On adaptive vs. non-adaptive security of multiparty protocols

Ran Canetti* Ivan Damgaard† Stefan Dziembowski† Yuval Ishai‡
Tal Malkin§

February 27, 2001

Abstract

Security analysis of multiparty cryptographic protocols distinguishes between two types of adversarial settings: In the non-adaptive setting, the set of corrupted parties is chosen in advance, before the interaction begins. In the adaptive setting, the adversary chooses who to corrupt during the course of the computation. We study the relations between adaptive security (i.e., security in the adaptive setting) and non-adaptive security, according to two definitions and in several models of computation. While affirming some prevailing beliefs, we also obtain some unexpected results. Some highlights of our results are:

- According to the definition of Dodis-Micali-Rogaway (which is set in the information-theoretic model), adaptive and non-adaptive security are equivalent. This holds for both honest-but-curious and Byzantine adversaries, and for any number of parties.
- According to the definition of Canetti, for honest-but-curious adversaries, adaptive security is equivalent to non-adaptive security when the number of parties is logarithmic, and is strictly stronger than non-adaptive security when the number of parties is super-logarithmic. For Byzantine adversaries, adaptive security is strictly stronger than non-adaptive security, for any number of parties.

*IBM Watson.

†Aarhus University, BRICS (Basic Research in Computer Science, Center of the Danish National Research Foundation).

‡DIMACS and AT&T. Work partially done while the author was at the Technion and while visiting IBM Watson.

§AT&T Labs – Research. Work partially done while the author was at MIT and while visiting IBM Watson.

Contents

1 Introduction

Security analysis of cryptographic protocols is a delicate task. A first and crucial step towards meaninful analysis is coming up with an appropriate definition of security of the protocol problem at hand. Formulating good definitions is non-trivial: They should be compehensive and stringent enough to guarantee security against a variety of threats and adversarial behaviors. On the other hand, they should be as simple, workable, and as permissive as possible, so as to facilitate design and analysis of secure protocols, and to avoid unnessecary requirements.

Indeed, in contrast with the great advances in constructing cryptographic protocols for a large variety of protocol problems, formalizing definitions of security for crypographic protocol problems has been progressing more slowly. The first protocols appearing in the literature use only intuitive and ad-hoc notions of security, and rigorous security analysis was virtually non-existent. Eventually, several general definitions of security for cryptographic protocols have appeared in the literature. Most notable are the works of Goldwasser and Levin [GL90], Micali and Rogaway [MR91], Beaver [B91], Canetti [C00] and Dodis and Micali [DM00] (that concentrate on the task of *secure function evaluation* [Y82, Y86, GMW87]), and Pfitzmann and Waidner [PW94], Pfitzmann Schunter and Waidner [PSW00], and Canetti [C00a] (that discuss general reactive tasks). In particular, only recently do we have precise and detailed definitions that allow rigorous study of “folklore beliefs” regarding secure protocols.

This work initiates a comparative study of notions of security, according to different definitions. We concentrate on secure function evaluation, and in particular the following aspect. Adversarial behavior of a computational environment is usually modelled via a single algorithmic entity, the adversary, the capabilities of which represent the actual security threats. Specifically, in a network of communicating parties the adversary is typically allowed to control (or, corrupt) some of the parties. Here the following question arises: How are the corrupted parties chosen? One standard model assumes that the set of corrupted parties is fixed before the computation starts. This is the model of non-adaptive adversaries. Alternatively, the adversary may be allowed to corrupt parties during the course of the computation, when the identity of each corrupted party may be based on the information gathered so far. We call such adversaries adaptive.

Indeed, attackers in a computer network (hackers, viruses, insiders) may break into computers during the course of the computation, based on partial information that was already gathered. Thus the adaptive model seems to better represent realistic security threats, and so provide a better security guarantee. However, defining and proving security of protocols is considerably easier in the non-adaptive model. One quintessential example for the additional complexity of guaranteeing adaptive security is the case of using encryption to transform protocols that assume ideally secure channels into protocols that withstand adversaries who hear all the communication. In the non-adaptive model standard Chosen-Ciphertext-Attack secure encryption [DDN91, CS98, S99] (or even plain semantically secure encryption [GM84], if used appropriately) is sufficient. To obtain adaptively secure encryption, it seems that one needs to either trust data erasures [BH92], or use considerably more complex constructs [CFGN96, B97, DN00].

Clearly, adaptive security implies non-adaptive security, under any reasonable definition of security. However, is adaptive security really a stronger notion than non-adaptive security? Some initial results (indicating clear separation in some settings) are provided in [CFGN96]. On the other hand, it is a folklore belief that in an “information theoretic setting” adaptive and non-adaptive security should be equivalent. Providing more complete answers to this question, in several models of computation, is the focus of this work. While some of our results affirm common beliefs, other results are quite surprising, and may considerably simplify the design and analysis of protocols.

Models of computation. We study the additional power of adaptive adversaries in a number of standard adversary models, and according to two definitions (the definition of Dodis, Micali, and Rogaway [MR91, DM00], and that of Canetti [C00]). To develop the necessary terminology for presenting our results let us very shortly outline the structure of definitions of security of protocols. (The description below applies to both definitions. The [MR91, DM00] definition imposes some additional requirements, sketched in a later section.)

As mentioned above, both definitions concentrate on the task of Secure Function Evaluation. Here the parties wish to jointly evaluate a given function at a point whose value is the concatenation of the inputs of the parties. In a nutshell, protocols for secure function evaluation are protocols that "emulate" an *ideal process* where all parties privately hand their inputs to an imaginary trusted party who privately computes the desired results, hands them back to the parties, and vanishes. A bit more precisely, it is required that for any adversary $\mathcal{A}$, that interacts with parties running a secure protocol π and induces some global output distribution, there exists an "ideal-process" adversary $\mathcal{S}$, that manages to obtain essentially the same global output distribution *in the ideal process.* The global output contains the adversary's output (which may be assumed to be his entire view of the computation), together with the identities and outputs of the uncorrupted parties. (Adversary $\mathcal{S}$ is often called a simulator, since it typically operates by simulating a run of $\mathcal{A}$.) The following parameters of the adversarial models turn out to be significant for our study.

ADVERSARIAL ACTIVITY: The adversary may be either passive (where even corrupted parties follow the prescribed protocol, and only try to gather additional information), or active, where corrupted parties are allowed to arbitrarily deviate from their protocol. Passive (resp., active) adversaries are often called honest-but-curious (resp., Byzantine).

NUMBER OF PLAYERS: We distinguish between the case of a small number of players, where n, the number of players, is $O(\log k)$, and a large number of players, where n is $\omega(\log k)$. (Here k is the security parameter.)

COMPLEXITY OF ADVERSARIES: We consider three cases. Information-Theoretic (IT) security does not take into account any computational complexity considerations. That is, both adversaries $\mathcal{A}$ and $\mathcal{S}$ have unbounded resources regardless of each other's resources. Universal security allows $\mathcal{A}$ unbounded resources, but requires $\mathcal{S}$ to be efficient (i.e., expected polynomial) in the complexity of $\mathcal{A}$. Computational security restricts both $\mathcal{A}$ and $\mathcal{S}$ to expected polynomial time (in the security parameter). Note that universal security implies both IT security and computational security (all other parameters being equal). However, IT security and computational security are incomparable. See [C00] for more discussion on the differences between these notions of security and their meaning.

QUALITY OF EMULATION: We consider either perfect emulation (where the output distributions of the real-life computation and of the ideal process must be identically distributed), or statistical emulation (where the output distributions shuld be statistically indistinguishable), or computational emulation (where the output distributions shuld be computationally indistinguishable).

The rest of the Introduction overviews the state of affairs regarding the added power of adaptivity, as discovered by our investigation. We do not attempt here to explain "why" things are as they are. Such (inevitably subjective) explanations require more familiarity with the definitions and are postponed to the body of the paper.

Our results: Canetti's definition. This definition is stated for several models of computation. We concentrate by default on the *secure channels* model, where the communication channels are perfectly secret and *universal security* is required. The same results hold also for the *computational* setting, where the adversary sees all communication but is restricted to polynomial time. Finally, we also consider a weaker variant of this definition, not considered in [C00], where only IT security is required (and the communication channels are secure).

The most distinctive parameter here seems to be whether the adversary is active or passive. If the adversary is active (i.e., Byzantine) then adaptive security is strictly stronger than non-adaptive security, regardless of the values of all other parameters. We show this via a protocol for three parties, that is non-adaptively universally secure with perfect emulation, but adaptively *in*secure, even if the adversary is computationally bounded and we are satisfied with computational emulation. This is the first such example involving only a constant number of players, for *any* constant.

In the case of passive adversaries the situation is more involved. Out of the nine settings to be considered (IT, universal, or computational security, with perfect, statistical, or computational emulation), we show that for one – IT security and perfect emulation – adaptive and non-adaptive security are equivalent, for any number of players. In all other eight settings we show that, roughly speaking, adaptive security is equivalent to non-adaptive security when the number of players is small, and is strictly stronger when the number of players is large. We elaborate below.

For a large number of players, it follows from an example protocol shown in [CFGN96] that for statistical or computational emulation, adaptive security is strictly stronger than non-adaptive security. We show separation also for perfect emulation, where universal or computational security is required. We complete the picture by showing that for a small number of players, and perfect emulation, adaptive and non-adaptive security are equivalent. Equivalence holds even in the case of statistical or computational emulation, if n is $O(\log k/\log\log k)$. (Notice that there is a small gap between this equivalence result and the known separating example for $n \in \omega(\log k)$. To close this gap, we also show that if one relaxes slightly the demands to the complexity of simulators and allows them to be expected polynomial time *except with negligible probability*, then this gap can be closed: equivalence holds for all $n \in O(\log k)$. In many cases, this definition of "efficient simulation" seems to be as reasonable as the standard one.)

Equivalence of adaptive and non-adaptive security for the case of passive adversaries and a small number of players is very good news: Many protocol problems (for instance, those related to threshold cryptography) make most sense in a setting where the number of parties is *fixed.* In such cases, when concentrating on passive adversaries, adaptivity comes "for free", which significantly simplifies the construction and analysis of these protocols.

Our results: Dodis-Micali-Rogaway definition. This definition holds for the *secure channels* setting only. It is incomparable to the definition of [C00]: On the one hand, it makes a number of additional requirements. On the other hand, only IT security is required. Here, to our surprise, adaptive and non-adaptive security turn out to be equivalent, even for active adversaries, and regardless of the number of players.

Two properties of the Dodis-Micali-Rogaway definition are essential for our proof of equivalence to work. The first is that only IT security is required. The second property may be roughly sketched as follows. It is required that there exists a stage in the protocol execution where all the parties are "committed" to their contributed input values; this stage must occur strictly before the stage where the output values become known to the adversary. (In order to formally state this requirement one needs to make some additional technical restrictions, amounting to what is known in the jargon as "one-pass black-box simulation". See more details within.)

Organization. Section 2 presents our results relating to the definition of [C00]. Section 3 presents our results relating to the definition of Dodis-Micali-Rogaway.

2 Adaptivity vs. Non-adaptivity in the definition of Canetti

This section describes our results relative to the [C00] definition of security. The main aspects of the definition that we will rely on were shortly described in the Introduction, and a more detailed review of the definition is provided in Section 2.1. Section 2.2 shows a separating example for the case of active adversary, Section 2.3 describes separating examples for passive adversary and a large number of players, Section 2.4 proves the equivalence for passive adversaries and a small number of players, and Section 2.5 shows the equivalence for passive adversaries in the setting of IT security and perfect emulation.

2.1 Review of the definition

For self containment this section briefly sketches the definitions of [C00] for the relevant settings. As stated in the Introduction, the following settings are considered. The adversary may be either non-adaptive or adaptive. It can also be either passive, or active. We distinguish between perfect, statistical and computational emulation, and also between the cases of universal, Information-Theoretic (IT) and computational security. Finally, we distinguish between the case of ideally secure channels, where the adversary has no access to information exchanged between uncorrupted parties, and the case of open channels, where the adversary learns all the communication among the parties. (We assume that the adversary cannot *modify* the communication among uncorrupted parties.)

In Section 2.1.1 we present the definition for the case of non-adaptive adversaries (both passive and active), and introduce universal, IT and computational security. In Section 2.1.2 we present the definition for the case of adaptive adversaries.

Preliminaries. We start by reviewing the notions of equal distribution and statistical and computational indistinguishability . A distribution ensemble $X = \{X(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ is an infinite sequence of probability distributions, where a distribution $X(k,a)$ is associated with the values of $k \in \mathbf{N}$ and $a \in \{0,1\}^*$. The distribution ensembles considered in the sequel are outputs of computations where the parameter a corresponds to various types of inputs, and the parameter k is a security parameter. All complexity characteristics of our constructions are measured in terms of the security parameter. In particular, we will be interested in the behavior of our constructions when the security parameter tends to infinity.

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

Ensembles X and Y are statistically indistinguishable *(written $X \stackrel{\mathrm{s}}{\approx} Y$) if for all $c > 0$ and for all large enough k we have $\mathrm{SD}(X(k,a),Y(k,a)) < k^{-c}$ where SD denotes statistical distance $\mathrm{SD}(Z_1,Z_2) = \frac{1}{2}\sum_a |\mathrm{Prob}(Z_1 = a) - \mathrm{Prob}(Z_2 = a)|$.*

Ensembles X and Y are computationally indistinguishable *(written $X \stackrel{\mathrm{c}}{\approx} Y$) if for all polynomial-time algorithm $\mathcal{D}$, for all $c > 0$ and for all large enough k we have $|\mathrm{Prob}(\mathcal{D}(1^k,a,x) = 1) - \mathrm{Prob}(\mathcal{D}(1^k,a,y) = 1)| < k^{-c}$, Where x is chosen from distribution $X(k,a)$, y is chosen from distribution $Y(k,a)$, and the probabilities are taken over the choices of x, y, and the random choices of $\mathcal{D}$.*

The functions to be evaluated by the parties are formalized as follows. An n-party function (for some $n \in \mathbf{N}$) is a probabilistic function $f : (D)^n \times \{0,1\}^* \rightarrow (D)^n$, for some finite domain D, where the last input is taken to be the random input.

Note that n, the number of parties, is treated as an unrelated quantity to the security parameter k. This allows capturing different relations between n and k, such as a constant n, n which is polynomial in k, or $n = \omega(\log k)$.

2.1.1 Non-adaptive Security

We first formalize the "real-life" model of computation. Next we formalize the ideal process. Finally we formalize the notion of emulation and state the definition. We develop the definitions for the cases of active and passive adversaries side by side, noting the differences throughout the presentation.

The real-life model. An n-party protocol π is a collection of n interactive, probabilistic algorithms. We use the term party P_i to refer to the ith algorithm. Each party P_i starts with value k for the security parameter, input $x_i \in D$, and random input $r_i \in \{0,1\}^*$. Let an adversary structure $\mathcal{B} \subset 2^{\{1...n\}}$ be a collection of subsets of $\{1...n\}$. A $\mathcal{B}$-limited real-life adversary, $\mathcal{A}$, is another algorithm determining the behavior of the corrupted parties. Adversary $\mathcal{A}$ starts off with security parameter k and input that contains the identities the corrupted parties (some set in $\mathcal{B}$), together with their inputs and random inputs. In addition, $\mathcal{A}$ receives auxiliary input z. The auxiliary input is a standard tool that allows proving the composition theorem. (Intuitively, the auxiliary input captures information gathered by the adversary from other interactions occurring before the current interaction.)

The computation proceeds in rounds, where each round proceeds as follows. First the uncorrupted parties generate their messages of this round, as described in the protocol. (That is, these messages appear on the outgoing communication tapes of the uncorrupted parties.) The messages addressed to the corrupted parties become known to $\mathcal{A}$ (i.e., they appear on the adversary's incoming communication tape). If the communication model is that of open channels then all the messages exchanged among the parties become known to $\mathcal{A}$. Next the adversary generates the messages to be sent by the corrupted parties in this round. If the adversary is passive then these messages are determined by the protocol. An active adversary determines the messages sent by the corrupted parties in an arbitrary way. Finally each uncorrupted party receives all the messages addressed to it in this round

At the end of the computation all parties locally generate their outputs. The uncorrupted parties output whatever is specified in the protocol. The corrupted parties output a special symbol, $\perp$, specifying that they are corrupted. In addition, the adversary outputs some arbitrary function of its view of the computation. The adversary view consists of its auxiliary input and random input, followed by the corrupted parties' inputs, random inputs, and all the messages sent and received by the corrupted parties during the computation. Without loss of generality, we can imagine that the real-life adversary's output consists of its entire view.

Let $\text{ADVR}_{\pi,\mathcal{A}}(k, \vec{x}, z, \vec{r})$ denote the output of real-life adversary $\mathcal{A}$ with security parameter k, auxiliary input z and when interacting with parties running protocol π on input $\vec{x} = x_1, \ldots, x_n$ and random input $\vec{r} = r_{\mathcal{A}}, r_1, \ldots, r_n$ as described above ($r_{\mathcal{A}}$ for $\mathcal{A}$, x_i and r_i for party P_i). Let $\text{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, z, \vec{r})_i$ denote the output of party P_i from this execution. Recall that if P_i is uncorrupted then this is the output specified by the protocol; if P_i is corrupted then $\text{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, z, \vec{r})_i = \perp$. Let

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

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

The ideal process. The ideal process is parameterized by the function to be evaluated. This is an n-party function $f:(D)^n\times\{0,1\}^*\rightarrow(D)^n$, as defined above. Each party P_i has security parameter k and input $x_i\in D$; no random input is needed for the parties in the ideal process (if f is a probabilistic function then the needed randomness will be chosen by the trusted party). Recall that the parties wish to compute $f(\vec{x},r_f)_1,\ldots,f(\vec{x},r_f)_n$, where r_f is an appropriately long random string, and P_i learns $f(\vec{x},r_f)_i$ (where $f(\vec{x},r_f)_i$ denote the ith component of $f(\vec{x},r_f)$). An ideal-process-adversary $\mathcal{S}$ is an algorithm describing the behavior of the corrupted parties. Adversary $\mathcal{S}$ starts off with security parameter k, the identities and inputs of the corrupted parties (parties P_i for $i\in C$), random input, and auxiliary input. In addition, there is an (incorruptible) trusted party, T. The ideal process proceeds as follows.

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

COMPUTATION: Each party P_i hands its (possibly modified) input value, y_i, to the trusted party T. Next, T chooses a value r_f randomly from $\mathcal{R}_f$, and hands each P_i the value $f(\vec{y},r_f)_i$.

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

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

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

Definition of security. We require that protocol π emulates the ideal process for evaluating f, in the following sense. For any real-life adversary $\mathcal{A}$ there should exist an ideal-process adversary $\mathcal{S}$, such that for any input vector $\vec{x}$ and any auxiliary input z, the global outputs $\text{IDEAL}_{f,\mathcal{S}}(\vec{x},z)$ and $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x},z)$ are similarly distributed.

We distinguish the following variants of this security requirement. First, if $\text{IDEAL}_{f,\mathcal{S}}(\vec{x},z)$ and $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x},z)$ are identically distributed (resp., statistically or computationally indistinguishable) then we say that the emulation is perfect (resp., statistical or computational). If both $\mathcal{A}$ and $\mathcal{S}$ are allowed unbounded computational resources, regardless of each other, then the security is information-theoretic (IT). If $\mathcal{A}$ is allowed unbounded computational resources, and $\mathcal{S}$ is required to be polynomial in the complexity of $\mathcal{A}$, then the security is universal. If both $\mathcal{A}$ and $\mathcal{S}$ are restricted to expected polynomial time then the security is computational.

Definition 2 *Let f be an n-party function and let π be a protocol for n parties. We say that π* non-adaptively, $\mathcal{B}$-securely evaluates f with IT security and perfect *(resp.,* statistical, computational*)* emulation *if for any $\mathcal{B}$-limited real-life adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that* $\text{IDEAL}_{f,\mathcal{S}} \stackrel{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A}}$ *(resp.,* $\text{IDEAL}_{f,\mathcal{S}} \stackrel{\text{s}}{\approx} \text{EXEC}_{\pi,\mathcal{A}}$ *or* $\text{IDEAL}_{f,\mathcal{S}} \stackrel{\text{c}}{\approx} \text{EXEC}_{\pi,\mathcal{A}}$*).*

If the running time of $\mathcal{S}$ runs in expected polynomial time in the running time of $\mathcal{A}$ then we say that π has universal security.

If both $\mathcal{A}$ and $\mathcal{S}$ are limited to expected polynomial time in the security parameter then we say that π has computational security.

If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries then we say that π $\mathcal{B}$-privately evaluates f.

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

2.1.2 Adaptive security

As in the non-adaptive case, we develop the definitions for the cases of active and passive adversaries side by side. One obvious difference from the definition of non-adaptive security is that here the adversary chooses the identities of the corrupted parties in an adaptive way; upon corruption, it sees the internal data of the corrupted party. (See more discussion on this point in [C00].)

An additional, more 'technical' difference is the way in which the interaction between the outside environment and a single protocol execution is captured. Capturing this interaction is useful for demonstrating that security is preserved under protocol composition. In the non-adaptive case this interaction is captured by the parties' inputs and outputs, plus an auxiliary input z given to the adversary before the computation starts. In the adaptive case a more involved construct is used. An additional entity, representing the external environment, is introduced to both the real-life model and the ideal process. This entity, called the environment and denoted $\mathcal{Z}$, is an algorithm that interacts with the adversary and the parties in a way described below. The notion of emulation is extended to include the environment.

The real-life model. Multiparty protocols are defined as in the non-adaptive case. An adaptive real-life adversary $\mathcal{A}$ is an algorithm that starts off with some random input. The environment is another algorithm, denoted $\mathcal{Z}$, that starts off with input z and random input. At certain points during the computation the environment interacts with the parties and the adversary. These points and the type of interaction are specified below. Let an adversary structure $\mathcal{B} \subset 2^{\{1...n\}}$ be a collection of subsets of $\{1...n\}$. An adversary is $\mathcal{B}$-limited if at all times the set of corrupted parties appears in $\mathcal{B}$.

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

Once a party is corrupted the party's input, random input, and the entire history of the messages sent and received by the party become known to $\mathcal{A}$. In addition, $\mathcal{Z}$ learns the identity of the corrupted party, and hands some additional auxiliary information to $\mathcal{A}$. (Intuitively, this information represents the party's internal data from other protocols run by the newly corrupted party.) From this point on $\mathcal{A}$ learns all the messages received by the party. If $\mathcal{A}$ is passive then the corrupted parties continue running protocol π. If $\mathcal{A}$ is active (Byzantine) then once a party becomes corrupted it follows the instructions of $\mathcal{A}$, regardless of protocol π.

At the end of the computation (say, at some pre-determined round) all parties locally generate their outputs. The uncorrupted parties output whatever is specified in the protocol. The corrupted parties output $\perp$. In addition, adversary $\mathcal{A}$ outputs some arbitrary function of its internal state.

Next, a "post-execution corruption (PEC) process" begins. (This process models the leakage of information from the current execution to the environment, caused by corrupting parties after the execution is completed. This process is necessary to guarantee secure composability of protocols in the adaptive setting.) First, $\mathcal{Z}$ learns the outputs of all the parties and of the adversary. Next $\mathcal{Z}$ and $\mathcal{A}$ interact in rounds, where in each round $\mathcal{Z}$ first generates a `'corrupt` P_i`'` request (for some P_i), and hands this request to $\mathcal{A}$. Upon receipt of this request, $\mathcal{A}$ may corrupt more parties as before (in which case $\mathcal{Z}$ learns their identity), and hands $\mathcal{Z}$ some arbitrary information. (Intuitively, this information is interpreted as P_i's internal data.) It is stressed that the set of corrupted parties is always in $\mathcal{B}$, even if $\mathcal{Z}$ requests to corrupt more parties; in this case $\mathcal{A}$ ignores the requests of $\mathcal{Z}$. The interaction continues until $\mathcal{Z}$ halts, with some output. Without loss of generality, this output can be $\mathcal{Z}$'s entire view of its interaction with $\mathcal{A}$ and the parties. Finally, the global output is defined to be the output of $\mathcal{Z}$. We use the following notation. Let the global output $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,\vec{x},z,\vec{r})$ denote $\mathcal{Z}$'s output on input z, random input r_Z and security parameter k, and after interacting with adversary $\mathcal{A}$ and parties running protocol π on inputs $\vec{x} = x_1 \ldots x_n$, random input $\vec{r} = r_Z, r_0 \ldots r_n$, and security parameter k as described above (r_0 for $\mathcal{A}$; x_i and r_i for party P_i). Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,\vec{x},z)$ denote the random variable describing $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,\vec{x},z,\vec{r})$ where $\vec{r}$ is uniformly chosen. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,\vec{x},z)\}_{k\in\mathbf{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$.[1]

The ideal process. As in the non-adaptive case, the ideal process is parameterized by the n-party function f to be evaluated. Each party P_i has security parameter k and input $x_i \in \{0,1\}^*$; no random input is needed. The model also involves an adaptive ideal-process-adversary $\mathcal{S}$, which is an algorithm that has random input r_0 and security parameter k, and an environment $\mathcal{Z}$ which is an algorithm that starts with input z, random input r_Z and the security parameter.In addition, there is an (incorruptible) trusted party, T. The ideal process proceeds as follows.

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

Computation stage: Once $\mathcal{S}$ completes the previous stage, the parties hand the following values

[1]The formalization of the global output $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ is different than in the non-adaptive case, in that here the global output contains *only* the output of the environment. We remark that the more complex formalization, where the global output contains the concatenation of the outputs of all parties and adversary, would yield an equivalent definition; this is so since the environment $\mathcal{Z}$ sees the outputs of all the parties and the adversary. We choose the current formalization for its simplicity.

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

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

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

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

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

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

Comparing computations in the two models. As in the non-adaptive case, we require that protocol π emulates the ideal process for evaluating f. Yet here the notion of emulation is slightly different. We require that for any real-life adversary $\mathcal{A}$ *and any environment* $\mathcal{Z}$ there should exist an ideal-process adversary $\mathcal{S}$, such that $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \stackrel{\mathrm{d}}{=} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. Note that the environment is the same in the real-life model and the ideal process. This may be interpreted as saying that "for any environment and real-life adversary $\mathcal{A}$, there should exist an ideal-process adversary that successfully simulates $\mathcal{A}$ *in the presence of this specific environment.*" As in the non-adaptive case, we distinguish perfect and statistical emulation, as well as universal, IT, and computational security.

Definition 3 *Let f be an n-party function and let π be a protocol for n parties. We say that π* adaptively, $\mathcal{B}$-securely evaluates f with IT security and perfect *(resp.,* statistical, computational*)* emulation *if for any $\mathcal{B}$-limited real-life adversary $\mathcal{A}$ and any environment $\mathcal{Z}$ there exists an ideal-process adversary $\mathcal{S}$ such that* $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \stackrel{\mathrm{d}}{=} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ *(resp.,* $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \stackrel{\mathrm{s}}{\approx} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ *or* $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \stackrel{\mathrm{c}}{\approx} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$*).*

If the running time of $\mathcal{S}$ runs in expected polynomial time in the running time of $\mathcal{A}$ then we say that π has universal security.

If both $\mathcal{A}$ and $\mathcal{S}$ are limited to expected polynomial time in the security parameter then we say that π has computational security.

If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries then we say that π $\mathcal{B}$-privately evaluates f.

Finally we also distinguish between the case of security with PEC, where the interaction proceeds as described above, and the case of security without PEC, where the post-execution corruption stage is omitted, both in the real-life model and in the ideal process.

2.2 Separation for active adversaries

This section shows that adaptive and non-adaptive security are not equivalent in the case of active adversaries, for all settings considered here: information-theoretic, universal, and computational security, with perfect, statistical, or computational emulation. This is proved by an example of a simple protocol for secure function evaluation which is non-adaptively secure, but adaptively insecure, in all above settings.

Our protocol involves three players D, R_1, R_2, where R_1, R_2 have no input, and D's input consists of two bits $s_1, s_2 \in \{0, 1\}$. The function f_{act} to be computed is the function that returns no output for D, s_1 for R_1, and s_2 for R_2. The *adversary structure* $\mathcal{B}$ (the collection of player subsets that can be corrupted) contains all subsets of $\{D, R_1\}$, namely the only restriction is that R_2 cannot be corrupted. The protocol π_{act} proceeds as follows.

1. D sends s_1 to R_1.
2. D sends s_2 to R_2.
3. Each R_i outputs the bit that was sent to it by D, and terminates. D outputs nothing and terminates.

Claim 4 *The protocol π_{act} non-adaptively, perfectly emulates f_{act} with universal security, against active adversary structure $\mathcal{B}$.*

Proof: Consider a non-adaptive real-life adversary $\mathcal{A}$ that corrupts D. The ideal-process simulator $\mathcal{S}$ proceed as follows. $\mathcal{S}$ corrupts D in the ideal model, and provides $\mathcal{A}$ with the inputs s_1, s_2 of D. $\mathcal{A}$ generates s'_1 to be sent to R_1 and s'_2 to be sent to R_2. $\mathcal{S}$ gives s'_1, s'_2 to the trusted party as D's input, outputs $\mathcal{A}$'s output, and terminates. It is easy to see that the global output generated by $\mathcal{S}$ in the ideal model is identical to the global output with the real-life $\mathcal{A}$.

The above simulator can be easily modified for the case that $\mathcal{A}$ breaks into both D and R_1 (here $\mathcal{S}$ may hand in to the trusted party $0, s'_2$ as the input of D, where s'_2 is the message prepared by $\mathcal{A}$ to be sent to R_2).

Finally, consider $\mathcal{A}$ that corrupts only R_1. The simulator $\mathcal{S}$ proceeds as follows. $\mathcal{S}$ corrupts R_1 in the ideal model, hands the empty input to the trusted party, and obtains the output s_1 in the ideal model. $\mathcal{S}$ then hands s_1 to $\mathcal{A}$ as the message that was sent from D to R_1, outputs $\mathcal{A}$'s output, and terminates. Again it is easy to see that the global output generated by $\mathcal{S}$ is identical to the global output with $\mathcal{A}$. $\square$

Claim 5 *The protocol π_{act} is adaptively insecure for evaluating the function f_{act}, with either universal, IT or computational security, against active adversary structure $\mathcal{B}$.*

Proof: We show an adaptive efficient real life adversary $\mathcal{A}$, such that there is no (even computationally unbounded) adaptive ideal-model adversary (simulator) $\mathcal{S}$ that can emulate the global view induced by $\mathcal{A}$ (even if the emulation is only required to be computational). Intuitively, the goal of our adversary is to ensure that whenever $s_1 = s_2$, R_2 will output 0, whereas we do not care what happens in other cases. $\mathcal{A}$ starts by corrupting R_1 and receiving s_1 in the first stage of the protocol. If $s_1 = 0$, $\mathcal{A}$ terminates. If $s_1 = 1$, $\mathcal{A}$ corrupts D and sends $s'_2 = 0$ to R_2 in the second stage of the protocol.

To prove that this $\mathcal{A}$ cannot be simulated in the ideal world, note that in the real world, $\mathcal{A}$ never corrupts D when D's input is $s_1 = s_2 = 0$, but always corrupts D when D's input is $s_1 = s_2 = 1$. In both these cases, R_2 always outputs 0. Now let $\mathcal{S}$ be an arbitrary unbounded adaptive ideal-process simulator. (Below "overwhelming probability" refers to $1 - neg$ for some negligible function *neg*.) If, when interacting with $\mathcal{S}$ in the ideal model, whenever $s_1 = s_2 = 1$, R_2 outputs 0 with overwhelming probability, then it must be that with overwhelming probability, whenever $s_1 = s_2 = 1$, $\mathcal{S}$ corrupts D in the first corruption stage (before the function is computed by the trusted party). However, recall that in the first corruption stage in the ideal process, corrupting a party provides only its input, and no other information. Thus, in our case, before D is corrupted $\mathcal{S}$ cannot gain any information. It follows that $\mathcal{S}$ corrupts D in the first corruption stage with the same probability for any input s_1, s_2, and in particular when the input is $s_1 = s_2 = 0$. However in the real world, $\mathcal{A}$ never corrupts D in this case, and so the global views are significantly different. □

Claim 4 and Claim 5 together imply that our example separates adaptive security from non-adaptive security for active adversaries in all settings considered. Thus we have:

Theorem 6 *For active adversaries, adaptive security is strictly stronger than non-adaptive security, under any notion of security, as long as there are at least three parties.*

Discussion. The essential difference between adaptive and non-adaptive security is well captured by the simplicity of the protocol used in our separating example, which at first look may seem like a very "harmless" protocol. Indeed, π_{act} is a straight-forward implementation of the function f_{act}, which just "mimics" the ideal-world computation, replacing the trusted party passing input from one party to the output of another party, by directly sending the message between the parties. For the non-adaptive setting, this intuition translates into a proof that any adversary $\mathcal{A}$ can be simulated by an adversary $\mathcal{S}$ in the ideal world. However, as we have shown, the protocol is susceptible to an attack by an adaptive adversary.

In the heart of this separation is the idea that some information in the protocol (the value of s_1 in our example) is revealed prematurely before the parties have "committed" to their inputs. Thus, an adaptive adversary may take advantage of that by choosing whether to corrupt a party (and which one) based on this information, and then changing the party's input to influence the global output of the execution.

On the other hand, as we will show, for a passive adversary and information theoretic security, non-adaptive security is equivalent to adaptive security. This may suggest the intuition that even for active adversaries, in the information-theoretic setting, adaptive and non-adaptive security may be equivalent for a subclass of protocols that excludes examples of the above nature; that is, for protocols where "no information is revealed before the parties have committed to their inputs". This is in fact the case for many existing protocols (cf., [BGW88, CDM98]), and furthermore, the definition of Dodis-Micali-Rogaway *requires* this condition. In Section 3 we indeed formalize and prove this intuition, showing equivalence for the definition of Dodis-Micali-Rogaway.

Finally, we remark that for *two* parties and active adversaries, the situation is more involved: In the IT setting, adaptive security is equivalent to non-adaptive security. In the universal and computational settings, we have a separating example showing that adaptive security is strictly stronger, assuming perfectly hiding bit-commitment exists (which holds under standard complexity assumptions). However, this example heavily relies on a technical requirement, called post-execution corruptibility (PEC), which is part of the definition of adaptive security, needed in order to guarantee secure composability of protocols (the technical meaning of the requirement is described along with the definition in Section 2.1). In contrast, the above three party separating example holds in all settings, regardless of whether the PEC requirement is imposed or not.[2]

2.3 Separation for passive adversaries and a large number of players

In [CFGN96], Canetti et al. show an example protocol that separates adaptive and non-adaptive security for passive adversaries and a large number of players, when only statistical or computational emulation is required. This separation holds for universal, IT, and computational security. Very roughly, the protocol is based on sharing a secret among a large set of players, making the identity of the set very hard to guess for a non-adaptive adversary, but easy for an adaptive one. We refer the reader to [CFGN96] for details of the example.

To complete the picture, we show an example that, under standard complexity assumptions, separates adaptive and non-adaptive security even when perfect emulation is required, for the universal or computational security model. The example is only sketched here, and the complete proof and definitions of the standard primitives used, are deferred to the final version of the paper.

Our example relies on the existence of perfectly hiding bit commitment schemes and collision-intractable hash functions.[3] For n players, we will need to hash n commitments in a collision-intractable manner. Thus, the number of players required depends on the strength of the assumption: For n that is polynomial in the security parameter k, this is a standard assumption, whereas for $n = \omega(\log k)$ this requires a stronger assumption. For simplicity, we refer below to a large number of players, instead of making the explicit distinction based on the quality of computational assumption.

The protocol involves players $P_0, P_1, \ldots, P_n$, where the input of P_0 is a function h from a family of collision intractable hash functions, and a public key pk for a perfectly hiding bit commitment scheme. The input of each other P_i is a bit b_i. The output of each player is h, pk. The protocol proceeds as follows:

1. P_0 sends h, pk to all other players.
2. Each P_i, $i \geq 1$ computes and broadcasts a commitment $c_i = commit(pk, b_i, r_i)$.
3. All players output h, pk.

We allow the adversary to corrupt P_0 and in addition any subset of size $n/2$ of the other players.

It is straightforward to check that this protocol is non-adaptively secure: the simulator asks to compute the function in the ideal process immediately, learns h, pk and can now perfectly simulate any message from non-corrupted players.

On the other hand, consider an adaptive adversary $\mathcal{A}$, who will first corrupt P_0, listen to the messages from the first two steps and then compute $h(c_1, .., c_n)$. Then $\mathcal{A}$ interprets the result in some fixed deterministic way as a subset of the players $P_1, .., P_n$ of size $n/2$, and corrupts this subset. Assuming that a simulator $\mathcal{S}$ exists for this adversary, the following algorithm breaks either the commitment scheme or the hash function family. The algorithm gets input h, pk and then proceeds as follows:

[2] The setting of two parties *without* PEC is only of interest if we are considering a 2-party protocol as a standalone application, without composing it with multi-player protocols. For this setting, we can prove equivalence of adaptive and non-adaptive security in the secure channels model or when the simulation is black box.

[3] This example is an extension of another example given in [CFGN96], which uses only bit commitment, and works only for black-box simulators.

1. Run $\mathcal{S}$ on random input r.
2. When $\mathcal{S}$ corrupts P_0, give it pk, h as the input.
3. When $\mathcal{S}$ corrupts a player P_i, $i \geq 1$, give it 0 as input bit for P_i.
4. Let v_0 be the view for $\mathcal{A}$ output by $\mathcal{S}$ and let P_j be the last player $\mathcal{S}$ corrupted when generating v_0.
5. Rewind $\mathcal{S}$ to its state just after Step 1. Run $\mathcal{S}$ forward again, and give it the same input values for corrupted players, except for P_j where we give a 1 as input bit. Let v_1 be the view produced this time.
6. Use v_0, v_1 to either break the hash function or the commitment scheme.

The reason why this works as required is first that the set of corrupted players must be the same in v_0 and v_1, since $\mathcal{A}$ always corrupts P_0 and $n/2$ players of the rest, and all of $\mathcal{S}$'s input in the two runs is the same until after the last corruption happens. Therefore the hash values computed from round 2 messages in the two views are the same. Now, it may be that the commitments $c_1, ..., c_n$ appearing in v_0 are not the same as those in v_1, in which case we have a collision for h. Otherwise c_n appears in both views, and these contain information on how to open it as both 1 and 0.

We thus have the following theorem.

Theorem 7 *For passive adversaries and a large number of parties, adaptive security is strictly stronger than non-adaptive security, under all notions of security except IT with perfect emulation. This holds unconditionally for either statistical or computational emulation, and under the assumption that a perfectly hiding bit commitment scheme and a collision intractable hash function family exist, for perfect emulation.*

2.4 Equivalence for passive adversaries and a small number of parties

This section proves that adaptive and non-adaptive security against a *passive* adversary are equivalent when the number of parties is small.

Before going into our results, we need to elaborate on a simplifying assumption we make in this section. As previously mentioned, the [C00] definition of adaptive security (as well as [B91, MR91], in different ways) include a special technical requirement, called post-execution corruptibility (PEC). This requirement is in general needed in order to guarantee secure composition of protocols in the adaptive setting (see Section 2.1 for more technical details about PEC).

However, in the particular setting of this section, i.e. passive adversaries and a small number of players, it turns out that PEC is an "overkill" requirement for guaranteeing composability of protocols. Very informally, the argument for this is the following. Let π and ρ be protocols that are adaptively secure without the PEC property. These protocols are (of course) also non-adaptively secure. Since the non-adaptive definition of security is closed under (non-concurrent) composition [C00], it follows that the 'composed' protocol, $\pi \circ \rho$, is non-adaptively secure. By our result given below, the composed protocol is also adaptively secure (without PEC).

We conclude that in the setting of this section, PEC is not needed to guarantee adaptively secure composition, and therefore we discuss in this section only results that hold without assuming the PEC requirement.[4]

[4]If we were to assume the PEC requirement, we can in fact show a two-party protocol which is non-adaptively secure, but adaptively insecure (this is the same example based on perfectly hiding bit commitment which was mentioned in the end of Section 2.2). Thus, strictly speaking, there is a separation in this setting under the [C00] definition. The results in other sections hold regardless of whether the PEC requirement is imposed or not.

We first note that the general definition detailed in Section 2.1 takes a simpler form in the passive case. In particular, in the passive case we may assume without loss of generality that the real-life adversary waits until the protocol terminates, and then starts to adaptively corrupt the parties; corrupting parties at an earlier stage is clearly of no advantage in the passive case. Similarly, the ideal-process adversary may be assumed to corrupt parties after the ideal function evaluation terminates. To further ease the exposition, we will make in the remainder of this section the following simplifying assumptions: (1) assume that the adversary is deterministic; (2) assume that the function computed by the protocol is deterministic; and (3) ignore auxiliary inputs. The results in this section generalize to hold without the above assumptions.

The card game

In attempting to prove equivalence between non-adaptive and adaptive security, it may be helpful to picture the following game. Let $\mathcal{B} \subseteq 2^{[n]}$ be a monotone adversary structure. The game involves two players, the *adversary* and the *simulator*, and n distinct cards. The two players are bound to different rules, as specified below.

Adversary. When the adversary plays, the faces of the n cards are picked from some (unknown) joint distribution $V = (V_1, \ldots, V_n)$ and are initially covered. The adversary proceeds by sequentially uncovering cards according to a fixed deterministic strategy; that is, the choice of the next card to be uncovered is determined by the contents of previously uncovered cards. Moreover, the index set of uncovered cards should always remain within the confines of the structure $\mathcal{B}$. After terminating, the adversary's output consists of the identity and the contents of all uncovered cards.

Simulator. The simulator plays in a different room. It is initially given n distinct *blank* cards, all of which are covered. Similarly to the adversary, it is allowed to gradually uncover cards, as long as the set of uncovered cards remains in $\mathcal{B}$. Its goal is to fill the blank uncovered cards with content, so that the final configuration (including the identity and contents of uncovered cards) is "similarly" distributed to the adversary's output. (The precise sense of this similarity requirement will depend on the specific security setting.) Note that unless the simulator has some form of access to the unknown distribution V, the game would not make much sense. Indeed, we grant the simulator the following type of restricted access to V. At each stage, when the set of uncovered cards is some $b \in \mathcal{B}$, the simulator may freely sample from some fixed distribution $\tilde{V}_b$ which is guaranteed to be "similar" to V_b, the restriction of V to b. (Again, the type of this similarity depends on the setting.) The $|\mathcal{B}|$ distributions V_b may be arbitrarily (or adversarially) fixed, as long as they conform to the above similarity condition.

Let us briefly explain the analogy between the above game and the question of non-adaptive versus adaptive security. Fix some n-party protocol π computing a deterministic function f, and suppose that π is non-adaptively secure against a passive $\mathcal{B}$-limited adversary. The n cards correspond to the n parties. The distribution V corresponds to the parties' joint view under an input x, which is a-priori unknown. Uncovering the i-th card by the adversary and learning V_i corresponds to corrupting the i-th party P_i in the real-life process and learning its entire view: its input, random input, communication messages, and output. Uncovering the i-th card by the simulator corresponds to corrupting P_i in the ideal-model process. Finally, each distribution $\tilde{V}_b$ from which the simulator can sample corresponds to a simulation of a non-adaptive adversary corrupting b, which exists under the assumption that π is non-adaptively secure. Note that the simulator can access $\tilde{V}_b$ only when all cards in b are uncovered; this reflects the fact that the non-adaptive simulation cannot proceed without learning the inputs and outputs of corrupted parties. The types of similarity between V_b and $\tilde{V}_b$ we will consider are *perfect*, *statistical*, and *computational*, corresponding to the type of non-adaptive emulation we assume. We will also consider the relation between the computational complexity of the adversary and that of the simulator, addressing the security variants in which the simulator is computationally bounded.

Remark. The above game models a secure channels setting, in which the adversary has no information before corrupting a party. To model open channels (or a "broadcast" channel), the distribution V should be augmented with an additional entry V_0, whose card is initially uncovered. The analysis that will follow can be easily adapted to deal with this more general setting.

2.4.1 Perfect emulation

We first deal with perfect emulation, i.e., the case where $\tilde{V}_b = V_b$ for all $b \in \mathcal{B}$. In this setting, we show how to construct an adaptive simulator running in (expected) time polynomial in the time of the adversary and the size of the adversary structure. The construction from this section will allow us to prove equivalence of non-adaptive and adaptive security both in the information-theoretic case (see Section 2.5) and, when the adversary structure is small, in the universal case.

A black-box simulator. To prove equivalence between non-adaptive and adaptive security it suffices to show that for any adversary strategy $\mathcal{A}$ there exists a simulator strategy $\mathcal{S}$, such that under any distribution V the simulator wins. In fact, we will construct a single simulator $\mathcal{S}$ with a black-box access to $\mathcal{A}$, and later analyze it in various settings.

A convention for measuring the running time of black-box simulators. In the following we view adaptive simulators as algorithms supplied with two types of oracles: *distribution oracles* $\tilde{V}_b$, implemented by a non-adaptive ideal-process adversary (to be referred to as a *non-adaptive simulator*), and an adaptive *adversary oracle* $\mathcal{A}$. In measuring the running time of a simulator, each oracle call will count as a single step. This convention is convenient for proving universal security: If the protocol has universal non-adaptive security and the black-box simulator $\mathcal{S}$ runs in expected time poly(k) then, after substituting appropriate implementations of the oracles, the expected running time of $\mathcal{S}$ is polynomial in k and the expected running time of $\mathcal{A}$.[5]

In the description and analysis of $\mathcal{S}$ we will use the following additional notation. By v_b, where v is an n-tuple (presumably an instance of V) and $b \subseteq [n]$ is a set, we denote the restriction of v to its b-entries. For notational convenience, we assume that the entries of a partial view v_b, obtained by restricting v or by directly sampling from $\tilde{V}_b$ or V_b, are labeled by their corresponding b-elements (so that b can be inferred from v_b). We write $v \xrightarrow{\mathrm{A}} b$ if the joint card contents (view) v leads the adversary $\mathcal{A}$ to uncover (corrupt) the set b *at some stage*. For instance, $v \xrightarrow{\mathrm{A}} \emptyset$ always holds. An important observation is that whether $v \xrightarrow{\mathrm{A}} b$ holds depends only on v_b. This trivially follows from the fact that cards cannot be covered once uncovered. Hence, we will also use the notation $v' \xrightarrow{\mathrm{A}} b$, where v' is a $|b|$-tuple representing a partial view.

In our description of the simulator we will adopt the simplified random variable notation from the game described above, but will revert to the original terminology of corrupting parties rather than uncovering cards.

Before describing our simulator $\mathcal{S}$, it is instructive to explain why a simpler simulation attempt fails. Consider a "straight line" simulator which proceeds as follows. It starts by corrupting $b = \emptyset$.

[5]Note that when the protocol has universal non-adaptive security, a distribution $\tilde{V}_b$ can be sampled in expected polynomial time from the view of an ideal-process adversary corrupting b.

At each iteration, it samples $\tilde{V}_b$ and runs the adversary on the produced view to find the first party outside b it would corrupt. The simulator corrupts this party, adds it to b, and proceeds to the next iteration (or terminates with the adversary's output if the adversary would terminate before corrupting a party outside b). This simulation approach fails for the following reason. When sampling $\tilde{V}_b$, the produced view is independent of the event which has lead the simulator to corrupt b. This makes it possible, for instance, that the simulator corrupts a set which cannot be corrupted at all in the real-life execution. The simulator $\mathcal{S}$, described next, will fix this problem by insisting that the view sampled from $\tilde{V}_b$ be consistent with the event that the adversary corrupts b.

Algorithm of $\mathcal{S}$:

1. Initialization:
   Let $b_0 = \emptyset$. The set b_i will contain the first i parties corrupted by the simulator.

2. For $i = 0, 1, 2, \ldots$ do:

   (a) Repeatedly sample $v' \stackrel{\text{R}}{\leftarrow} \tilde{V}_{b_i}$ (by invoking the non-adaptive simulator) until $v' \stackrel{\text{A}}{\rightarrow} b_i$ (i.e., the sampled partial view would lead $\mathcal{A}$ to corrupt b_i). Let v_i be the last sampled view. (Recall that v' includes the identities of parties in b_i.)

   (b) Invoke $\mathcal{A}$ on v_i to find the index p_{i+1} of the party which $\mathcal{A}$ is about to corrupt next (if any). If there is no such party (i.e., $\mathcal{A}$ terminates), output v_i. Otherwise, corrupt the p_{i+1}-th party, let $b_{i+1} = b_i \cup \{p_{i+1}\}$, and iterate to the next i.

In the remainder of this section we analyze the performance of the simulator $\mathcal{S}$. To this end, let $\tilde{B}_i, \tilde{V}_i$ be random variables containing the corrupted set b_i and the partial view v_i in the i-th iteration of $\mathcal{S}$. Similarly, let B_i, V_i be the corresponding random variables induced by the real-life execution of $\mathcal{A}$. In the event that an execution terminates *before* the i-th iteration, the random variables indexed by i will be set to a special value '$\perp$'.

Lemma 8 *In the case of a perfect non-adaptive emulation, for every iteration i and set $b_i \in \mathcal{B}$, the distribution $\tilde{V}_i$ conditioned on $\tilde{B}_i = b_i$ is identical to V_i conditioned on $B_i = b_i$.*

Proof: If $b_i = \perp$ then both V_i conditioned on $B_i = b_i$ and $\tilde{V}_i$ conditioned on $\tilde{B}_i = b_i$ are forced to be $\perp$. Otherwise, it follows from the description of $\mathcal{S}$ that $\tilde{V}_i$ given $\tilde{B}_i = b_i$ is sampled from the distribution $V_{b_i}(\stackrel{\text{d}}{=} \tilde{V}_{b_i})$ conditioned on the event $V_{b_i} \stackrel{\text{A}}{\rightarrow} b_i$. On the other hand, the distribution $V_i(= V_{B_i})$ conditioned on $B_i = b_i$ is the same as V_{b_i} conditioned on $B_i = b_i$, which in turn is the same as V_{b_i} conditioned on $V \stackrel{\text{A}}{\rightarrow} b_i$ (since $B_i = b_i$ and $V \stackrel{\text{A}}{\rightarrow} b_i$ are two names for the same the event). $\square$

Lemma 9 *In the case of a perfect non-adaptive emulation, $\tilde{V}_i \stackrel{\text{d}}{=} V_i$ for every i.*

Proof: It follows from Lemma 8 that if $\tilde{B}_i \stackrel{\text{d}}{=} B_i$ then $\tilde{V}_i \stackrel{\text{d}}{=} V_i$. It thus suffices to show that $\tilde{B}_i = B_i$ for all i. Clearly, both $\tilde{B}_0$ and B_0 are deterministically the empty set. Now, suppose that $\tilde{B}_i \stackrel{\text{d}}{=} B_i$ and hence also $\tilde{V}_i \stackrel{\text{d}}{=} V_i$. We show that $\tilde{B}_{i+1} \stackrel{\text{d}}{=} B_{i+1}$ by conditioning on the i-th iteration's view v_i. The crucial observation is that $\tilde{B}_{i+1}$ is determined by the simulator from v_i in the same way that B_{i+1} it is determined by the adversary from v_i. In particular, if $v_i = \perp$ or if v_i leads $\mathcal{A}$ or $\mathcal{S}$ to terminate, then $\tilde{B}_{i+1}$ and B_{i+1} are both set to $\perp$. It follows that the conditional distribution of $\tilde{B}_{i+1}$ given $\tilde{V}_i = v_i$ is the same as B_{i+1} given $V_i = v_i$, from which it follows that $\tilde{B}_{i+1} \stackrel{\text{d}}{=} B_{i+1}$. $\square$

Claim 10 *In the case of a perfect non-adaptive emulation, $\mathcal{S}$ perfectly emulates $\mathcal{A}$.*

Proof: We need to show that the output distributions of $\mathcal{S}$ and $\mathcal{A}$ are identical. Note that the *joint* distribution $(\tilde{V}_1, \tilde{V}_2, \ldots, \tilde{V}_n)$ may be different from $(V_1, V_2, \ldots, V_n)$. For instance, the path $\mathcal{S}$ takes in arriving at a set b_i may be impossible for $\mathcal{A}$ to take in the real-life process. However, from $\tilde{V}_i$ (respectively, V_i) *alone*, one may determine: (1) the probability of $\mathcal{S}$ (resp., $\mathcal{A}$) terminating in the i-th iteration; and (2) the output distribution of $\mathcal{S}$ (resp., $\mathcal{A}$) given that it terminates in the i-th iteration. Using Lemma 9 it follows that the outputs are identically distributed. □

We turn to analyze the complexity of $\mathcal{S}$, still for a perfect non-adaptive emulation. A trivial observation is that the number of iterations is bounded by the number of parties (or, more precisely, by the maximal size of a set in $\mathcal{B}$). The complexity of each iteration, however, may be unbounded. The following lemma implies a bound on the total *expected* running time of $\mathcal{S}$.

Claim 11 *In the case of perfect non-adaptive emulation, the expected running time of $\mathcal{S}$ is linear in $|\mathcal{B}|$.*

Proof: We count the expected number of times a view v' is sampled (in step 2a) throughout the execution of $\mathcal{S}$. Let $\tilde{T}_i$ be a random variable counting the number of samples taken in the i-th iteration, and $\tilde{T} = \sum_i \tilde{T}_i$ be the total number of samples. To bound $E[\tilde{T}_i]$, we condition this expectation on the set b_i corrupted by the simulator in the i-th iteration. Given that $\tilde{B}_i = b_i$ ($b_i \neq \perp$), $\tilde{T}_i$ is distributed as the number of independent samples taken from $\tilde{V}_{b_i}$ until obtaining one that would lead the adversary to b_i. The success probability of a single sampling attempt is:

$$\text{Prob}[\tilde{V}_{b_i} \xrightarrow{\text{A}} b_i] = \text{Prob}[V_{b_i} \xrightarrow{\text{A}} b_i] = \text{Prob}[B_i = b_i] = \text{Prob}[\tilde{B}_i = b_i]$$

where the first equality relies on the perfect emulation assumption, the second on equality of the relevant events, and the third on Lemma 9. Hence $E[\tilde{T}_i | \tilde{B}_i = b_i] = 1/\text{Prob}[\tilde{B}_i = b_i]$. Letting $\mathrm{sup(\tilde{B}_i)}$ denote the support set of the random variable $\tilde{B}_i$ (excluding $\perp$) we obtain:

$$E[\tilde{T}_i] = \sum_{b_i \in \mathrm{sup(\tilde{B}_i)}} E[\tilde{T}_i | \tilde{B}_i = b_i] \cdot \text{Prob}[\tilde{B}_i = b_i] = \sum_{b_i \in \mathrm{sup(\tilde{B}_i)}} (1/\text{Prob}[\tilde{B}_i = b_i]) \cdot \text{Prob}[\tilde{B}_i = b_i] = |\mathrm{sup(\tilde{B}_i)}|.$$

Finally, since all sets in $\mathrm{sup(\tilde{B}_i)}$ are of size i, we have: $E[\tilde{T}] = \sum_i E[\tilde{T}_i] = \sum_i |\mathrm{sup(\tilde{B}_i)}| \leq |\mathcal{B}|$ as required. □

If $n = O(\log k)$, $|\mathcal{B}|$ is guaranteed to be polynomial in k. We may thus conclude from Claim 10 and Claim 11 the following:

Theorem 12 *For function evaluation protocols with passive adversary, universal perfect security, and $n = O(\log k)$ parties, adaptive and non-adaptive security are equivalent.*

2.4.2 Imperfect Emulation

We next address the cases of statistical and computational security against a passive adversary. Suppose that we are given an imperfect (statistical or computational) non-adaptive simulator and attempt to construct an adaptive one. If we use exactly the same approach as before, some technical problems arise: with imperfect non-adaptive emulation, it is possible that a real life adversary $\mathcal{A}$ corrupts some set with a very small probability, whereas this set is *never* corrupted in emulated views. As a result, the loop in step (2a) of the algorithm of $\mathcal{S}$ will never terminate, and the expected time will be infinite. Consequently, it is also unclear whether $\mathcal{S}$ will produce a good output distribution when given access to imperfect non-adaptive simulation oracles $\tilde{V}_b$.

We start by showing that when the size of the adversary structure is polynomial, the simulator $\mathcal{S}$ will indeed produce a (statistically or computationally) good output distribution even when given access to (statistically or computationally) imperfect non-adaptive simulators. Moreover, it will turn out that when the adversary structure is polynomial, the expected running time of $\mathcal{S}$ is polynomial *except with negligible probability.* Later, we define a more sophisticated simulator $\mathcal{S}'$ which achieves strict expected-polynomial time simulation, at the expense of requiring a stronger assumption on the size of the adversary structure.

A main tool in the following is a technical lemma referred to in the sequel as the *adaptive sampling lemma.* For simplicity we will use a non-uniform notion of computational indistinguishability. The lemma and its corollaries can be extended to the uniform setting well. The lemma will use the following terminology and notation. Let $D = \{D(k)\}_{k\in\mathcal{N}}$ be a distribution ensemble. An *adaptive sampling algorithm* S is an algorithm which, given oracle access to D and an input 1^k, may take a variable number of independent samples from $D(k)$. At each stage, based on all previous samples, the algorithm decides whether to take an additional sample or to terminate. Upon termination, the algorithm outputs some function of all samples it took. Let $S^D(k)$ denote the output distribution of S running with oracle access to D and $\mathrm{Time}(S^D(k))$ be a random variable measuring the running time of the corresponding execution, where an oracle call is counted as a single step.

We first state and prove the computational version of the lemma, and then state its statistical version.

Lemma 13 (Adaptive Sampling Lemma: Computational Version) *Let $C = \{C(k)\}, D = \{D(k)\}$ be two distribution ensembles such that $C \stackrel{c}{\approx} D$, and S be an adaptive sampling algorithm such that S^C runs in expected polynomial time. Then:*

1. *$S^C \stackrel{c}{\approx} S^D$;*
2. *S^D runs in expected polynomial time except with negligible probability. That is, in the execution of S^D there exists an event occurring with negligible probability, such that conditioned on the complement of this event the expected running time is polynomial.*

Proof: Assume towards contradiction that P is an efficient distinguisher between S^C, S^D. That is, there exists a constant $c > 0$ such that for some infinite $K\subseteq\mathcal{N}$

$$|\mathrm{Prob}[P(S^C(k)) = 1] - \mathrm{Prob}[P(S^D(k)) = 1]| > k^{-c} \tag{1}$$

for all $k \in K$. Let $p(k)$ be a polynomial bounding the running time of S^C. Define a *non-adaptive* sampler S_0 which first samples its oracle $q(k) = 3p(k)k^c$ times, and then simulates S on the samples it generated. If S terminates, S_0 outputs the same output as S. If S attempts to make an additional sample beyond the $q(k)$ samples S_0 can provide, S_0 terminates and outputs a special symbol. By Markov inequality, for all $k \in K$ we have $\mathrm{Prob}[\mathrm{Time}(S^C)(k) > q(k)] \leq \frac{1}{3}k^{-c}$, hence

$$\mathrm{SD}(S^C(k), S_0^C(k)) \leq \frac{1}{3}k^{-c}. \tag{2}$$

It follows from the robustness of computational indistinguishability under efficient *non-adaptive* sampling (cf. [G95]) that $S_0^C \stackrel{c}{\approx} S_0^D$ and hence:

$$|\mathrm{Prob}[P(S_0^C(k)) = 1] - \mathrm{Prob}[P(S_0^D(k)) = 1]| \leq k^{-\omega(1)}. \tag{3}$$

From the computational indistinguishability of S_0^C, S_0^D it also follows that $|\mathrm{Prob}[\mathrm{Time}(S^C(k)) > q(k)] - \mathrm{Prob}[\mathrm{Time}(S^D(k)) > q(k)]| \leq k^{-\omega(1)}$, from which we can conclude that for every $k \in K$, $\mathrm{Prob}[\mathrm{Time}(S^D)(k) > q(k)] \leq \frac{1}{3}k^{-c} + k^{-\omega(1)}$ and

$$\mathrm{SD}(S_0^D(k), S^D(k)) \leq \frac{1}{3}k^{-c} + k^{-\omega(1)}. \tag{4}$$

Finally, combining Eq. (2), Eq. (3), and Eq. (4), we have that for every $k \in K$

$$|\text{Prob}[P(S^C(k)) = 1] - \text{Prob}[P(S^D(k)) = 1]| \geq \frac{1}{3}k^{-c} + k^{-\omega(1)} + (\frac{1}{3}k^{-c} + k^{-\omega(1)}),$$

contradicting Eq. (1). This concludes the proof of the first part of the lemma.

Towards proving the second part, define the distribution ensembles $T_C \stackrel{\text{def}}{=} \{\text{Time}(S^C(k))\}$ and $T_D \stackrel{\text{def}}{=} \{\text{Time}(S^D(k))\}$. We first argue that T_C, T_D are *statistically* indistinguishable. Otherwise, by a similar argument to the above, there exists an adaptive sampling algorithm S_0 such that $\{\text{Time}(S_0^C(k))\}$ and $\{\text{Time}(S_0^D(k))\}$ are both polynomially-bounded and are statistically distinguishable (i.e., not indistinguishable) from each other. Since the two distribution ensembles have polynomial-size support, their statistical distinguishability implies computational distinguishability, contradicting the assumption that $C \stackrel{\text{c}}{\approx} D$.

Now, let

$$\mathcal{T}(k) = \left\{ t \in \mathcal{N} \;:\; \text{Prob}[\text{Time}(S^D(k)) = t] > 2 \cdot \text{Prob}[\text{Time}(S^C(k)) = t] \right\}.$$

Since $T_C \stackrel{\text{s}}{\approx} T_D$, the event $\text{Time}(S^D(k)) \in \mathcal{T}(k)$ must occur with negligible probability. It remains to show that the expected running time of S^D conditioned on the complement of this event is polynomial. Since $\text{Time}(S^D(k)) \in \mathcal{T}(k)$ occurs with small probability, we may conclude that for all sufficiently large k and $t \notin \mathcal{T}(k)$:

$$\text{Prob}[\text{Time}(S^D(k)) = t \mid \text{Time}(S^D(k)) \notin \mathcal{T}] \leq 2 \cdot \text{Prob}[\text{Time}(S^D(k)) = t]$$

and so

$$\begin{aligned} E[\text{Time}(S^D(k)) \mid \text{Time}(S^D(k)) \notin \mathcal{T}] &\leq 2 \sum_{t \notin \mathcal{T}} t \cdot \text{Prob}[\text{Time}(S^D(k)) = t] \\ &\leq 4 \sum_{t \notin \mathcal{T}} t \cdot \text{Prob}[\text{Time}(S^C(k)) = t] \\ &\leq 4 \cdot E[\text{Time}(S^C(k))] \\ &\leq k^{O(1)} \end{aligned}$$

as required. □

A proof of the following statistical version of the adaptive sampling lemma may proceed similarly to the proof for the computational case.

Lemma 14 (Adaptive Sampling Lemma: Statistical Version)) *Let $C = \{C(k)\}, D = \{D(k)\}$ be two distribution ensembles such that $C \stackrel{\text{s}}{\approx} D$. Let S be an adaptive sampling algorithm which, in addition to its distribution oracle, has access to an (arbitrarily powerful) oracle $\mathcal{A}$. Suppose that $S^{C,\mathcal{A}}$ runs in expected polynomial time. Then:*

1. *$S^{C,\mathcal{A}} \stackrel{\text{s}}{\approx} S^{D,\mathcal{A}}$.*
2. *$S^{D,\mathcal{A}}$ runs in expected polynomial time except with negligible probability.*

The adaptive sampling lemma can be used to analyze the quality of the simulator $\mathcal{S}$ from Section 2.4.1 when given access to imperfect non-adaptive simulator oracles.

Corollary 15 *Suppose that the simulator $\mathcal{S}$ is run with oracle access to computationally (respectively, statistically) good non-adaptive simulators $\tilde{V}_b$ and an expected-polynomial time (resp., unbounded) adaptive adversary $\mathcal{A}$. Moreover, suppose that the size of the adversary structure $\mathcal{B}$ is polynomial in the security parameter. Then:*

1. *Ignoring the running time of $\mathcal{S}$, it produces a computationally (resp., statistically) good emulation of $\mathcal{A}$.*

2. *$\mathcal{S}$ runs in expected polynomial time except with negligible probability.*

Proof: Let S be the simulator $\mathcal{S}$ running an expected polynomial time implementation of $\mathcal{A}$ (resp., with oracle access to $\mathcal{A}$), let C be a concatenation of all perfect non-adaptive simulator oracles $(V_b)_{b\in\mathcal{B}}$, and D a concatenation of all imperfect oracles $(\tilde{V}_b)_{b\in\mathcal{B}}$. Since $|\mathcal{B}|$ is polynomial, we have $C \overset{c}{\approx} D$ (resp., $C \overset{s}{\approx} D$). From an analysis of the simulator $\mathcal{S}$ in the perfect case, we have that: (1) S^C perfectly emulates $\mathcal{A}$; (2) S^C runs in expected polynomial time. Noting that S^D corresponds to the execution of $\mathcal{S}$ with access to the imperfect non-adaptive simulators, the corollary follows by applying the adaptive sampling lemma to S, C, D defined above. $\square$

We stress that the expected running time of $\mathcal{S}$ may be unbounded even if the non-adaptive simulators are arbitrarily close to being perfect. In the rest of the section we describe and analyze a modified simulator $\mathcal{S}'$ which attempts to remedy this situation. While the efficiency and security of $\mathcal{S}$ were analyzed in terms of $|\mathcal{B}|$, the number of possible *sets* the adversary may corrupt, those of $\mathcal{S}'$ will be analyzed in terms of the number of possible *corruption paths* an adversary may take. Formally, let

$$\vec{\mathcal{B}} \overset{\text{def}}{=} \{(b_0, b_1, \ldots, b_i) : 0 \le i \le n, b_0 = \emptyset, |b_j \setminus b_{j-1}| = 1, b_j \in \mathcal{B}\ (1 \le j \le i)\}$$

be the *directed structure* corresponding to $\mathcal{B}$. The simulator $\mathcal{S}'$ will enjoy the following properties. When $|\vec{\mathcal{B}}|$ is polynomial, $\mathcal{S}'$ will output a good emulation of $\mathcal{A}$, similarly to $\mathcal{S}$. However, in this case $\mathcal{S}'$ will be guaranteed to run in expected polynomial time regardless of the quality of the non-adaptive simulators.

Before describing $\mathcal{S}'$, it will be helpful to consider the following modification of $\mathcal{S}$. For any path $\pi \in \vec{\mathcal{B}}$, view v, and adaptive adversary $\mathcal{A}$, we write $v \overset{\text{A}}{\to} \pi$ if the view v leads $\mathcal{A}$ to corrupt all parties in π *in the order prescribed by* π. Now, let $\vec{\mathcal{S}}$ denote a variant of $\mathcal{S}$ which keeps track of an entire path $\pi_i = (b_0, b_1, \ldots, b_i)$ in addition to the currently corrupted set b_i. The condition $v \overset{\text{A}}{\to} b_i$ in step (2a) of $\mathcal{S}$ is replaced by $v \overset{\text{A}}{\to} \pi_i$, and after adding p_{i+1} to b_i in step (2b) to form b_{i+1}, the set b_{i+1} is concatenated to π_i. A slight modification of the analysis from the previous section gives the following:

Lemma 16 *In the case of* perfect *non-adaptive emulation ($V_b \overset{\text{d}}{=} \tilde{V}_b$), the emulation of $\vec{\mathcal{S}}$ is perfect. Moreover, its expected running time is linear in $|\vec{\mathcal{B}}|$.*

We turn to describing the new simulator $\mathcal{S}'$. The underlying idea is the following. Similarly to $\vec{\mathcal{S}}$, the simulator will keep track of the current corruption path. However, before extending the current path π_i, it will try to obtain "significant statistical evidence" that the probability of the event $\tilde{V}_{b_i} \overset{\text{A}}{\to} \pi_i$ is not much smaller than that of $\mathcal{S}'$ arriving at π_i. This is done by repeatedly rerunning a "lighter" version of the simulation history in parallel to sampling from $\tilde{V}_{b_i}$. (Rerunning the entire simulation from scratch will be too costly, and will only allow to efficiently handle a constant number of parties. The lighter version assumes that π_i has already been verified to be good.) Unless evidence as above is obtained, the simulation terminates. This careful path extension policy will guarantee that the contribution of each path to the expected running time is small.

The simulator $\mathcal{S}'$ is described in detail below. Somewhat abusing notation, given a path π_i we will denote by $\pi_{i'}$, for $i' < i$, the length-i' prefix of π_i.

Algorithm of $\mathcal{S}'$:

1. Initialization:
   Let b_0 be the empty set and $\pi_0 = (b_0)$ be the initial path (π_i is the currently corrupted path thereafter).

2. For $i = 0, 1, 2, \ldots$ do:

   (a) Initialize counters c, c' to 0;
   Given the currently corrupted path π_i:
   Repeat

   i. Call procedure $\mathsf{Rerun}(\pi_i)$, defined below; if it returns *success* increment the counter c.

   ii. Sample $v' \stackrel{\text{R}}{\leftarrow} \tilde{V}_{b_i}$; if $v' \stackrel{\text{A}}{\rightarrow} \pi_i$, increment the counter c'.

   Until $c = k$

   (b) If $c' < k/2$, terminate the simulation and output *fail*;

   (* This will only happen with negligible probability, and should warn us that it is not a good idea to proceed. *)

   (c) Using the view v' which led to first incrementing c' in step (a.ii), run $\mathcal{A}$ to determine the next party p_{i+1} to corrupt. If $\mathcal{A}$ decides to terminate, terminate the simulation outputting v'. Otherwise, let b_{i+1} be b_i plus p_{i+1}, let π_{i+1} be the path obtained by concatenating b_{i+1} to the end of π_i, and iterate to next the i.

Procedure $\mathsf{Rerun}(\pi_i = (b_0, ..., b_i))$
For $i' = 0$ to $i - 1$ do
  Sample $v' \stackrel{\text{R}}{\leftarrow} \tilde{V}_{b_{i'}}$ until $v' \stackrel{\text{A}}{\rightarrow} \pi_{i'}$;
  Run $\mathcal{A}$ to determine the next party to corrupt;
  If this party is inconsistent with $\pi_{i'+1}$, return *fail*;
Return *success*.

Analysis of $\mathcal{S}'$. We begin by analyzing the running time. Let $\#\text{paths}_i$ denote the number of paths of length i in $\vec{\mathcal{B}}$, $\tilde{\Pi}'_i$ a random variable taking the value of the corrupted path in the i-th iteration of $\mathcal{S}'$, and $\tilde{T}'_i$ the running time of the i-th iteration. It will be helpful to compare the execution of $\mathcal{S}'$ on the imperfect distributions $\tilde{V}_b$ to the execution of its simpler variant $\vec{\mathcal{S}}$ described above on the same distributions. We let $\tilde{\Pi}_i$ denote a random variable taking the value of the corrupted path in the i-th iteration of $\vec{\mathcal{S}}$.

We will show that the expected running time of the i-th iteration of $\mathcal{S}'$ is polynomial in k and $\#\text{paths}_i$. Conditioning on the path π_i, we have:

$$\begin{aligned} E[\tilde{T}'_i] &= \sum_{\pi_i} E[\tilde{T}'_i \mid \tilde{\Pi}'_i = \pi_i] \cdot \text{Prob}[\tilde{\Pi}'_i = \pi_i] \\ &= \sum_{\pi_i} k \cdot E[\#\text{calls to } \mathsf{Rerun}(\pi_i) \text{ until } \textit{success}] \cdot E[\text{Time}(\mathsf{Rerun}(\pi_i))] \cdot \text{Prob}[\tilde{\Pi}'_i = \pi_i] \quad (5) \end{aligned}$$

We bound the above expression using the following lemmas.

Lemma 17 $\text{Prob}[\tilde{\Pi}'_i = \pi_i] \leq \text{Prob}[\tilde{\Pi}_i = \pi_i]$.

Proof: It is clear from the description of $\mathcal{S}'$ that unless it prematurely terminates, it produces the same path distribution as $\vec{\mathcal{S}}$. $\square$

Lemma 18 *The expected number of calls to* $\mathsf{Rerun}(\pi_i)$ *until it returns "success" is* $1/\text{Prob}[\tilde{\Pi}_i = \pi_i]$.

Proof: Procedure $\mathsf{Rerun}(\pi_i)$ emulates $\vec{\mathcal{S}}$, truncating its execution only when it is clear that it will not lead to π_i. We may therefore conclude that:

$$\text{Prob}[\mathsf{Rerun}(\pi_i) = \textit{success}] = \text{Prob}[\tilde{\Pi}_i = \pi_i]$$

from which the lemma follows. $\square$

Lemma 19 $E[\text{Time}(\mathsf{Rerun}(\pi_i))] = \sum_{i'<i} \text{Prob}[\tilde{\Pi}_i = \pi_{i'}]/\text{Prob}[\tilde{V}_{b_{i'}} \xrightarrow{\text{A}} \pi_{i'}]$

Proof: For each $0 \leq i' < i$, the probability that $\mathsf{Rerun}(\pi_i)$ gets to the stage where it samples $\tilde{V}_{b_{i'}}$ (until it is consistent with $\pi_{i'}$) is exactly $\text{Prob}[\tilde{\Pi}_{i'} = \pi_{i'}]$. The expected contribution of the i'-th segment to the run time of $\mathsf{Rerun}(\pi_i)$ is therefore $\text{Prob}[\tilde{\Pi}_{i'} = \pi_{i'}]/\text{Prob}[\tilde{V}_{b_{i'}} \xrightarrow{\text{A}} \pi_{i'}]$. $\square$

Lemma 20 *For any path* π_i*,*

$$\text{Prob}[\tilde{\Pi}'_i = \pi_i] \cdot (1/\text{Prob}[\tilde{\Pi}_i = \pi_i]) \cdot E[\text{Time}(\mathsf{Rerun}(\pi_i))] = O(i)$$

Proof: First note that if either $\text{Prob}[\tilde{\Pi}_i = \pi_i] = 0$ or $E[\text{Time}(\mathsf{Rerun}(\pi_i))]$ is unbounded, then $\text{Prob}[\tilde{\Pi}'_i = \pi_i] = 0$ (since a path is extended only after there is evidence that it is reachable by $\vec{\mathcal{S}}$).

From Lemma 17, $\text{Prob}[\tilde{\Pi}'_i = \pi_i] \cdot (1/\text{Prob}[\tilde{\Pi}_i = \pi_i]) \leq 1$. It therefore suffices to show that, for sufficiently large m, k, if $E[\text{Time}(\mathsf{Rerun}(\pi_i))] > mi$ then $\text{Prob}[\tilde{\Pi}'_i = \pi_i] < \text{Prob}[\tilde{\Pi}_i = \pi_i]/m$. This will imply that the survival probability of a bad path is (at most) inverse proportional to the time penalty it incurs on Rerun.

Suppose that $E[\text{Time}(\mathsf{Rerun}(\pi_i))] > mi$. Then, by Lemma 19, there is $i' < i$ such that

$$\text{Prob}[\tilde{\Pi}_{i'} = \pi_{i'}]/\text{Prob}[\tilde{V}_{b_{i'}} \xrightarrow{\text{A}} \pi_{i'}] > m. \tag{6}$$

To analyze the probability of passing the i'-th test in step (2b), let $p_1 = \text{Prob}[\tilde{\Pi}_{i'} = \pi_{i'}]$ and $p_2 = \text{Prob}[\tilde{V}_{b_{i'}} \xrightarrow{\text{A}} \pi_{i'}]$. By Eq. (6), $p_2 < p_1/m$. In the following we show that when flipping in parallel two coins with success probabilities p_1, p_2 such that $p_2 < p_1/m$, the probability that the p_2-trials will have $k/2$ successes before the p_1-trials have k successes is less than $1/m$ (for sufficiently large m, k). We refer to the above event as a success of the test. Let $s = k\sqrt{m}/p_1$ be a number of trials. The probability of the test succeeding is bounded by the probability that either there are less than k successes in s independent p_1-trials, or there are at least $k/2$ successes in s independent p_2-trials (for otherwise the test clearly fails). We show that both of these probabilities are asymptotically smaller than $1/m$. The first experiment has expectation $\mu = k\sqrt{m}$ and a relative deviation greater than a constant. The tail probability is bounded by $F^-(\delta, \mu) < e^{-\Omega(\mu)}$ which is $o(1/m)$. The second experiment has expectation $\mu < k/\sqrt{m}$ and relative deviation $\delta = \Omega(\sqrt{m})$. The tail probability is bounded by $F^+(\delta, \mu) < (e/(1+\delta))^{(1+\delta)\mu} = (1/\sqrt{m})^{\Omega(k)}$. For k greater than some constant, this probability is again $o(1/m)$. We may conclude that for m, k greater than some absolute constant, $\text{Prob}[\tilde{\Pi}'_i = \pi_i] < 1/m \cdot \text{Prob}[\tilde{\Pi}_i = \pi_i]$ as required. $\square$

We are now ready to bound the expected running time of the i-th iteration of $\mathcal{S}'$.

Lemma 21 $E[\tilde{T}_i'] = O(k \cdot i \cdot \#\text{paths}_i)$.

Proof: Substituting Lemma 18 in Eq. (5) and applying Lemma 20, we get:

$$E[\tilde{T}_i'] = O(\sum_{\pi_i} k \cdot i) = O(k \cdot i \cdot \#\text{paths}_i).$$

$\square$

From the last lemma we may conclude the following:

Claim 22 *Regardless of the non-adaptive emulation quality, the expected running time of $\mathcal{S}'$ is polynomial in $|\vec{\mathcal{B}}|$ and the security parameter.*

We turn to analyze the emulation quality of $\mathcal{S}'$.

Claim 23 *Suppose that $|\vec{\mathcal{B}}|$ is polynomial in k, and that the non-adaptive simulators $\tilde{V}_b$ are computationally (respectively, statistically) good. Then the simulator $\mathcal{S}'$ produces a computationally (resp., statistically) good output.*

Proof: It follows from Lemma 16 and the adaptive sampling lemma that when $|\vec{\mathcal{B}}|$ is polynomial, the output of $\vec{\mathcal{S}}$ is computationally (resp., statistically) good. We will argue that the output produced by $\mathcal{S}'$ is statistically close to that of $\vec{\mathcal{S}}$. Since an execution of $\mathcal{S}'$ produces the same output distribution as $\vec{\mathcal{S}}$ except for the event that $\mathcal{S}'$ terminates prematurely and outputs *fail*, it suffices to show that this event occurs with negligible probability. Consider a termination test performed during the execution of $\mathcal{S}'$ with current path π_i, and let p_1, p_2 be the two relevant probabilities. That is, $p_1(\pi_i) = \text{Prob}[\tilde{\Pi}_i = \pi_i]$ and $p_2(\pi_i) = \text{Prob}[\tilde{V}_{b_i} \xrightarrow{\text{A}} \pi_i]$. The difference $|p_1 - p_2|$ must be bounded by some negligible function $\epsilon(k)$. Indeed, both probabilities are negligibly close to the probability of $\vec{\mathcal{S}}$ arriving at π_i when given access to perfect distributions V_b. Now, call π_i *good* if $p_1(\pi_i) > 3\epsilon(k)$ and *bad* otherwise. Clearly, since by Lemma 17 the probability of $\mathcal{S}'$ arriving at π_i is at most $p_1(\pi_i)$, the probability of $\mathcal{S}'$ arriving at *any* bad path during its execution is negligible. Finally, since for any good path π_i we have $p_2(\pi_i) > \frac{2}{3}p_1(\pi_i)$, the probability of premature termination at a good path π_i is negligible in k (as the probability of having k successes of p_1-trials before $k/2$ successes of p_2-trials). $\square$

Noting that $|\vec{\mathcal{B}}|$ is polynomial when $n = O(\log k / \log\log k)$, the results of this section can be summarized by the following theorem.

Theorem 24 *For function evaluation protocols with passive adversary and $n = O(\log k / \log\log k)$ parties, adaptive and non-adaptive security are equivalent under* any *notion of security. Moreover, with a relaxed notion of efficiency allowing a negligible failure probability, the bound on the number of parties can be improved to $n = O(\log k)$.*

We remark that Theorem 24 is essentially tight in the following sense: when $n = \omega(\log k)$, adaptive security is separated from non-adaptive security even if the adaptive simulator is allowed to be computationally unbounded.

2.5 Equivalence for passive adversaries and IT security

Claim 10 immediately implies the following:

Theorem 25 *For function evaluation protocols with passive adversary and perfect information-theoretic security, adaptive and non-adaptive security are equivalent.*

Note that there is no dependence on the number of players in the above theorem.

3 Adaptivity vs. Non-adaptivity in the definition of Dodis-Micali-Rogaway

3.1 Review of the definition

For completeness, we start with a very short summary of the definition of secure multiparty computation by Micali and Rogaway, more specifically the version that appears in the paper by Dodis and Micali [DM00]. For additional details, please refer to [DM00].

We have n players, each player P_i starts with a value x_i as input and auxiliary input a_i. We set $a = (a_1, ...a_n); x = (x_1, ..., x_n)$.

To satisfy the definition, a protocol π must have a fixed *committal round* CR, the point at which inputs become uniquely defined, as follows: The *traffic* of a player consists of all messages he sends and receives. π must specify *input- and output functions* that map traffic to input- and output values for the function f computed. The *effective inputs* $\hat{x}_1^\pi, ..., \hat{x}_n^\pi$ are determined by applying the input functions to the traffic of each player up to and including CR. So these values are the ones that players "commit to" as their inputs. The *effective outputs* $\hat{y}_1^\pi, ..., \hat{y}_n^\pi$ are determined by applying the output functions to the entire traffic of each player.

For adversary $\mathcal{A}$ (taking random input and auxiliary input α), random variable $View(\mathcal{A}, \pi)$ is the view of $\mathcal{A}$ when attacking π. We define:

$$History(\mathcal{A}, \pi) = View(\mathcal{A}, \pi), \hat{x}^\pi, \hat{y}^\pi$$

The way $\mathcal{A}$ interacts with the protocol is as follows: in each round, $\mathcal{A}$ sees all messages from honest players in this round. He may then issue some number of corruption requests adaptively, and only then must he generate the messages to be sent to the remaining honest players.

The definition calls for existence of a simulator $\mathcal{S}$ which may depend on the protocol in question, but not the adversary. The goal of the simulator is to sample the distribution of $History(\mathcal{A}, \pi)$. To do so, it is allowed to interact with $\mathcal{A}$, but it is restricted to one-pass black-box simulation with no bound on the simulator's running time, i.e., $\mathcal{A}$ interacts with $\mathcal{S}$ in the same way it interacts with π, and $\mathcal{S}$ is not allowed to rewind $\mathcal{A}$. The simulator $\mathcal{S}$ gets an oracle O as help (where the oracle knows x, a):

- If P_j is corrupted before CR, the oracle sends x_j, a_j to $\mathcal{S}$.
- At CR, $\mathcal{S}$ applies the input functions to the view of $\mathcal{A}$ it generated so far to get effective inputs of corrupted players $\hat{x}_j^{\mathcal{S}}$. It sends these values to O. O computes the function choosing random input r and using as input the values it got from $\mathcal{S}$ for corrupted players and the real x_j's for honest players. The result is $\hat{y}^{\mathcal{S}} = (\hat{y}_1^{\mathcal{S}}, ..., \hat{y}_n^{\mathcal{S}})$. O sends the results for corrupted players back to $\mathcal{S}$.
- If P_j is corrupted in or after CR, O sends $x_j, a_j, \hat{y}_j$ to $\mathcal{S}$.

The random variable $View(\mathcal{A}, \mathcal{S})$ is the view of $\mathcal{A}$ when interacting with $\mathcal{S}$. The effective inputs $\hat{x}^{\mathcal{S}}$ are as defined above, i.e., if a P_j is corrupted before CR, then his effective input $\hat{x}_j^{\mathcal{S}}$ is determined by the input function on his traffic, else $\hat{x}_j = x_j$. The effective outputs $\hat{y}^{\mathcal{S}}$ are defined as what the oracle outputs, i.e. $\hat{y}^{\mathcal{S}} = f(\hat{x}^{\mathcal{S}}, r)$.

$$History(\mathcal{A}, \mathcal{S}) = View(\mathcal{A}, \mathcal{S}), \hat{x}^{\mathcal{S}}, \hat{y}^{\mathcal{S}}$$

We can now define that π computes f securely iff there exists a simulator $\mathcal{S}$ such that for every adversary $\mathcal{A}$, and every x, a, α,

$$History(\mathcal{A}, \mathcal{S}) \equiv History(\mathcal{A}, \pi)$$

i.e., the two variables have identical distributions.

At first sight it may seem strange that the definition does not explicitly require that players who are honest up to CR actually commit to their real inputs, or that players who are never corrupted really receive "correct" values. But this follows from the definition:

Lemma 26 *If π computes f securely, then the input- and output functions are such that if P_j remains honest up to CR, then $\hat{x}^{\pi}_j = x_j$. And if P_j is never corrupted, then $\hat{y}^{\pi}_j$ is the j'th component of $f(\hat{x}^{\pi}, r)$, for a random r.*

Proof: Consider an adversary $\mathcal{A}_j$ that never corrupts P_j. Then the first claim follows from $x_j = \hat{x}^{\mathcal{S}}_j$ and $History(\mathcal{A}_j, \mathcal{S}) \equiv History(\mathcal{A}_j, \pi)$. The second follows from $History(\mathcal{A}_j, \mathcal{S}) \equiv History(\mathcal{A}_j, \pi)$ and the fact that the correlation $\hat{y}^{\mathcal{S}}_j = f(\hat{x}^{\mathcal{S}}, r)_j$ between $\hat{x}^{\mathcal{S}}$ and $\hat{y}^{\mathcal{S}}$ always holds. □

Note that this lemma continues to hold, even if we only assume static security.

3.2 Equivalence of adaptive and non-adaptive security

It turns out to be convenient in the following to define the notion of a *partial history*, of an adversary $\mathcal{A}$ that either attacks π or interacts with a simulator. A partial history constrains the history up to a point at the start of, or inside round j for some j. That is, round $j-1$ has been completed but round j has not. If $j \leq CR$, then such a partial history consists of a view of the adversary up to round j, and possibly including some part of round j. If $j > CR$, but the protocol is not finished, a partial history consists of a partial view of $\mathcal{A}$ as described before plus the effective inputs. Finally, if the protocol is finished at round j, the history is as defined earlier: complete view of $\mathcal{A}$ plus the effective inputs and outputs.

Note that if $\mathcal{S}$ is such that $History(\mathcal{A}, \pi) \equiv History(\mathcal{A}, \mathcal{S})$, then trivially it also holds that the partial histories of $\mathcal{A}, \pi$ and of $\mathcal{A}, \mathcal{S}$ ending at any point are identically distributed. Moreover, since $\mathcal{S}$ never rewinds, the value of the partial history of $\mathcal{A}, \mathcal{S}$ at some point in time will be fixed as soon as $\mathcal{S}$ has reached that point in the simulation.

We can then slightly extend the actions an adversary can take: a *halting adversary* $\mathcal{A}'$ is one that interacts with protocol or simulator in the normal way, but may at any point output a special halting symbol and then stop. In the simulation, if the simulator receives such a symbol, the simulation process also stops. The histories $History(\mathcal{A}', \pi), History(\mathcal{A}', \mathcal{S})$ are defined to be whatever the partial history is at the point when $\mathcal{A}$ stops.

Trivially protocol π is secure in the above definition if and only if, for any halting adversary $\mathcal{A}'$, $History(\mathcal{A}', \pi) \equiv History(\mathcal{A}', \mathcal{S})$. Note that this extension of the definition does not capture any new security properties, it is simply a "hack" that turns out to be convenient in the proof of the following theorem.

In the following we assume that there exists a static (non-adaptive) simulator $\mathcal{S}_0$ such that for every *static* adversary $\mathcal{A}_0$, and every x, a, α,

$$History(\mathcal{A}_0, \mathcal{S}_0) \equiv History(\mathcal{A}_0, \pi)$$

We want to make a general simulator $\mathcal{S}$ that shows that π in fact is secure against any adaptive adversary $\mathcal{A}$, in other words, we claim

Theorem 27 *Adaptive and non-adaptive security are equivalent under the Dodis-Micali-Rogaway definition.*

To this end, we construct a static adversary $\mathcal{A}_B$ (of the halting type), for every set B that it is possible for $\mathcal{A}$ to corrupt. $\mathcal{A}_B$ plays the following strategy, where we assume that $\mathcal{A}_B$ is given black-box access to (adaptive) adversary $\mathcal{A}$, running with some random and auxiliary inputs $r_{\mathcal{A}}$ and α[6]:

Algorithm of $\mathcal{A}_B$

1. Corrupt the set B initially. For each $P_j \in B$, initialize the *honest* algorithm for P_j, using as input x_j, a_j learnt from corrupting P_j (and fresh random input).
2. Start executing the protocol, initially letting the players in B play honestly, but keeping a record of their views. At the same time, start running $\mathcal{A}$.
3. Whenever $\mathcal{A}$ issues a corruption request for player P_j, we do the following: if $P_j \in B$, we provide $\mathcal{A}$ with x_j, a_j and all internal data of P_j. After this point, all messages for P_j are sent to $\mathcal{A}$, and we let $\mathcal{A}$ decide the actions of P_j from this point. If $P_j \notin B$, output a halt symbol and stop.

The idea in the following is to use the assumed ability (by $\mathcal{S}_0$) to generate histories of $\mathcal{A}_B$ attacking π to generate histories of $\mathcal{A}$ attacking π. Note that in any round of π, the current history of $\mathcal{A}_B$ contains both the (so far honest) history of 0 or more players that $\mathcal{A}$ has not yet corrupted, plus the view so far of $\mathcal{A}$. So for any such (partial) history u of $\mathcal{A}_B$, we let $Aview(u)$ be the view of $\mathcal{A}$ that can be extracted from u in the natural way.

In particular, if u is a history of $\mathcal{A}_B$ that ends after the final round of the protocol, then $Aview(u)$ is a complete view of $\mathcal{A}$ where $\mathcal{A}$ corrupted only players in B, whereas if u ends before the protocol is complete, $Aview(u)$ ends in the some round where $\mathcal{A}$ requested to corrupt some player outside B.

We are now ready to describe the algorithm of $\mathcal{S}$. We assume $\mathcal{S}$ interacts with an adaptive adversary $\mathcal{A}$ who starts from some random input and is given some arbitrary auxiliary input α. Also we are given an oracle O, that knows the actual inputs x and makes some random choice r when computing the function.

Algorithm of $\mathcal{S}$:

1. Initialization:
   Set $B = \emptyset$. B will contain the current set of corrupted parties
   Set v = the empty string. v will contain the (simulated) view of $\mathcal{A}$ so far
   Set a_B, x_B, and $\hat{y}_B^{\mathcal{S}}$ = the empty string, these variables will contain the inputs, auxiliary inputs, and effective outputs of players in B.
2. The purpose of this step is to obtain a random sample of the output of $\mathcal{S}_0$ when interacting with $\mathcal{A}_B$, conditioned on the event that the history u produced, is such that v is a prefix of $Aview(u)$.

   We do this by repeatedly executing $\mathcal{S}_0, \mathcal{A}_B$ until a useful u is obtained (this make take more than polynomial time, but we consider unbounded simulation here). We will not always run $\mathcal{S}_0, \mathcal{A}_B$ until they halt, in cases where it is clear that there is no hope of getting a useful u, we will stop immediately, as described below. Note that, in order to execute $\mathcal{S}_0$, we need to

[6] We could also have given $r_{\mathcal{A}}, \alpha$ as input to $\mathcal{A}_B$, letting it simulate the algorithm of $\mathcal{A}$, but the set-up we use is more convenient in the following.

provide access it needs to an oracle (which we call O_0 to distinguish from the oracle O that $\mathcal{S}$ uses), also $\mathcal{A}_B$ needs black-box access to $\mathcal{A}$. We describe how to emulate O_0, As below. Thus, the following subroutine for sampling $\mathcal{S}_0, \mathcal{A}_B$ is repeated until a history u is produced such that v is a prefix of $Aview(u)$.

(a) Initialize the algorithms of $\mathcal{A}_B$ and $\mathcal{S}_0$ using random inputs chosen uniformly among those we have not used before. (but note that we do not restart $\mathcal{A}$). Send x_B, a_B to $\mathcal{S}_0$ (on behalf of O_0). The variable u will at all times hold the current history of $\mathcal{S}_0, \mathcal{A}_B$ we have produced so far. It is initially empty, and we maintain the following invariant: either v is a prefix of $Aview(u)$ (which includes the case $v = Aview(u)$), or $Aview(u)$ is a proper prefix of v.

(b) Do the following for each round:
Get messages for players in B from $\mathcal{S}_0$ and send these to $\mathcal{A}_B$. Now we want to simply run the algorithm of $\mathcal{A}_B$ to compute the actions in this round of players in B. But recall that $\mathcal{A}_B$ needs black-box access to $\mathcal{A}$. At this point, however, $\mathcal{A}$ thinks that it is in the middle of an attack on the protocol and we are not allowed to rewind. Fortunately, since we are only interested in generating views with v as prefix, we can do something else:

- If $Aview(u)$ is a proper prefix of v, look at the set of messages that $\mathcal{A}_B$ sends to $\mathcal{A}$ at this point. Check if this equals the set of messages sent to $\mathcal{A}$ at this point according to the view v. If so, we take the response of $\mathcal{A}$ from v and send it to $\mathcal{A}_B$. This response may be a corruption request, in which case we check if the reaction to this from $\mathcal{A}_B$ is consistent with v, and continue to process the next response from $\mathcal{A}$ (again taken from v). If any inconsistencies with v are discovered, we stop the current run of $(\mathcal{A}_B, \mathcal{S}_0)$, and go back to step 2a (since it is then clear that the history u we are generating will not be consistent with v). Otherwise, we keep going, extending the contents of u until *either* the interaction between $\mathcal{A}_B$ and $\mathcal{A}$ in this round is finished, *or* we reach a point where $Aview(u) = v$ (in the latter case continue with the next item).
- If we are not finished with the current round and if v is a prefix of $Aview(u)$, send the messages generated by $\mathcal{A}_B$ at this point to $\mathcal{A}$, and let $\mathcal{A}_B$ conduct its interaction directly with $\mathcal{A}$ starting from whatever state $\mathcal{A}$ is in at this point. Note that this may cause $\mathcal{A}_B$ (and hence $\mathcal{S}_0$) to halt if $\mathcal{A}$ tries to corrupt a player outside B.

We reach this point if $\mathcal{S}_0, \mathcal{A}_B$ completed the current round without halting. If we are in the CR at this point, we must also emulate the behavior of O_0 in the CR. We do as follows:

- If we have not queried O in CR before, send the effective inputs produced by $\mathcal{S}_0$ and send them to O to get $\hat{y}_j^{\mathcal{S}}, P_j \in B$, we save these values in $\hat{y}_B^{\mathcal{S}}$ and also send them to $\mathcal{S}_0$.
- If we have queried O before, send the current value of $\hat{y}_B^{\mathcal{S}}$ to $\mathcal{S}_0$.

3. At this stage, the previous step has produced a history u of $\mathcal{A}_B$, such that $w = Aview(u)$ has v as a prefix. Now, if w extends all the way to the end of the protocol, we output w and stop. Otherwise, go to next step.

4. If we reach this step, w ends prematurely because $\mathcal{A}$ requested to corrupt a party $P_j \notin B$. Then get x_j, a_j and possibly $\hat{y}_j^{\mathcal{S}}$ from O
set $B = B \cup \{P_j\}$

set $v = w$
set $a_B = a \cup \{a_j\}$, $x_B = x \cup \{x_j\}$ and $\hat{y}_B^{\mathcal{S}} = \hat{y}_B^{\mathcal{S}} \cup \hat{y}_j^{\mathcal{S}}$.

5. Go to step 2.

To show that $\mathcal{S}$ works as required it is clearly enough to show the following

Claim: For any fixed random and auxiliary input for $\mathcal{A}$ and for any input x and random choice r for the oracle O, the algorithm for $\mathcal{S}$ terminates and conditioned on the data we fixed, $History(\mathcal{A}, \pi) \equiv History(\mathcal{A}, \mathcal{S})$.

In the entire following discussion, we assume that the data mentioned in the claim are fixed. The fact that $\mathcal{S}$ terminates is one consequence of the following lemma:

Lemma 28 *Fix any $B, v, a_B, x_B, \hat{y}_B^{\mathcal{S}}$ that may occur as values at the start of some iteration i of the algorithm of $\mathcal{S}$. Then the following iteration terminates and produces a value u of $History(\mathcal{A}_B, \mathcal{S}_0)$, where the distribution of u equals that of the history of $\mathcal{S}_0$ when interacting with $\mathcal{A}_B$ conditioned on the values $B, v, a_B, x_B, \hat{y}_B^{\mathcal{S}}$, and on v being a prefix of $Aview(u)$.*

Proof: If $i = 1$, the lemma is trivially true: in this case all the variables $B, v, a_B, x_B, \hat{y}_B^{\mathcal{S}}$ are empty, so we are not conditioning on anything, and $\mathcal{A}_B$ and $\mathcal{S}_0$ are executed according to their respective algorithms with black-box access to correct data. Termination is also trivial because the first attempt to run $\mathcal{A}_B$ and $\mathcal{S}_0$ will always result in a useful u.

For $i > 0$, let B' be the value of B in iteration $i-1$. Since iteration i is executed, we may assume that the view v ends by $\mathcal{A}$ corrupting a player $P_j \notin B'$. The view v was produced in the previous iteration by $\mathcal{S}_0$ interacting with $\mathcal{A}_{B'}$. Since $\mathcal{S}_0$ is a perfect simulator, there exists an execution of the real protocol, where $\mathcal{A}$'s view is v and where P_j has some honest view v_j until the end of v. Then by definition of $\mathcal{A}_B$, a possible (partial) view of $\mathcal{A}_B$ is one where its interaction with $\mathcal{A}$ results in view v and honest view v_j for P_j. And again, since $\mathcal{S}_0$ is a perfect simulator, it must be possible to generate these same views by having $\mathcal{A}_B$ interact with $\mathcal{S}_0$. Since the algorithm of $\mathcal{S}$ in the i'th iteration searches exhaustively through the random inputs of $\mathcal{S}_0, \mathcal{A}_B$, a history u consistent with v will eventually be found.

Finally, for the claim on the distribution of u, note that the algorithm for the i'th iteration chooses uniformly among those random inputs for $\mathcal{S}_0, \mathcal{A}_B$ that will produce a u consistent with v. The claim therefore follows if we show that the data obtained from our simulated black-box access to $\mathcal{A}$ and our simulation of oracle O_0 are distributed as in a normal execution. This is clear for the black-box access to $\mathcal{A}$ because we just force the output view for $\mathcal{A}$ to have v as prefix, and otherwise query the real adversary $\mathcal{A}$. For the simulation of O_0, we split in the two cases considered also in the algorithm of $\mathcal{S}$, according to what the situation is when $\mathcal{S}_0$ queries O_0:

- If we have not queried O in CR before, it is clear that the view v must end in or before CR. So therefore, the effective inputs supplied by $\mathcal{S}_0$ is a random set of values as $\mathcal{S}_0, \mathcal{A}_B$ would choose them, conditioned on B, v, a_B, x_B. Hence the results are also correctly distributed because we obtain them from O.

- If we have queried O before, Suppose we are doing iteration i currently, and suppose we queried O in iteration i'. Let $B' \subset B$ be the corrupted set in iteration i'.

  Note that we cannot have $i = i'$: the first time we queried O we must have had the current v as a proper prefix of the current view for $\mathcal{A}$ since the simulation is one-pass. This guarantees that we finished that run of $\mathcal{S}_0, \mathcal{A}_B$ successfully extending v, and so we would not need to run $\mathcal{S}_0, \mathcal{A}_B$ again in the same iteration.

So $i' < i$. Let v' be the view output by iteration i', this view must of course extend beyond the CR, and must determine some effective inputs $\hat{x}'$, namely the input functions applied to v' for players in B' and the real inputs for the other players. Moreover, v determines the same effective inputs $\hat{x}'$: v' is a prefix of v so the input functions for players in B' return the same results on v as on v'. For players in $B \setminus B'$, v contains the honest history of these players up to and including the CR, so the input functions for these players return the real inputs, by Lemma 26

Let u be the current history produced by $\mathcal{S}_0$ when the query to O_0 is made. Since u is consistent with v, u also determines the same input vector $\hat{x}'$, again by applying the input functions. Hence the inputs on which the trusted party would compute the function are exactly the same as the ones it used in iteration i', so it is correct to return the results we already know.

□

Lemma 29 *Fix any corruptible set B. Let $Distr_B$ be the distribution obtained from the distribution of $History(\mathcal{A}, F)$ by truncating every history such that it ends at the first point where $\mathcal{A}$ corrupts a player not in B (no truncation if $\mathcal{A}$ never corrupts a player outside B). Then the distribution of $History(\mathcal{A}_B, \mathcal{S}_0)$ is $Distr_B$.*

Proof: Follows immediately from $History(\mathcal{A}_B, \mathcal{S}_0) \equiv History(\mathcal{A}_B, \pi)$ and by definition of $\mathcal{A}_B$. □

We now return to the proof of the claim above. Let $Distr_i$ be the distribution obtained from the distribution of $History(\mathcal{A}, \pi)$ by truncating every history such that it ends at the point where $\mathcal{A}$ corrupts the i'th player (no truncation if $\mathcal{A}$ corrupts less than i players).

We will show by induction on i that running the algorithm for $\mathcal{S}$ for at most i iterations produces a history with $Distr_i$ as distribution, for any i. The claim then follows because $Distr_{n+1}$ is the distribution of $History(\mathcal{A}, \pi)$, since it cannot corrupt more than the total number of players.

The basis of the reduction ($i = 1$) follows immediately from the two lemmas with $B = \emptyset$, and $v = a_B = x_B = \hat{y}_B^{\mathcal{S}} =$ the empty string (note that $Distr_1 = Distr_\emptyset$).

So consider the induction step for some $i > 1$. By the induction hypothesis, the cases where $\mathcal{S}$ halts after $i - 1$ steps produce with the right distribution those histories where $\mathcal{A}$ completes the protocol having corrupted at most $i - 2$ players. In the cases where the i'th iteration is executed, the induction hypothesis also implies that the values of $B, v, a_B, x_B, \hat{y}_B^{\mathcal{S}}$ we have going into the i'th iteration are distributed exactly as they would be in a real protocol execution in a case where $\mathcal{A}$ has just corrupted the $i - 1$'th player.

We therefore only have to show that this next iteration of $\mathcal{S}$ will produce a history that is distributed according to $Distr_i$, conditioned on the event that v is a prefix of the view of $\mathcal{A}$ (Note that v determines the values of $B, a_B, x_B, \hat{y}_B^{\mathcal{S}}$). Let us call this distribution $Distr_i(v)$.

Similarly, let $Distr_B(v)$ be the distribution obtained by starting from $Distr_B$ and conditioning on the event that v is a prefix of $\mathcal{A}$'s view. It is now clear that $Distr_i(v) = Distr_B(v)$ – namely both equal the distribution over (partial) histories in which $\mathcal{A}$'s view has v as a prefix and continue until $\mathcal{A}$ corrupts the next player or completes the protocol.

Now, by Lemma 29, $Distr_B(v)$ is the distribution produced by $\mathcal{S}_0$ interacting with $\mathcal{A}_B$ when we condition on v. Finally by Lemma 28, this in turn equals the distribution produced by the i'th iteration of $\mathcal{S}$, when that iteration starts from $B, v, a_B, x_B, \hat{y}_B^{\mathcal{S}}$.

References

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

[B97] D. Beaver, "Plug and Play Encryption", CRYPTO 97.

[BH92] D. Beaver and S. Haber, "Cryptographic Protocols Provably secure Against Dynamic Adversaries", *Eurocrypt,* 1992.

[BGW88] M. Ben-Or, S. Goldwasser and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation", *20th Symposium on Theory of Computing (STOC),* ACM, 1988, pp. 1-10.

[C00] R. Canetti, "Security and Composition of Multiparty Cryptographic Protocols", *Journal of Cryptology,* Vol. 13, No. 1, Winter 2000. On-line version at http://philby.ucsd.edu/cryptolib/1998/98-18.html.

[C00a] R. Canetti, "A unified framework for analyzing security of Protocols", manuscript, 2000. Available at http://eprint.iacr.org/2000/067.

[CDDIM01] R. Canetti, I. Damgaard, S. Dziembowski, Y. Ishai and T. Malkin, "On adaptive vs. non-adaptive security of multiparty protocols", http://eprint.iacr.org/2001.

[CFGN96] R. Canetti, U. Feige, O. Goldreich and M. Naor, "Adaptively Secure Computation", *28th Symposium on Theory of Computing (STOC),* ACM, 1996. Fuller version in MIT-LCS-TR #682, 1996.

[CDM98] R.Cramer, I.Damgaard and U.Maurer: *General Secure Multiparty Computation from Any Linear Secret-Sharing Scheme*, EuroCrypt 2000.

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

[CS98] R. Cramer and V. Shoup, "A paractical public-key cryptosystem provably secure against adaptive chosen ciphertext attack", *CRYPTO '98,* 1998.

[DN00] I. Damgaard and J. Nielsen, "Improved non-committing encryption schemes based on a general complexity assumption", CRYPTO 2000.

[DM00] Y. Dodis and S. Micali, "Parallel Reducibility for Information-Theoretically Secure Computation", CRYPTO 2000.

[DDN91] D. Dolev, C. Dwork and M. Naor, "Non-malleable cryptography", SICOMP, to appear. Preliminary version in STOC 91.

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

[GMW87] O. Goldreich, S. Micali and A. Wigderson, "How to Play any Mental Game", *19th Symposium on Theory of Computing (STOC),* ACM, 1987, pp. 218-229.

[GL90] S. Goldwasser, and L. Levin, "Fair Computation of General Functions in Presence of Immoral Majority", *CRYPTO '90, LNCS 537,* Springer-Verlag, 1990.

[GM84] S. Goldwasser and S. Micali, "Probabilistic encryption", *JCSS,* Vol. 28, No 2, April 1984, pp. 270-299.

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

[PW94] B. Pfitzmann and M.Waidner, "A General Framework for Formal Notions of Secure Systems", Hildesheimer Informatik-Berichte, ISSN 0941-3014, April 1994.

[PSW00] B. Pfitzmann, M. Schunter and M.Waidner, "Secure Reactive Systems", IBM Technical report RZ 3206 (93252), May 2000.

[S99] A. Sahai, "Non malleable, non-interactive zero knowlege and adaptive chosen ciphertext security", FOCS 99.

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

[Y86] A. Yao, "How to generate and exchange secrets", In *Proc. 27th Annual Symp. on Foundations of Computer Science (FOCS),* pages 162–167. IEEE, 1986.