations, arguments, and examples regarding secure multi-party protocols. We have benefited a lot from reading this work.

[33] 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.

and $\mathcal{A}$ seem to be left undefnined). 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 1 in Section 3.1.1). An additional weakness of this defintion is that, unlike here, $\mathcal{A}'$ is not required to be as efficient at $\mathcal{A}$. (See Remark 2 in Section 3.1.1).

In addition, to allow secure sequential composition, a secure protocol should be also postprotocol corruptible in a way that is somewhat similar to out notion of post-protocol corruptibility. Yet, the [2] notion does not define a post-protocol corruptor; instead the security requirement is made with respect to "any sequence of post-protocol corruption requests". This formalization has two main drawbacks. First, it does not take into account the fact that the post-protocol corruption requests can be adaptive and depend on the execution of the protocol itself (rather than being fixed in advance). Second, this formalization does not generalize to the computational setting.a (See Remark 2 in Section 4.1.1.)

The [7, 9] approach is very similar to the one here, except for the following differences. First, their definition requires the ideal-process adversary to operate via a restricted version of black-box simulation. Next, they do not incorporate auxiliary input in the definition, and do not include a post-protocol 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.