J. Cryptology (2004) 17: 153–207
DOI: 10.1007/s00145-004-0135-x

Journal of
CRYPTOLOGY
© 2004 International Association for Cryptologic Research

Adaptive versus Non-Adaptive Security of Multi-Party Protocols*

Ran Canetti
IBM Research, 19 Skyline Drive,
Hawthorne, NY 10532, U.S.A.
canette@watson.ibm.com

Ivan Damgård
BRICS, Aarhus University,
Aabogade 34, DK-8000 Aarhus C, Denmark
ivan@daimi.au.dk

Stefan Dziembowski
Institute of Informatics, University of Warsaw,
Banacha 2, PL-02-D97 Warsaw, Poland
std@mimuns.edu.pl

Yuval Ishai
Department of Computer Science, Technion,
Haifa 32000, Israel
yuvali@cs.technion.ac.il

Tal Malkin
Computer Science Department, Columbia University,
New York, NY 6027-7003, U.S.A.
tal@cs.columbia.edu

Communicated by Oded Goldreich

Received 5 September 2002 and revised 16 September 2002
Online publication 22 March 2004

Abstract. Security analysis of multi-party 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.

* Work partially done while the third author was a PhD student at BRICS and while he was a post-doc at ETH Zurich. He was supported in part by Polish KBN Grant 4 T11C 042 25 and by the Foundation for Polish Science (FNP). Work done in part while the fourth author was at DIMACS and AT&T Labs, and while visiting IBM Watson. Work done while the fifth author was at MIT and AT&T Labs, and while visiting IBM Watson.

While affirming some prevailing beliefs, we also obtain some unexpected results. Some highlights of our results are:

- According to a more basic definition (due to 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.
- According to an augmented definition which is cast in an information-theoretic setting (due to Dodis, Micali, and Rogaway), adaptive and non-adaptive security are essentially equivalent. This holds for both honest-but-curious and Byzantine adversaries, and for any number of parties.

Key words. Multi-party protocols, Definitions of security, Adaptive security.

1. Introduction

Security analysis of cryptographic protocols is a delicate task. A first and crucial step towards meaningful 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 comprehensive 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 unnecessary requirements.

Indeed, in contrast with the great advances in constructing cryptographic protocols for a large variety of protocol problems, formalizing definitions of security for cryptographic 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 [GL], Micali and Rogaway [MR], Beaver [B1], Canetti [C1], and Dodis and Micali [DM] that concentrate on the task of *secure function evaluation* [Y1], [Y2], [GMW], and Pfitzmann and Waidner [PW], Pfitzmann et al. [PSW], and Canetti [C2] 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 modeled 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 represent realistic security threats better, 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 [DDN], [CS], [S] (or even plain semantically secure encryption against chosen plaintext attacks [GM], if used appropriately) is sufficient. To obtain adaptively secure encryption, it seems that one needs either to trust data erasure [BH] or use considerably more complex constructs [CFGN], [B2], [DN].

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? That is, do there exist protocols that are non-adaptively secure and at the same time adaptively insecure? Some initial results, indicating clear separation in some settings, are provided in [CFGN]. 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 Canetti [C1] (which we call the *basic* definition) and the definition of Dodis and coworkers [MR], [DM] (which we call the *augmented* definition). To develop the necessary terminology for presenting our results we very briefly outline the structure of the definitions of security of protocols.

As mentioned above, both definitions concentrate on the task of Secure Function Evaluation. Here the parties wish to evaluate jointly 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. Let k be the security parameter.

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 deviate arbitrarily from their protocol). Passive (resp., active) adversaries are often called honest-but-curious (resp., Byzantine).

NUMBER OF PARTIES: We distinguish among the following three cases. First is the case where n, the number of parties, is large, or $\omega(\log k)$. Second is the case of a small number of parties, where is $n = O(\log k)$. Third is the case of two parties.

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 and S's resources do not depend on A's. 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 [C1] 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), statistical emulation (where the output distributions should be statistically indistinguishable), or computational emulation (where the output distributions should 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: *the basic 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. Essentially the same results hold also for the *open channels* setting, where the adversary sees all communication but only *computational* security is required. (The only exception is the two-party case, as described below.)

The definition of adaptive security from [C1] requires a so-called *post-execution corruptibility* (*PEC*) property, geared towards allowing composition of adaptively secure protocols (see Section 2.1.2). Using the PEC requirement we can show a separation between adaptive and non-adaptive security for all settings except IT security (where adaptive and non-adaptive security are equivalent). However, as we will argue, our results indicate that PEC is an overkill requirement in some cases. We thus also investigate the basic definition *without* the PEC requirement, as follows. To give a more complete picture, we also consider two relaxed variants of the basic definition that are not explicitly addressed in [C1]. The first variant requires only IT security (in the secure channels model). The second variant allows the ideal-process adversary to send an explicit input to and receive output from the trusted party, modeling, respectively, the adversary's allowed influence on the outputs of uncorrupted parties, and the tolerable "information leakage" to the adversary.

We now outline our results. The most distinctive parameter here (when the PEC requirement is ignored) seems to be whether the adversary is active or passive. If the adversary is active (i.e., Byzantine), then, for three or more parties, 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 parties, for *any* constant.

The case of two parties with active adversaries is somewhat more involved. In the secure channels model, adaptive and non-adaptive security are always equivalent. However, in the open channels model, a variant of the example for three parties demonstrates that adaptive security is strictly stronger than non-adaptive security, as long as the ideal-process allows the adversary to receive an output from the trusted party. It remains open whether such a separation exists for the case of strict function evaluation where the ideal-process adversary has no output or input.

In the case of passive adversaries the situation is as follows. 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 parties. In all other eight settings we show that, roughly speaking, adaptive security is equivalent to non-adaptive security when the number of parties is small, and is strictly stronger when the number of parties is large. We elaborate below.

For a large number of parties, it follows from an example protocol shown in [CFGN] 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 parties and perfect emulation, adaptive and non-adaptive security are equivalent. Equivalence holds even in the case of statistical or computational emulation, if n is $O(\log k/\log\log k)$.[1]

Equivalence of adaptive and non-adaptive security for the case of passive adversaries and a small number of parties 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 one can thus concentrate on non-adaptive security, and adaptive security comes "for free." This significantly simplifies the construction and analysis of these protocols.

[1] 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.

Table 1. A summary of our main results.*

	Adversary	Number of parties	Adaptive vs. non-adaptive
Basic definition	Active	Three and above	Non-equivalent
		Two	Model-dependent
	Passive	Large $= \omega(\log k)$	Non-equivalent
		Small $= O(\log k)$	Equivalent
Augmented definition	Any	Any	Equivalent

*Equivalence results under the basic definition only hold without the PEC requirement. Some additional results and subtleties, depending on which variation of the model is used in the definition, are omitted from this table.

Our results: *the augmented definition*. This definition is formulated only for the case of IT security. It is strictly stronger (i.e., more restrictive) than the IT version of the basic definition. Here, to our surprise, adaptive and non-adaptive security turn out to be essentially equivalent, even for active adversaries, and regardless of the number of parties. Here, by "essentially," we mean that non-adaptive security implies adaptive security for all protocols, except for a class of degenerate cases. This class of protocols is easy to characterize but is not empty. More details are given within.

Two properties of the augmented 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 state this requirement formally 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.)

Our main results are summarized in Table 1.

Additional definition variants. We note that the equivalence and separation results of Table 1 also hold for the following two variants of our basic definition considered in the literature. A first such variant, used in [BH] to obtain adaptive security, assumes that uncorrupted parties can be trusted to properly erase their internal data. (In contrast, our basic definition assumes that an adaptive adversary learns *everything* that was viewed by a newly corrupted party before the corruption occurred.) A second variant of the basic definition allows an active adversary to "abort," namely, to decide whether uncorrupted parties obtain an output from the computation. Such a variant, used in [G1] and elsewhere, allows one to obtain general feasibility results for (computationally) secure function evaluation even when an active adversary can corrupt half or more of the parties, and is particularly useful in the two-party case.[2]

[2] The basic definition (without abort), where both parties should generate output, cannot be realized in the case of two parties and active adversaries and can corrupt one party. However, this definition *does* allow one to evaluate functions in which only one player receives an output.

On the ability to evaluate functions adaptively. This paper concentrates on the question of the existence of *protocols* that are adaptively insecure but non-adaptively secure. A related question is whether, and under which computational assumptions, there exist *functions* that can be securely computed non-adaptively, but cannot be adaptively securely computed. We leave this interesting question out of the scope of this paper.

Organization. Section 2 presents our results relating to the basic definition. Section 3 presents our results relating to the augmented definition.

2. Adaptivity versus Non-Adaptivity in the Basic Definition

This section describes our results relative to the [C1] definition of security. A more detailed review of the definition is provided in Section 2.1. The main parameters used for distinguishing among the different cases were briefly described in the Introduction.

First, in Section 2.2 we present a simple example that shows separation between non-adaptive and adaptive security for two or more parties, for either universal or computational security, and for both passive and active adversaries. This example demonstrates separation almost across the board. However, it strongly relies on the post execution corruptibility (PEC) property required by the adaptive definition. As argued later (Section 2.3), in some cases the PEC property is not necessary, and it is interesting to investigate the relations between non-adaptive and adaptive security *without* PEC. This is done next.

Section 2.3 proves equivalence of non-adaptive and adaptive security without PEC, for passive adversaries and a small number of parties. Section 2.4 shows equivalence for passive adversaries and *any* number of parties, in the setting of IT security and perfect emulation. (This result holds even with PEC.)

Next, we demonstrate some separating examples that work even without PEC. Section 2.5 describes separating examples for passive adversaries and a large number of parties. Section 2.6 shows a separating example for the case of an active adversary with at least three parties. Finally, Section 2.7 discusses the case of active adversaries and two parties.

2.1. *Review of the Definition*

For self containment, this section briefly sketches the definitions of [C1] 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, 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 what follows 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 are 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 $X \overset{\mathrm{d}}{=} Y$) 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 $X \overset{\mathrm{s}}{\approx} Y$) 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, defined by $\mathrm{SD}(Z_1, Z_2) = \frac{1}{2}\sum_a |\mathrm{Prob}(Z_1 = a) - \mathrm{Prob}(Z_2 = a)|$.

Ensembles X and Y are computationally indistinguishable (written $X \overset{\mathrm{c}}{\approx} Y$) if for all polynomial-time algorithms $\mathcal{D}$, 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 $f\colon \mathbf{N} \times (\{0, 1\}^*)^{n+1} \times \{0, 1\}^* \to (\{0, 1\}^*)^{n+1}$, where the first input is the security parameter k and the last input is taken to be the random input. The second input is taken to be the input of the adversary, and the first output is taken to be the output of the adversary (see more details below). When f is specified by a function from $(\{0, 1\}^*)^n$ to $(\{0, 1\}^*)^n$, it should be interpreted as a deterministic function mapping the n parties' inputs to their outputs.

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 $\mathcal{B} \subset 2^{\{1\cdots n\}}$ be a monotone collection of subsets of $\{1, \ldots, n\}$;

that is, if $B \in \mathcal{B}$ and $B' \subset B$, then $B' \in \mathcal{B}$.[3] 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 of the corrupted parties (some set in $\mathcal{B}$), together with their inputs and random inputs. In addition, $\mathcal{A}$ receives an auxiliary input z. The auxiliary input is a standard tool aimed to allow 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's 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

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

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 $\{\text{EXEC}_{\pi,\mathcal{A}}(k, \vec{x}, z)\}_{k\in\mathbf{N}, \langle\vec{x},z\rangle\in\{0,1\}^*}$.

The ideal process. The ideal process is parameterized by the function to be evaluated. This is an n-party function $f\colon \mathbf{N} \times (\{0,1\}^*)^{n+1} \times \{0,1\}^* \to (\{0,1\}^*)^{n+1}$, as

[3] This standard requirement models the assumption that if a certain set of parties can be corrupted, then so can all of its subsets. In the setting of adaptive security, it allows us to assume that a *single* additional party is corrupted in each step.

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). The parties wish to compute $f(k, x_0, \vec{x}, r_f)_1, \ldots, f(k, x_0, \vec{x}, r_f)_n$, where k is the security parameter, x_0 is the adversary's input, r_f is an appropriately long random string, the adversary learns $f(k, x_0, \vec{x}, r_f)_0$, and P_i learns $f(k, x_0, \vec{x}, r_f)_i$ (where $f(k, x_0, \vec{x}, r_f)_i$ denotes the ith component of $f(k, x_0, \vec{x}, r_f)$).[4] 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 (i.e., the parties P_i in the set C of corrupted parties), 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. In addition, $\mathcal{S}$ hands an arbitrary value x_0 to T. (If $\mathcal{S}$ is passive, then even the corrupted parties hand their original inputs to T.) Next, T chooses a value r_f uniformly from $\{0, 1\}^k$, hands $f(k, \vec{y}, r_f)_0$ to $\mathcal{S}$, and hands each P_i the value $f(k, \vec{y}, r_f)_i$.

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 $\{f(\vec{y}, r_f)_i\colon P_i \text{ is corrupted}\}$.

Let $\text{ADVR}_{f,\mathcal{S}}(k, \vec{x}, z, \vec{r})$, where $\vec{r} = (r_f, r)$, denote the output of the 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

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

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

[4] In [C1] the adversary is not allowed to provide explicitly input to f nor to receive output from f. See more discussion on this issue at the end of this section.

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 a (probabilistic) ideal-process adversary $\mathcal{S}$ such that $\text{IDEAL}_{f,\mathcal{S}} \overset{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A}}$ (resp., $\text{IDEAL}_{f,\mathcal{S}} \overset{\text{s}}{\approx} \text{EXEC}_{\pi,\mathcal{A}}$ or $\text{IDEAL}_{f,\mathcal{S}} \overset{\text{c}}{\approx} \text{EXEC}_{\pi,\mathcal{A}}$). If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries, then we say that π $\mathcal{B}$-privately evaluates f.

If the expected running time of $\mathcal{S}$ is polynomial in the expected running time of $\mathcal{A}$ and k, then we say that π has universal security.[5]

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

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.

Remark: *On the trusted party's input from and output to the adversary.* Allowing the ideal-process adversary to give explicit input to the trusted party (to be included in the computation of f) models the allowed *adversarial influence* on the outputs of the uncorrupted parties, and is only meaningful when the adversary is *active*. Allowing the ideal-process adversary to receive explicit output from the trusted party models the allowed information that the adversary may obtain on the inputs and outputs of the uncorrupted parties (even without corrupting any party).

We note that in [C1] the ideal process does not allow the adversary to provide the trusted party directly with input, nor can it receive output directly from the trusted party. Similarly, the definition of f does not allow for explicit adversarial input and output. Indeed, in the multi-party case, extending the ideal process to allow explicit adversarial

[5] The expected running time of an interactive algorithm $\mathcal{I}$ is measured as a function of k and is defined as follows. For any input x, random tape r of $\mathcal{I}$, and a computationally unbounded algorithm $\mathcal{E}$ interacting with $\mathcal{I}$ (i.e., playing the role of all parties with whom $\mathcal{I}$ communicates), let $T_{\mathcal{I}}(k, \mathcal{E}, x, r)$ denote the total running time of $\mathcal{I}$ on input k, x throughout the interaction with $\mathcal{E}$. The expected running time of $\mathcal{I}$ is defined as $T_{\mathcal{I}}(k) = \max_{\mathcal{E},x} \mathbf{E}_r[T_{\mathcal{I}}(k, \mathcal{E}, x, r)]$.

input and output is not necessary from a technical point of view, since one can always turn any n-party function in the extended model into an equivalent $(n+1)$-party function in the original model (i.e., without adversarial input/output) by designating a special party (say P_1) to be always corrupted and thus provide the adversary with the ability to give input to and get output from the trusted party.

Still, we choose to present here the more general formalization for two reasons. First, it allows for more direct and natural formalization of multi-party functions, thus it seems appropriate for a general definitional framework. Second, in the case of two parties and active adversaries the more general formalization simplifies our treatment (see more details in Section 2.7). In all other sections our results hold whether or not the adversary is allowed input to and output from the trusted party.

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. This includes its input, random input, and all messages it received and sent up to the point of corruption. This is contrasted with the model of [BH] which assumes trusted erasures. (See more discussion on this point in [C1].)

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 (non-concurrent) protocol composition.[6] 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. Multi-party 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 $\mathcal{B} \subset 2^{\{1\cdots n\}}$ be a monotone collection of subsets of $\{1 \cdots n\}$. 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

[6] By "non-concurrent composition" we mean modular composition as in [C1], where at each moment in time there is only a single protocol copy running (and all other copies are suspended).

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 predetermined 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 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 $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, \vec{x}, z)\}_{k\in\mathbf{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$.[7]

[7] 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.

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, each party P_i hands its (possibly modified) input value, y_i, to the trusted party T. In addition, $\mathcal{S}$ hands an arbitrary value x_0 to T. (If $\mathcal{S}$ is passive, then even the corrupted parties hand their original inputs to T.) Next, T chooses a value r_f uniformly from $\{0,1\}^k$, hands $f(k, \vec{y}, r_f)_0$ to $\mathcal{S}$, 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 $\vec{x} = x_1 \cdots x_n$ 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 $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \overset{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. Note that the environment is the same in the real-life model and the ideal process. This may be interpreted as saying that "for any environment and real-life adversary $\mathcal{A}$, there should exist an ideal-process adversary that successfully simulates $\mathcal{A}$ *in the presence of this specific environment*." 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 a (probabilistic) ideal-process adversary $\mathcal{S}$ such that $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \overset{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ (resp., $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \overset{\text{s}}{\approx} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ or $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \overset{\text{c}}{\approx} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$). If $\mathcal{A}$ and $\mathcal{S}$ are passive adversaries, then we say that π $\mathcal{B}$-privately evaluates f.

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

If $\mathcal{A}$ and $\mathcal{Z}$ are limited to expected polynomial time in k, and the expected running time of $\mathcal{S}$ is polynomial, then we say that π has computational security.

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 PEC stage is omitted, both in the real-life model and in the ideal process.

2.2. *Separation Relying on PEC*

This section demonstrates that, assuming the existence of (two-round, perfectly hiding) bit commitment, adaptive and non-adaptive security are not equivalent in the case of two or more parties, in all settings except IT security. Specifically, we show a separation between adaptive and non-adaptive security with any type of emulation, either universal or computational security, and either passive or active adversaries. This is done using the full requirements of the basic definition, including PEC.

Recall that the PEC requirement states that the simulator S in the adaptive setting first outputs a completed view of the protocol, and then is able to receive requests to post-execution corrupt a party. It should then respond with inputs, random choices and internal data of the corrupted party. These data should of course be consistent with the view that was output earlier.[8]

Our example uses the concept of a perfectly hiding bit commitment scheme,[9] which we briefly recall here. (See [G2] for more complete definitions.) A first ingredient in the definition of such a scheme is a probabilistic poly-time algorithm G, which takes 1^k as input, where k is a security parameter, and outputs a public key pk. The second ingredient is a poly-time computable function *commit* which takes as input pk, a bit b,

[8] Formally speaking, the PEC requests are issued by the *environment*, as described in Section 2.1, but this is not so important here.

[9] If instead we use statistically hiding or computationally hiding bit commitment, we obtain a separation for the setting with statistical or computational emulation, respectively.

and a random bit string r of length polynomial in k, and outputs a bit string called a *commitment* to b.

The way to use such a scheme is that a committer will be given a public key, and will commit to a bit b by evaluating the commit function and give the result to a verifier. Later, the committer may open the commitment by revealing b and the random choices he used. The two central properties required for this to work are:

hiding For any pk, the distributions of $\mathit{commit}(pk, 0, r)$ and $\mathit{commit}(pk, 1, r)$, where r is uniformly chosen, are identical.

binding Let A be any probabilistic poly-time algorithm. Let $p_A(k)$ be the probability that A on input pk produced by $G(1^k)$ returns a commitment c and strings r, s such that $c = \mathit{commit}(pk, 0, r) = \mathit{commit}(pk, 1, s)$. Then $p_A(k)$ is negligible in k.

It is well known (see [BC], [CDG], [P], and [CHP]) that (two-round) perfectly hiding commitment schemes exist under standard complexity assumptions such as hardness of discrete log, and in general existence of claw-free pairs of permutations

Given such a two-round perfectly hiding bit commitment scheme, consider the following function. There are two parties C and V, where C has as input a bit b and V has input $pk \in \{0, 1\}^*$ that is interpreted as a public key for a commitment scheme. C should output pk and V should output nothing. The protocol proceeds as follows:

1. V sends pk to C.
2. C computes $c = \mathit{commit}(pk, b, r)$, where r is uniformly chosen, and sends c to V. C outputs pk. V outputs nothing.

We allow an adversary to corrupt any subset of parties. It is then easy to verify that this protocol is secure against an active adversary in a non-adaptive setting: if only V is corrupted, let pk' be the value prepared by the adversary to be sent to C in the first step. Then a correctly distributed c can be constructed by using pk' to compute a commitment to an arbitrary value (this works by the hiding property). If only C is corrupted, the simulator obtains pk from the trusted party. If both parties are corrupted, the simulation can just follow the adversary's instructions.

We claim that on the other hand, this protocol is adaptively insecure, even for passive adversaries. Assume for contradiction that it is adaptively secure, and consider the adversary that corrupts V from the start, but lets the protocol finish without corrupting C. Let S be an (adaptive and poly-time) simulator for this adversary. Now consider the following probabilistic polynomial-time algorithm for breaking the binding property of the commitment scheme. It gets as input a public key pk for the scheme:

1. Run the simulator S. When S corrupts V in the ideal process, give it pk as the input for V. Note that in the view output by S, a commitment c (with respect to pk) must occur.
2. Issue a request to S to post-execution corrupt C. When S corrupts C in the ideal process, give it 0 as input for C. Save the output produced.
3. Rewind S to its state at the start of Step 2. Issue again a request to post-execution corrupt C, this time giving it 1 as input for C. Output the data obtained from S in this and the previous step.

It should be clear that internal data of C consistent with c and input b must contain a string r_b such that $c = commit(pk, b, r_b)$, and consequently the above algorithm contradicts the binding property we assumed. Note that the above adaptive adversary is passive, namely, we have proved that the protocol is adaptively insecure even for passive adversaries, while we have seen that the protocol is non-adaptively secure even for active adversaries. We thus have:

Theorem 4. *Assume that two-round perfectly hiding bit commitment schemes exist. Then for the case of two parties in the setting of universal or computational security with perfect emulation, if the PEC requirement is imposed in the definition, then adaptive security is strictly stronger than non-adaptive security, both for active adversaries and for passive adversaries.*

We stress that Theorem 4 does not apply to the case of IT security (which is addressed in Section 2.4).

2.3. *Equivalence for Passive Adversaries and a Small Number of Parties* (*No PEC*)

This section proves that, when the PEC requirement is not imposed, adaptive and non-adaptive security against a *passive* adversary are equivalent when the number of parties is small.

As previously discussed, the technical requirement of PEC is part of the [C1] definition of adaptive security (as well as [B1] and [MR], in different ways). This requirement is in general needed in order to guarantee secure composition of protocols in the adaptive setting. However, in the particular setting of this section, i.e., passive adversaries and a small number of parties, it turns out that PEC is an "overkill" requirement for guaranteeing non-concurrent composability of secure function evaluation protocols. Informally, the argument for this is the following. Let π and ρ be secure function evaluation 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 [C1], 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 (no PEC).

We conclude that whenever non-adaptive security is equivalent to adaptive security without PEC, then PEC is not needed for guaranteeing adaptively secure non-concurrent composition.[10] In this section we show that this is the case for passive adversaries and a small number of parties.[11]

As a first step towards the proof, we 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

[10] One should note that the PEC requirement allows a secure non-concurrent composition of an *arbitrary* protocol π (not necessarily a secure function evaluation protocol) with a secure protocol ρ [C1]. Thus, in the general case the properties that can be proved about the composed protocol in the non-adaptive setting may not be implied in the adaptive setting. The above discussion refers only to the (important) special case of composing secure function evaluation protocols.

[11] We remark that in the case of *concurrent* (and also universal) composition, a requirement that implies PEC is necessary for guaranteeing secure composition *even in the case of non-adaptive adversaries* [C2]. Thus, adaptive and non-adaptive security are not equivalent in that framework.

assume without loss of generality that the real-life adversary waits until the protocol terminates, and then starts to corrupt the parties adaptively; 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 of the remainder of this section, we make the following simplifying assumptions: (1) assume that the real-life model adversary is deterministic; (2) assume that the function computed by the protocol is deterministic and gives no output to the adversary; 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 parties, the *adversary* $\mathcal{A}$ and the *simulator* $\mathcal{S}$, and n distinct cards. The two parties 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 $V = (V_1, \dots, V_n)$ 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 uncover cards gradually, 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 $\tilde{V}_b$ may be arbitrarily (or adversarially) fixed, as long as they conform to the above similarity condition.

We 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 ith card by the adversary and learning V_i corresponds to corrupting the ith party P_i in the real-life model and learning its entire view: its input, random input, communication messages, and output. Uncovering the ith card by the simulator corresponds to corrupting P_i in the ideal process. Finally, the simulator sampling from $\tilde{V}_b$ corresponds to the adaptive ideal-process adversary invoking the non-adaptive ideal-process adversary; by the non-adaptive security of π, the produced distribution $\tilde{V}_b$ is similar to V_b. 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 consider are *perfect*, *statistical*, and *computational*, corresponding to the type of non-adaptive emulation we assume. We 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. 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 admissible distribution V, $\tilde{V}_b$ the simulator wins.

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.

In the rest of this section we adopt the simplified random variable notation from the game described above, but revert to the original terminology of corrupting parties rather than uncovering cards.

Our simulation paradigm

We now give some intuition for the simulator constructions that is presented next. Before doing so, it is instructive to explain why some simpler approaches fail. The simplest conceivable approach for simulating a given adversary strategy $\mathcal{A}$ is to let the simulator $\mathcal{S}$ "guess" the set b eventually corrupted by $\mathcal{A}$, corrupt the parties in b, and output a view sampled from $\tilde{V}_b$. However, one obvious problem with this approach is that the correct distribution of the set b is not known a priori, as it may depend on the unknown distribution V. Thus, we cannot simulate $\tilde{V}_b$ without first corrupting parties in b, and on the other hand we cannot corrupt parties without knowing that the adversary would corrupt them (since parties cannot be "uncorrupted").

Next, consider a "straight line" simulator which proceeds as follows. It starts by corrupting $b = \emptyset$. 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.

Our simulators are similar to the above in the sense that they incrementally expand a corrupted set b based on a view sampled from $\tilde{V}_b$. However, in contrast to the above, they also carry some state information from one iteration to the next, and insist that the sample from $\tilde{V}_b$ used for expanding b be consistent with this state information. In our main simulator construction (described in Section 2.3.1), this state information is simply the set b itself; to find the next party to corrupt, the simulator will attempt to sample a view from $\tilde{V}_b$ conditioned on the event that this view leads the adversary to corrupts b. A refinement of this state information is used in Section 2.3.2 for the case of imperfect emulation.

Black-box simulation. Instead of obtaining a different simulator $\mathcal{S}$ for every adversary $\mathcal{A}$, we present a single simulator $\mathcal{S}$ having a *black-box* access to $\mathcal{A}$. More specifically, $\mathcal{S}$ uses both the adaptive adversary $\mathcal{A}$ and a non-adaptive simulator (generating the distributions $\tilde{V}_b$) as "oracles." Each call to the non-adaptive simulator on input b will produce a sample from the distribution $\tilde{V}_b$, independently of all other samples. In each call to $\mathcal{A}$ on input $v = (v_1, \ldots, v_n)$, the simulator $\mathcal{S}$ may learn the full sequence of parties corrupted by $\mathcal{A}$ given the fixed view v.[12] By convention, the running time of a black-box simulator $\mathcal{S}$ will be measured as the number of oracle calls made by $\mathcal{S}$. The actual expected running time after implementing all oracle calls can be analyzed using the following lemma, which is a variant of Wald's equality (see Problem 22.9 in [B3]).

Lemma 5. *Let S be an algorithm which makes an expected number of s calls to a randomized procedure P. Each invocation of P uses independent random coins, and its expected running time (on every input) is bounded by t. Then the expected total time S spends on its calls to P is bounded by st.*

Note that Lemma 5 holds regardless of the stopping rule of S; in particular, the number of calls to P may depend on the internal random coins (or the running time) of these calls.

The above convention for measuring the running time of a black-box simulator $\mathcal{S}$ is particularly useful for proving universal security. Indeed, suppose that a protocol π is universally non-adaptively secure (hence each call to $\tilde{V}_b$ can be implemented in expected poly(k)-time). To prove that π is universally adaptively secure, it suffices to construct a black-box adaptive simulator $\mathcal{S}$ (with the required emulation quality), such that:

- The expected running time of $\mathcal{S}$ (i.e., the expected number of calls to $\mathcal{A}$ or $\tilde{V}_b$) is polynomial in k.
- The additional work of $\mathcal{S}$ per oracle call is at most polynomial in k and *linear* in the length of the oracle's output (hence also linear in the time of its implementation).

The second requirement, which is met by all of our simulators, allows us to absorb the additional work of $\mathcal{S}$ into the cost of implementing the oracle calls. Thus, in our analysis we ignore this additional work and only focus on counting the number of oracle calls.

2.3.1. *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 a black-box adaptive simulator $\mathcal{S}$ whose expected

[12] Recall that in this section we make the simplifying assumption that $\mathcal{A}$ is deterministic. In the general case, $\mathcal{A}$ uses an independent random tape for each call.

running time is linear in the size of the adversary structure. The construction from this section allows us to prove equivalence of non-adaptive and adaptive security both in the IT case (see Section 2.4) and, when the adversary structure is small, in the universal case.

In the description and analysis of $\mathcal{S}$ we use the following 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 $v \xrightarrow{A} b$ if the joint view v leads the adversary $\mathcal{A}$ to corrupt the set b *at some stage*. For instance, $v \xrightarrow{A} \emptyset$ always holds. An important observation is that whether $v \xrightarrow{A} b$ holds depends only on v_b.[13] This follows from our assumption that $\mathcal{A}$ starts corrupting parties only after the protocol terminates and from the fact that corrupted parties cannot be uncorrupted. Hence, we also use the notation $v' \xrightarrow{A} b$, where v' is a $|b|$-tuple representing a partial view. Note that, by the above conventions, each test whether $v \xrightarrow{A} b$ (for some fixed view v and set b) is counted as a single oracle call to $\mathcal{A}$.

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 $v' \overset{R}{\leftarrow} \tilde{V}_{b_i}$ until $v' \xrightarrow{A} b_i$ (i.e., the sampled partial view would lead $\mathcal{A}$ to corrupt b_i).[14] Let v_i be the last sampled view.
   (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 ith 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 ith iteration, the random variables indexed by i will be set to a special value "$\perp$."

Lemma 6. *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

[13] The general form of this statement (for the case of a randomized $\mathcal{A}$) is that Prob$[v \xrightarrow{A} b]$ depends only on v_b.

[14] Recall that sampling from $\tilde{V}_{b_i}$ corresponds to invoking the non-adaptive simulator on the inputs and outputs of parties in b_i. These are known to $\mathcal{S}$ because all parties in b_i have already been corrupted.

sampled from the distribution V_{b_i} ($\overset{\mathrm{d}}{=} \tilde{V}_{b_i}$) conditioned on the event $V_{b_i} \xrightarrow{A} b_i$. 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 $V \xrightarrow{A} b_i$ (since $B_i = b_i$ and $V \xrightarrow{A} b_i$ are two names for the same event). □

Lemma 7. *In the case of a perfect non-adaptive emulation*, $\tilde{V}_i \overset{\mathrm{d}}{=} V_i$ *for every* i.

Proof. It follows from Lemma 6 that if $\tilde{B}_i \overset{\mathrm{d}}{=} B_i$ then $\tilde{V}_i \overset{\mathrm{d}}{=} V_i$. It thus suffices to show that $\tilde{B}_i \overset{\mathrm{d}}{=} B_i$ for all i. Clearly, both $\tilde{B}_0$ and B_0 are deterministically the empty set. Now, suppose that $\tilde{B}_i \overset{\mathrm{d}}{=} B_i$ and hence also $\tilde{V}_i \overset{\mathrm{d}}{=} V_i$. We show that $\tilde{B}_{i+1} \overset{\mathrm{d}}{=} B_{i+1}$ by conditioning on the ith 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} 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 $\tilde{B}_{i+1} \overset{\mathrm{d}}{=} B_{i+1}$. □

Claim 8. *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 $(\tilde{V}_1, \tilde{V}_2, \dots, \tilde{V}_n)$ may be different from $(V_1, V_2, \dots, V_n)$. In particular, 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.[15] However, from $\tilde{V}_i$ (resp., V_i) *alone*, one may determine: (1) the probability of $\mathcal{S}$ (resp., $\mathcal{A}$) terminating in the ith iteration; and (2) the output distribution of $\mathcal{S}$ (resp., $\mathcal{A}$) given that it terminates in the ith iteration. Using Lemma 7 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 claim establishes a bound on the total *expected* running time of $\mathcal{S}$.

Claim 9. *In the case of perfect non-adaptive emulation, the expected running time of* $\mathcal{S}$ (*measured as the number of calls to* $\mathcal{A}$ *and* $\tilde{V}_b$) *is linear in* $|\mathcal{B}|$.

Proof. We count the expected number of times a view v' is sampled (in Step 2(a)) throughout the execution of $\mathcal{S}$. Let $\tilde{T}_i$ be a random variable counting the number of samples taken in the ith iteration, and let $\tilde{T} = \sum_i \tilde{T}_i$ 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

[15] This is a consequence of the fact that $\mathcal{S}$ does not keep track of the order by which it corrupted the parties in b_i. For instance, if the possible corruption sequences by $\mathcal{A}$ in the real-life model are (1, 2, 3) and (2, 1, 4), then $\mathcal{S}$ will also allow the sequences (1, 2, 4) and (2, 1, 3).

ith 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

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

where the first equality relies on the perfect emulation assumption, the second on equality of the relevant events, and the third on Lemma 7. Hence $E[\tilde{T}_i | \tilde{B}_i = b_i] = 1/\text{Prob}[\tilde{B}_i = b_i]$. Letting $\sup(\tilde{B}_i)$ denote the support set of the random variable $\tilde{B}_i$ (excluding $\perp$) we obtain

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

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

If $n = O(\log k)$, $|\mathcal{B}|$ is guaranteed to be polynomial in k. We may thus conclude the following froms Claims 8 and 9:

Theorem 10. *In the case of protocols for secure function evaluation, universal perfect security against passive adversaries, and $n = O(\log k)$ parties, adaptive and non-adaptive security are equivalent.*

2.3.2. *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 2(a) 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 in k, the simulator $\mathcal{S}$ will produce a (statistically or computationally) good output distribution even when given access to (statistically or computationally) imperfect non-adaptive simulators. Moreover, it turns out that when the adversary structure is polynomial in size, the expected running time of $\mathcal{S}$ is polynomial *except with negligible probability*. Later, we define a more sophisticated simulator $\mathcal{S}'$ which strictly achieves expected polynomial-time simulation, at the expense of making a stronger assumption on the size of the adversary structure.

A main tool in the following is a technical lemma referred to in what follows as the *adaptive sampling lemma*. For simplicity we use a non-uniform notion of computational indistinguishability. The lemma and its corollaries can be extended to the uniform setting as well. The lemma uses 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 let $\mathrm{Time}(S^D(k))$ 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 11 (Adaptive Sampling Lemma: Computational Version). *Let $C = \{C(k)\}$, $D = \{D(k)\}$ be two distribution ensembles such that $C \overset{c}{\approx} D$, and let S be an adaptive sampling algorithm such that S^C runs in expected polynomial time. Then*:

1. $S^C \overset{c}{\approx} S^D$;
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 $K \subseteq \mathcal{N}$,

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

for all $k \in K$. Let $p(k)$ be a polynomial bounding the expected running time of S^C. Define a *non-adaptive* sampling algorithm 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, then 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, then S_0 terminates and outputs a special symbol. By the Markov inequality, for all $k \in K$ we have $\mathrm{Prob}[\mathrm{Time}(S^C)(k) > q(k)] \le \frac{1}{3}k^{-c}$, hence

$$\mathrm{SD}(S^C(k), S_0^C(k)) \le \tfrac{1}{3}k^{-c}. \qquad (2)$$

It follows from the robustness of computational indistinguishability under multiple *non-adaptive* samples (see [G2]) that $S_0^C \overset{c}{\approx} S_0^D$ and hence

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

From the computational indistinguishability of S_0^C, S_0^D it also follows that

$$|\mathrm{Prob}[\mathrm{Time}(S^C(k)) > q(k)] - \mathrm{Prob}[\mathrm{Time}(S^D(k)) > q(k)]| \le k^{-\omega(1)},$$

from which we can conclude that for every $k \in K$, $\mathrm{Prob}[\mathrm{Time}(S^D)(k) > q(k)] \le \frac{1}{3}k^{-c} + k^{-\omega(1)}$ and

$$\mathrm{SD}(S_0^D(k), S^D(k)) \le \tfrac{1}{3}k^{-c} + k^{-\omega(1)}. \qquad (4)$$

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

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

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

Towards proving the second part, define the distribution ensembles $T_C \overset{\text{def}}{=} \{\text{Time}(S^C(k))\}$ and $T_D \overset{\text{def}}{=} \{\text{Time}(S^D(k))\}$. We first argue that T_C, T_D are *statistically* indistinguishable. Otherwise, by a similar Markov bound argument as above, there exists a non-adaptive sampling algorithm S_0 such that $\{\text{Time}(S_0^C(k))\}$ and $\{\text{Time}(S_0^D(k))\}$ are both polynomially bounded *in the worst case* 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 by an efficient non-uniform distinguisher, contradicting the assumption that $C \overset{\text{c}}{\approx} D$.

Now, let

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

Since $T_C \overset{\text{s}}{\approx} T_D$, 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 proceeds similarly to the proof for the computational case.

Lemma 12 (Adaptive Sampling Lemma: Statistical Version). *Let $C = \{C(k)\}$ and $D = \{D(k)\}$ be two distribution ensembles such that $C \overset{\text{s}}{\approx} D$. 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. $S^{C,\mathcal{A}} \overset{\text{s}}{\approx} S^{D,\mathcal{A}}$.
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 2.3.1 when given access to imperfect non-adaptive simulator oracles.

Corollary 13. *Suppose that the simulator $\mathcal{S}$ is run with oracle access to non-adaptive simulators $\tilde{V}_b$ that provide computational* (*resp*., *statistical*) *emulation*, *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 computational* (*resp*., *statistical*) *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 let D be a concatenation of all imperfect oracles $(\tilde{V}_b)_{b\in\mathcal{B}}$. Since $|\mathcal{B}|$ is polynomial, we have $C \overset{\mathrm{c}}{\approx} D$ (resp., $C \overset{\mathrm{s}}{\approx} D$). From an analysis of the simulator $\mathcal{S}$ in the perfect case, we have that: (1) S^C perfectly emulates $\mathcal{A}$; and (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. □

An alternative simulation strategy. The expected running time of the above simulator $\mathcal{S}$ 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

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

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 $v \xrightarrow{A} \pi$ 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 $v \xrightarrow{A} b_i$ in Step 2(a) of $\mathcal{S}$ is replaced by $v \xrightarrow{A} \pi_i$, and after adding p_{i+1} to b_i in Step 2(b) 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 14. *In the case of* perfect *non-adaptive emulation* ($V_b \overset{\mathrm{d}}{=} \tilde{V}_b$), *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 evidence that the path is *good*, in the sense that the probability of the event $\tilde{V}_{b_i} \overset{A}{\to} \pi_i$ is not much smaller than that of $\mathcal{S}'$ arriving at π_i. This is done by repeatedly rerunning the simulation history *in parallel* to sampling from $\tilde{V}_{b_i}$, and verifying that before history repeats itself k times (i.e., the same path π_i is taken in k reruns) the event $\tilde{V}_{b_i} \overset{A}{\to} \pi_i$ occurs at least $k/2$ times. Unless such evidence 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.

However, rerunning the entire simulation from scratch is too costly, as it would make the running time grow by (at least) a factor of k for each corrupted party. Consequently, this approach can only be used in a case where the number of corrupted parties is constant. To get around this difficulty, we replace a rerun of the entire simulation history by a lighter procedure, which attempts to arrive at the corrupted path π_i without verifying that its sub-paths are good. This is where we utilize the fact that the simulator keeps track of the entire corruption path (as in $\vec{\mathcal{S}}$) rather than just the set of corrupted parties (as in $\mathcal{S}$). Otherwise, the rerunning procedure could potentially loop forever while trying to follow a path which is different from the one originally taken, even if this path eventually leads to corrupting the same set of parties.

The simulator $\mathcal{S}'$ is described in detail below. Somewhat abusing notation, given a path π_i we 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 let $\pi_0 = (b_0)$ be the initial path ($\pi_i$ 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 $\pi_i$:
       **Repeat**
           (i) Call procedure Rerun($\pi_i$), defined below; if it returns *success* incre-
               ment the counter $c$.
          (ii) Sample $v' \overset{R}{\leftarrow} \tilde{V}_{b_i}$; if $v' \overset{A}{\to} \pi_i$, 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 first to 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 $\pi_{i+1}$ be the
       path obtained by concatenating $b_{i+1}$ to the end of $\pi_i$, and iterate to the next $i$.
```

Procedure $\mathsf{Rerun}(\pi_i = (b_0, \ldots, b_i))$

```
For $i' = 0$ to $i - 1$ do
    Sample $v' \overset{R}{\leftarrow} \tilde{V}_{b_{i'}}$ until $v' \overset{A}{\rightarrow} \pi_{i'}$;
    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 $\vec{\mathcal{B}}$, let $\tilde{\Pi}'_i$ be a random variable taking the value of the corrupted path in the ith iteration of $\mathcal{S}'$, and let $\tilde{T}'_i$ be the running time of the ith iteration. It is 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 ith iteration of $\vec{\mathcal{S}}$.

We will show that the expected running time of the ith iteration of $\mathcal{S}'$ is polynomial in k and $\#\text{paths}_i$. Conditioning on the path π_i and using Lemma 5, we have

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

We bound the above expression using the following lemmas.

Lemma 15. $\text{Prob}[\tilde{\Pi}'_i = \pi_i] \leq \text{Prob}[\tilde{\Pi}_i = \pi_i]$.

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

Lemma 16. *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. □

Lemma 17. $E[\text{Time}(\mathsf{Rerun}(\pi_i))] = \sum_{i'<i} \text{Prob}[\tilde{\Pi}_{i'} = \pi_{i'}]/\text{Prob}[\tilde{V}_{b_{i'}} \overset{A}{\rightarrow} \pi_{i'}]$.

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 $\text{Prob}[\tilde{\Pi}_{i'} = \pi_{i'}]/\text{Prob}[\tilde{V}_{b_{i'}} \overset{A}{\rightarrow} \pi_{i'}]$. □

Lemma 18. *For any path π_i such that* $\text{Prob}[\tilde{\Pi}'_i = \pi_i] > 0$,

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

Proof. First note that when the above expression is undefined, i.e., when 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 15, $\text{Prob}[\tilde{\Pi}'_i = \pi_i] \cdot (1/\text{Prob}[\tilde{\Pi}_i = \pi_i]) \le 1$. 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 17, there is $i' < i$ such that

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

To analyze the probability of passing the i'th test in Step 2(b), let $p_1 = \text{Prob}[\tilde{\Pi}_{i'} = \pi_{i'}]$ and $p_2 = \text{Prob}[\tilde{V}_{b_{i'}} \xrightarrow{A} \pi_{i'}]$. By (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. □

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

Lemma 19. $E[\tilde{T}'_i] = O(k \cdot i \cdot \#\text{paths}_i)$.

Proof. Substituting Lemma 16 in (5) and applying Lemma 18, we get

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

From the last lemma we may conclude the following:

Claim 20. *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 21. *Suppose that $|\vec{\mathcal{B}}|$ is polynomial in k, and that the non-adaptive simulators $\tilde{V}_b$ provide computational (resp., statistical) emulation. Then the simulator $\mathcal{S}'$ provides computational (resp., statistical) emulation.*

Proof. It follows from Lemma 14 and the adaptive sampling lemma that when $|\vec{\mathcal{B}}|$ is polynomial, $\vec{\mathcal{S}}$ provides computational (resp., statistical) emulation. 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 by $\mathcal{S}'$ in Step 2(b) with π_i as the current path, and let p_1, p_2 be the two relevant probabilities from the proof of Lemma 18. That is, $p_1(\pi_i) = \text{Prob}[\tilde{\Pi}_i = \pi_i]$ and $p_2(\pi_i) = \text{Prob}[\tilde{V}_{b_i} \overset{A}{\rightarrow} \pi_i]$. The difference $|p_1 - p_2|$ must be bounded by some negligible function $\varepsilon(k)$. Indeed, both probabilities are negligibly close to $\text{Prob}[V_{b_i} \overset{A}{\rightarrow} \pi_i]$. Now, call π_i *good* if $p_1(\pi_i) > 3\varepsilon(k)$ and *bad* otherwise. Since by Lemma 15 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 in k. Finally, since for any good path π_i we have $p_2(\pi_i) \geq p_1(\pi_i) - \varepsilon(k) > \frac{2}{3}p_1(\pi_i)$, 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). □

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 22. *For function evaluation protocols and $n = O(\log k/\log\log k)$ parties, adaptive and non-adaptive security against* passive *adversaries 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 22 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.

2.4. *Equivalence for Passive Adversaries and Perfect IT Security*

In the case of IT security and perfect emulation, Claim 8 immediately implies the following:

Theorem 23. *For function evaluation protocols with passive adversary and perfect IT security, adaptive and non-adaptive security are equivalent.*

Note that there is no dependence on the number of parties in the above theorem. We also remark that this result holds even if the adaptive definition is augmented to require PEC.

2.5. *Separation for Passive Adversaries and a Large Number of Parties*

In [CFGN] Canetti et al. show an example protocol that separates adaptive and non-adaptive security for passive adversaries and a large number of parties, 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 parties, 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 [CFGN] 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 universal or computational security. This example holds even when PEC is not required.

Our example relies on the existence of perfectly hiding bit commitment schemes and collision-intractable hash functions.[16] For n parties, we need to hash n commitments in a collision-intractable manner. Thus, the number of parties 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 sub-exponential hardness assumption. For simplicity, we refer below to a large number of parties, instead of making the explicit distinction based on the quality of computational assumption.

The protocol involves parties $P_0, P_1, \ldots, P_n$, 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 party is h, pk. The protocol proceeds as follows:

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

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

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 by the perfect hiding of the commitment scheme it can now perfectly simulate any message from non-corrupted parties.

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 $h(c_1, \ldots, c_n)$. Then $\mathcal{A}$ interprets the result in some fixed efficient deterministic way as a subset of the parties $P_1, \ldots, P_n$ of size $n/2$, and corrupts this subset.

We will show that an adaptive simulator $\mathcal{S}$ for $\mathcal{A}$ can be used to break either the commitment scheme or the hash function family. The high-level argument is as follows. The simulator $\mathcal{S}$ must decide on the identity of the corrupted set b before having full access to the inputs of the parties in b. (In particular, the input of the last party corrupted

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

by $\mathcal{S}$ does not affect the identity of b.) Thus, given inputs h, pk for P_0, the simulator $\mathcal{S}$ can be used to produce *efficiently* two simulated views v_0, v_1 (consistent with h, pk) such that the same set b is corrupted in the two views, but the inputs of the corrupted parties differ. By the perfect emulation requirement, the messages $c_1, \ldots, c_n$ in the two views should be consistent with b, so by the security of the hash function these messages will almost always be the same in v_0 and v_1. However, since the random inputs r_j $(j \in b)$ are included in the views, v_0 and v_1 allows us to open the commitments $(c_j)_{j \in b}$ in two different ways, corresponding to the difference between the inputs in these two views. We thus get a contradiction to the security of the commitment scheme.

An algorithm which uses $\mathcal{S}$ to break either of the two primitives is formally described as follows. The algorithm gets input h, pk and then proceeds as follows:

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 party P_i, $i \geq 1$, give it 0 as an 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 party $\mathcal{S}$ 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 parties, except for P_j where we give a 1 as an 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 that the set of corrupted parties must be the same in v_0 and v_1. This is so since $\mathcal{A}$ always corrupts P_0 and $n/2$ parties 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 $c_1, \ldots, c_n$ appearing in v_0 are not the same as those in v_1, in which case we have a collision for h. Otherwise c_j 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 24. *For passive adversaries and a large number of parties, adaptive security (even without the PEC requirement) 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.*

2.6. *Separation for Active Adversaries with at Least Three Parties*

This section shows that when three or more parties are present, adaptive and non-adaptive security are not equivalent in the case of active adversaries, for all settings considered here: IT, universal, and computational security, with perfect, statistical, or computational emulation. This is proved via a simple three-party protocol for secure function evaluation which is non-adaptively secure, but adaptively insecure, in all above settings. This separation applies also to other variations in the model, such as open channels, and whether we require security with or without PEC.

Our separating protocol involves three parties 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 party subsets that can be corrupted) contains any possible subset of parties. 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 25. *Protocol π_{act} non-adaptively, $\mathcal{B}$-securely evaluates f_{act} with universal security and perfect emulation.*

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 D and R_1 (resp., D and R_2): Here $\mathcal{S}$ may hand in to the trusted party $0, s_2'$ (resp., $s_1', 0$) as the input of D, where s_2' (resp., s_1') is the message prepared by $\mathcal{A}$ to be sent to R_2 (resp., to R_1).

Next, consider $\mathcal{A}$ that corrupts R_1 (and/or R_2). The simulator $\mathcal{S}$ proceeds as follows. $\mathcal{S}$ corrupts R_1 (and/or R_2) in the ideal model, hands the empty input to the trusted party, and obtains the output s_1 (and/or s_2) in the ideal model. $\mathcal{S}$ then hands s_1 (and/or s_2) to $\mathcal{A}$ as the message that was sent from D to R_1 (and/or to R_2), 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 $\mathcal{A}$.

Finally, if $\mathcal{A}$ corrupts all parties D, R_1, R_2, a simulator $\mathcal{S}$ may simply corrupt all parties, hand the inputs to $\mathcal{A}$, and proceed to follow $\mathcal{A}$'s instructions, thus inducing an identical global output. □

Claim 26. *Protocol π_{act} does* not *adaptively $\mathcal{B}$-securely evaluate function f_{act}, with either universal, IT, or computational security.*

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 $s_1 = 1$, R_2 will output 0, while at the same time corrupting D only when "necessary" (i.e., when $s_1 = 1$), and never corrupting R_2. While each of the above goals (influencing the output, and the identity of the parties corrupted) can be achieved separately by an (active and adaptive) adversary in the ideal model, their *combination* cannot, implying the adaptive insecurity.

$\mathcal{A}$ is defined as follows. $\mathcal{A}$ starts by corrupting R_1 and receiving s_1 in the first stage of the protocol. If $s_1 = 0$, then $\mathcal{A}$ terminates. If $s_1 = 1$, then $\mathcal{A}$ 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, whenever $s_1 = 1$, R_2 always outputs 0. At the same time, $\mathcal{A}$ never corrupts R_2, and, moreover, $\mathcal{A}$ never corrupts D when D's input contains $s_1 = 0$. Now let $\mathcal{S}$ be an arbitrary unbounded adaptive ideal-process simulator, which never corrupts R_2. (Below "overwhelming probability" refers to $1 - neg$ for some negligible function neg.) If, when interacting with $\mathcal{S}$ in the ideal model, R_2 outputs 0 with overwhelming probability whenever $s_1 = 1$, then it must be that with overwhelming probability, whenever $s_1 = 1$, $\mathcal{S}$ 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 D is corrupted with overwhelming probability when the input is $s_1 = 0$. However, in the real world, $\mathcal{A}$ never corrupts D in this case, and so the global views are significantly different. □

Since Claim 25 asserts the security of π_{act} for all types of security and all types of emulation, and Claim 26 asserts the *in*security of π_{act} for all types of security and all types of emulation, together the two claims separate adaptive security from non-adaptive security for active adversaries in all settings considered. We have:

Theorem 27. *For active adversaries, adaptive security is strictly stronger than non-adaptive security, for any type of security, and any type of emulation, as long as there are at least three parties.*

Discussion. The above separating example captures in a natural way the essence of the difference between non-adaptive and adaptive security. Indeed, π_{act} is a straightforward 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.

At 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. An adaptive active adversary may then use this information to decide whether to corrupt a party (and which one), and then change the party's input to influence the global output of the execution.

On the other hand, as we have shown, for a passive adversary and IT security, non-adaptive security is equivalent to adaptive security. This may suggest the intuition that even for active adversaries, in the IT 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 (see [BGW] and [CDM]), and, furthermore, the augmented definition [MR], [DM] *requires* this condition. In Section 3 we indeed formalize and prove this intuition, showing equivalence for the augmented definition.

2.7. *Active Adversaries*: *The Case of Two Parties*

While for three parties and active adversaries the separating example above holds in all settings, for two parties and active adversaries the situation is more involved, and depends on the details of the model. Note that in the two-party case we are generally only interested in the case where the adversary is computationally bounded. (Indeed, if the adversary is computationally unbounded, then only trivial functions can be computed securely, see, e.g., [CK], [K], and [KKMO].)

As we have already seen in Section 2.2, with the PEC requirement there is a separating example relying on bit-commitment. Without PEC, we distinguish two main cases, depending on whether the communication channel between the two parties is *secure* or *open*. In Section 2.7.1 we show that if the channel is secure, then adaptive and non-adaptive security are equivalent. In Section 2.7.2 we show that if the channel is open, then adaptive security is strictly stronger than non-adaptive security, as long as we allow the function to provide an output to the adversary in the ideal model even when none of the parties is corrupted.

2.7.1. *Equivalence with Secure Channel* (*no PEC*)

In this section we show equivalence in the model where the adversary is active and the communication between the two parties remains unknown to the adversary as long as he did not corrupt any party. Let P_0 and P_1 be two parties executing a protocol π. Without loss of generality assume that the adversary structure $\mathcal{B}$ consists of all the subsets of $\{P_0, P_1\}$, and that the adversary and the environment are deterministic. Also, as before we assume that the function f to be evaluated takes no input from and gives no output to the adversary. The proofs extend in a straightforward way to the general case.

For clarity we first analyze the case of perfect emulation, although in the two-party setting it is of little interest by itself. We then proceed to the more interesting case of imperfect (i.e., statistical or computational) emulation.

Perfect emulation. Suppose that protocol π non-adaptively $\mathcal{B}$-securely evaluates some function $f(x_0, x_1)$ with perfect emulation. Let $\mathcal{A}$ be an adaptive adversary and let $\mathcal{Z}$ be an environment. We will construct an adaptive simulator $\mathcal{S}$ which perfectly emulates the interaction of $\mathcal{A}$ with the parties and $\mathcal{Z}$ (i.e., such that (8) below holds). We first present a high-level description of $\mathcal{S}$ and then proceed to the details. At the beginning (Step 2 below), $\mathcal{S}$ checks whether $\mathcal{A}$ corrupts any party at all. If it does not, then the simulation is trivial. Otherwise let P_i be the party that $\mathcal{A}$ corrupts first. In this case $\mathcal{S}$ constructs (in Step 4) a non-adaptive adversary $\mathcal{A}_{\text{na}}$ that also corrupts P_i and then repeats the actions of $\mathcal{A}$. The adversary $\mathcal{A}_{\text{na}}$ can do it as long as $\mathcal{A}$ does not request corrupting P_{1-i} (in this case $\mathcal{A}_{\text{na}}$ outputs `error`). Then, $\mathcal{S}$ runs a non-adaptive simulator $\mathcal{S}_{\text{na}}$ of $\mathcal{A}_{\text{na}}$. If $\mathcal{S}_{\text{na}}$ outputs a non-error message then $\mathcal{S}$ outputs this message and terminates. Otherwise it knows that $\mathcal{A}$ (whose actions $\mathcal{A}_{\text{na}}$ is repeating) requested corrupting P_{1-i}, and therefore corrupts it as well. Now both parties are corrupted and the simulation can be done by simply executing the protocol (Step 6). We now present $\mathcal{S}$ in detail.

Simulator $\mathcal{S}$

1. Input a security parameter k and receive auxiliary information *aux* from the environment $\mathcal{Z}$.
2. Start $\mathcal{A}$ with a security parameter k, and pass *aux* to it (acting as the environment). Since the channel between P_0 and P_1 is secure, $\mathcal{A}$ does not expect to get any information until he corrupts some party. Therefore the execution of $\mathcal{A}$ works until one of the following happens:
   (a) $\mathcal{A}$ requests corrupting some party, or
   (b) $\mathcal{A}$ halts with some output w.
   If (b) happened, then output w and halt. Otherwise let P_i (with $i \in \{0, 1\}$) be the party that $\mathcal{A}$ requested to corrupt.
3. Corrupt P_i. Let aux_i be the message received from $\mathcal{Z}$ after corrupting P_i and let x_i be the input of P_i.
4. Construct a non-adaptive adversary $\mathcal{A}_{\text{na}}$ that corrupts P_i and then repeats the actions of $\mathcal{A}$. More precisely, he does it by simulating $\mathcal{A}$ in the following way:[17]

   Adversary $\mathcal{A}_{\text{na}}$

   (a) $\mathcal{A}_{\text{na}}$ takes an auxiliary input z_{na}. In our contexts z_{na} will always be equal to (aux, i, aux_i).
   (b) Start $\mathcal{A}$. Pass auxiliary information *aux* to $\mathcal{A}$ (acting as the environment).
   (c) Corrupt P_i. Let x'_i be the input of P_i. Until $\mathcal{A}$ requests corrupting P_i force P_i to behave according to the protocol π (with an input x'_i and some randomly chosen random input).
   (d) When $\mathcal{A}$ requests corrupting P_i (we know that it has to happen since $\mathcal{A}$ is deterministic) pass the internal state of the simulated P_i and (acting as the environment) the message aux_i to $\mathcal{A}$. Give to the simulated adversary $\mathcal{A}$ full control over P_i. The simulation continues until one of the following happens:
       (i) $\mathcal{A}$ requests corrupting P_{1-i}, or
       (ii) $\mathcal{A}$ halts with some output w.
       If (ii) happened, then output w, otherwise output a message `error`. Then halt.

   From the non-adaptive security of π we know that there exists a simulator $\mathcal{S}_{\text{na}}$ that simulates $\mathcal{A}_{\text{na}}$ (i.e., such that (7) below holds). Next, run $\mathcal{S}_{\text{na}}$ (with a security parameter k and auxiliary input (aux, i, aux_i)). $\mathcal{S}_{\text{na}}$ will corrupt P_i and expect to receive the input of P_i. Send x_i to $\mathcal{S}_{\text{na}}$. As long as $\mathcal{S}_{\text{na}}$ does not go to the computation stage, stay in the first corruption stage. Once $\mathcal{S}_{\text{na}}$ goes to the computation

[17] The reader may observe that $\mathcal{A}_{\text{na}}$ could be constructed beforehand, i.e., before the start of $\mathcal{S}$.

stage, receive the input y_i of the corrupted P_i (acting as a trusted party), enter the computation stage and send y_i to the trusted party. Then go to the second corruption stage, receive from the trusted party the output of P_i and send it to $\mathcal{S}_{\mathrm{na}}$.

If the $\mathcal{S}_{\mathrm{na}}$ outputs `error` then go to the next step. Otherwise output the output of $\mathcal{S}_{\mathrm{na}}$.

5. Corrupt P_{1-i} (let aux_{1-i} be the message received from the environment).
6. Now we know the inputs x_0 and x_1 of P_0 and P_1, resp. (we also know the auxiliary messages sent by the environment when P_0 and P_1 get corrupted). Execute the following procedure sim_both$_{\mathcal{S}}$:

sim_both$_{\mathcal{S}}$

(a) Simulate the execution of π with $\mathcal{A}$ and $\mathcal{Z}$, providing P_0 and P_1 with randomly chosen random inputs and with inputs x_0 and x_1, and (acting as $\mathcal{Z}$) sending to $\mathcal{A}$ the values aux, aux_i and aux_{1-i} if needed.

(b) If in this simulation both players got corrupted then output the output of $\mathcal{A}$ and halt.
Otherwise (i.e. when P_{1-i} remained honest) restart the procedure, i.e. go to (a).

This completes the description of $\mathcal{S}$. We now prove that $\mathcal{S}$ provides a perfect emulation of $\mathcal{A}$.

Lemma 28. *Let $f, \pi, \mathcal{A}, \mathcal{A}_{\mathrm{na}}, \mathcal{S}, \mathcal{S}_{\mathrm{na}}$ and $\mathcal{Z}$ be as above. Suppose*

$$\mathrm{IDEAL}_{f,\mathcal{S}_{\mathrm{na}}} \overset{\mathrm{d}}{=} \mathrm{EXEC}_{\pi,\mathcal{A}_{\mathrm{na}}}. \tag{7}$$

Then

$$\mathrm{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \overset{\mathrm{d}}{=} \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}. \tag{8}$$

Proof. For simplicity we assume that $\mathcal{A}$ always corrupts at least one party (the proof can be easily extended to cover the general case). For every k, z, and $\vec{x} = (x_0, x_1)$, consider the real-life experiment producing $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, \vec{x}, z)$, and define the random variable

$$\mathcal{C}_{\mathcal{E}}(k, \vec{x}, z) \overset{\mathrm{def}}{=} \begin{cases} 1 & \text{if } \mathcal{A} \text{ corrupts both parties,} \\ 0 & \text{otherwise (i.e., } \mathcal{A} \text{ corrupts one party).} \end{cases}$$

Similarly, consider the ideal process experiment producing $\mathrm{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(k, \vec{x}, z)$, and let

$$\mathcal{C}_{\mathcal{I}}(k, \vec{x}, z) \overset{\mathrm{def}}{=} \begin{cases} 1 & \text{if } \mathcal{S} \text{ corrupts both parties,} \\ 0 & \text{otherwise (i.e., } \mathcal{S} \text{ corrupts one party).} \end{cases}$$

Since the probability of $\mathcal{S}_{\mathrm{na}}$ outputting `error` is equal to the probability of $\mathcal{A}$ corrupting both parties, we have

$$\mathcal{C}_{\mathcal{E}} \overset{\mathrm{d}}{=} \mathcal{C}_{\mathcal{I}}. \tag{9}$$

To simplify notation, let $\mathcal{E}(k, \vec{x}, z) \overset{\text{def}}{=} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, \vec{x}, z)$ and $\mathcal{I}(k, \vec{x}, z) \overset{\text{def}}{=} \text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(k, \vec{x}, z)$. Using this notation, we need to prove that $\mathcal{E} \overset{\text{d}}{=} \mathcal{I}$.

Observe that the adversary $\mathcal{A}_{\text{na}}$, constructed in Step 4, behaves exactly like $\mathcal{A}$ as long as $\mathcal{A}$ does not corrupt both parties (in which case $\mathcal{A}_{\text{na}}$ outputs `error`). Therefore, from the fact that $\mathcal{S}_{\text{na}}$ is a (non-adaptive) simulator of $\mathcal{A}_{\text{na}}$ (satisfying (7)) we get the following equality of conditional distributions:

$$\begin{aligned}[\mathcal{E} \mid \mathcal{C}_{\mathcal{E}} = 0] &\overset{\text{d}}{=} [\mathcal{I} \mid \mathcal{S}_{\text{na}} \text{ does not output } \texttt{error}] \\ &\overset{\text{d}}{=} [\mathcal{I} \mid \mathcal{C}_{\mathcal{I}} = 0] \qquad (10)\end{aligned}$$

(where $[\mathcal{X} \mid E]$ denotes the ensemble containing the distribution $\mathcal{X}(k, \vec{x}, z)$ conditioned on the event $E(k, \vec{x}, z)$). Now, suppose that $\mathcal{C}_{\mathcal{I}} = 1$, i.e., $\mathcal{S}$ has corrupted both parties and entered Step 6. In this step $\mathcal{S}$ simply simulates the experiment $\mathcal{E}$ repeatedly until it finds an execution in which both parties get corrupted. Therefore we get

$$\begin{aligned}[\mathcal{E} \mid \mathcal{C}_{\mathcal{E}} = 1] &\overset{\text{d}}{=} [\mathcal{I} \mid \mathcal{S} \text{ has entered Step 6}] \\ &\overset{\text{d}}{=} [\mathcal{I} \mid \mathcal{C}_{\mathcal{I}} = 1]. \qquad (11)\end{aligned}$$

Combining (9), (10), and (11) we obtain (8). □

Theorem 29. *Consider the model with active adversary, secure channels, and no PEC. Suppose that a two-party protocol π non-adaptively $\mathcal{B}$-securely evaluates f with IT (resp., universal, computational) security and perfect emulation. Then π adaptively $\mathcal{B}$-securely evaluates f with IT (resp., universal, computational) security and perfect emulation.*

Proof. From (8) we get that for every $\mathcal{A}$ and $\mathcal{Z}$ there exists $\mathcal{S}$ such that $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}} \overset{\text{d}}{=} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. It remains to show that the expected running time of $\mathcal{S}$ is polynomial in that of $\mathcal{A}$.

The only non-trivial part is to prove that the expected number of restarts in the $\mathsf{sim\_both}_{\mathcal{S}}$ procedure is small. Let p be the probability that $\mathcal{S}_{\text{na}}$ outputs `error`. From the fact that $\mathcal{S}_{\text{na}}$ perfectly emulates $\mathcal{A}_{\text{na}}$ (and from the construction of $\mathcal{A}_{\text{na}}$) we know that p is equal to the probability that $\mathcal{A}$ corrupts both parties. Hence, similarly to the analysis in Claim 9, the expected number of restarts of $\mathsf{sim\_both}_{\mathcal{S}}$ is equal to $p \cdot (1/p) = 1$ (the probability of executing $\mathsf{sim\_both}_{\mathcal{S}}$ times the expected number of restarts given that $\mathsf{sim\_both}_{\mathcal{S}}$ is executed). □

Statistical and computational emulation. The case of imperfect emulation is slightly more involved. Similarly to the passive model (see Section 2.3.2), there may be situations where $\mathsf{sim\_both}_{\mathcal{S}}$ is restarted forever with non-zero (though negligible) probability. To guarantee bounded (expected) running time also in the case of imperfect emulation, we modify the simulator $\mathcal{S}$ in the spirit of the modified simulator from Section 2.3.2.[18]

[18] In fact, the current two-party setting allows for a considerably simpler modification of $\mathcal{S}$ than in Section 2.3.2.

We denote the modified simulator by $\mathcal{S}'$. The only difference between $\mathcal{S}$ and $\mathcal{S}'$ is in Step 6, which now attempts to limit the number of restarts of the sim_both procedure. Before each restart we execute the simulator $\mathcal{S}_{\text{na}}$. If $\mathcal{S}_{\text{na}}$ outputs `error` k times, then $\mathcal{S}'$ decides not to restart anymore and outputs `loop`. We later show that $\mathcal{S}'$ outputs `loop` with negligible probability (equation (14)).

The new sim_both procedure is described as follows.

sim_both$_{\mathcal{S}'}$

(a) Set $c := 1$.
(b) Simulate the execution of π with $\mathcal{A}$ and $\mathcal{Z}$, providing P_0 and P_1 with randomly chosen random inputs and with inputs x_0 and x_1, and (acting as $\mathcal{Z}$) sending to $\mathcal{A}$ the values *aux*, aux_i, and aux_{1-i} if needed.
(c) If in this simulation both players got corrupted, then output the output of $\mathcal{A}$ and halt.
Otherwise execute $\mathcal{S}_{\text{na}}$ (with fresh randomness and auxiliary inputs (aux, i, aux_i)). Recall that $\mathcal{S}_{\text{na}}$ interacts with the trusted party. Therefore we need to simulate it (we can do it since we know the inputs x_0 and x_1).
If $\mathcal{S}_{\text{na}}$ outputs `error`, then increase c by 1.
(d) If $c > k$, then output `loop` and halt. Otherwise go to (b) (we refer to it as "restarting" the procedure).

We now prove the main lemma.

Lemma 30. *Let f, π, $\mathcal{A}$, $\mathcal{A}_{\text{na}}$, $\mathcal{S}'$, $\mathcal{S}_{\text{na}}$, and $\mathcal{Z}$ be as above. Suppose*

$$\text{IDEAL}_{f,\mathcal{S}_{\text{na}}} \overset{\text{c}}{\approx} \text{EXEC}_{\pi,\mathcal{A}_{\text{na}}}. \tag{12}$$

Then

$$\text{IDEAL}_{f,\mathcal{S}',\mathcal{Z}} \overset{\text{c}}{\approx} \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}. \tag{13}$$

Proof. Again, for simplicity assume that $\mathcal{A}$ always corrupts at least one party, where the party first corrupted is P_i. We define the random variables $\mathcal{E}, \mathcal{I}, \mathcal{C}_{\mathcal{E}}, \mathcal{C}_{\mathcal{I}}$ as in the proof of Lemma 28, where in the definitions of $\mathcal{I}, \mathcal{C}_{\mathcal{I}}$ we replace the original simulator $\mathcal{S}$ with the modified simulator $\mathcal{S}'$. Finally, we define an additional random variable corresponding to the ideal-process experiment $\mathcal{I}$:

$$\mathcal{L}(k, \vec{x}, z) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } \mathcal{S}' \text{ outputs } \texttt{loop}, \\ 0 & \text{otherwise.} \end{cases}$$

We first prove that the probability of $\mathcal{S}'$ outputting `loop` is negligible. That is,

$$\text{Prob}[\mathcal{L}(k, \vec{x}, z) = 1] \le k^{-\omega(1)}. \tag{14}$$

The argument is similar to the one from Claim 21, and thus we only sketch it below for completeness. Let $p_{\mathcal{E}}(k, \vec{x}, z) \overset{\text{def}}{=} \text{Prob}[\mathcal{C}_{\mathcal{E}}(k, \vec{x}, z) = 1]$ be the probability that $\mathcal{A}$

corrupts both parties (equivalently, the probability that $\mathcal{A}_{\text{na}}$ outputs `error`). Similarly, let $p_{\mathcal{I}}(k, \vec{x}, z) \overset{\text{def}}{=} \text{Prob}[\mathcal{C}_{\mathcal{I}}(k, \vec{x}, z) = 1]$ be the probability that $\mathcal{S}'$ corrupts both parties (equivalently, the probability that $\mathcal{S}_{\text{na}}$ outputs `error`). It follows from (12) that

$$\mathcal{C}_{\mathcal{E}} \overset{\text{s}}{\approx} \mathcal{C}_{\mathcal{I}}, \tag{15}$$

and in particular $|p_{\mathcal{E}} - p_{\mathcal{I}}|$ is negligible. We now consider two cases, depending on the ratio between these probabilities. If this ratio is large, say $p_{\mathcal{I}} > 2p_{\mathcal{R}}$, then $p_{\mathcal{I}}$ must be negligible, and therefore the probability of $\mathsf{sim\_both}_{\mathcal{S}'}$ being invoked (let alone outputting `loop`) is negligible. Otherwise ($p_{\mathcal{I}} \leq 2p_{\mathcal{E}}$), the probability of $\mathsf{sim\_both}_{\mathcal{S}'}$ outputting `loop` is negligible, as the probability of getting k successes of a $p_{\mathcal{I}}$-trial before the first success of a $p_{\mathcal{E}}$-trial is negligible. Thus, in both cases we can bound $\text{Prob}[\mathcal{L}(k, \vec{x}, z) = 1]$ by a negligible function of k, concluding the proof of (14).

Next, observe that $\mathcal{A}_{\text{na}}$ behaves exactly as $\mathcal{A}$ as long as player P_{1-i} remains honest. Since $\mathcal{S}'$ runs the simulator $\mathcal{S}_{\text{na}}$ of $\mathcal{A}_{\text{na}}$ (in Step 4) we get that conditioned on the event that P_{1-i} remains honest, the distributions $\mathcal{E}$ and $\mathcal{I}$ are computationally indistinguishable, i.e.,

$$[\mathcal{E} \mid \mathcal{C}_{\mathcal{E}} = 0] \overset{\text{c}}{\approx} [\mathcal{I} \mid \mathcal{C}_{\mathcal{I}} = 0]. \tag{16}$$

On the other hand, conditioned on the event that both players get corrupted, the distributions of $\mathcal{E}(k, \vec{x}, z)$ and $\mathcal{I}(k, \vec{x}, z)$ are statistically indistinguishable. Indeed,

$$\begin{aligned}[\mathcal{E} \mid \mathcal{C}_{\mathcal{E}} = 1] &\overset{\text{d}}{=} [\mathcal{I} \mid \mathcal{C}_{\mathcal{I}} = 1, \mathcal{L} = 0] \\ &\overset{\text{s}}{\approx} [\mathcal{I} \mid \mathcal{C}_{\mathcal{I}} = 1],\end{aligned} \tag{17}$$

where the first transition follows from the description of $\mathcal{S}'$ and the second from (14). We now show our goal (13), using (15), (16), and (17). Consider an arbitrary polynomial-time algorithm $\mathcal{D}$ and arbitrary $c > 0$. From (16) we get that for sufficiently large k and every $(\vec{x}, z)$ we have

$$\begin{aligned}&\text{Prob}[\mathcal{D}(1^k, (\vec{x}, z), \mathcal{E}(k, \vec{x}, z)) = 1 | \mathcal{C}_{\mathcal{E}}(k, \vec{x}, z) = 0] \\ &\qquad \leq \text{Prob}[\mathcal{D}(1^k, (\vec{x}, z), \mathcal{I}(k, \vec{x}, z)) = 1 | \mathcal{C}_{\mathcal{I}}(k, \vec{x}, z) = 0] + k^{-2c},\end{aligned} \tag{18}$$

and from (17) we have

$$\begin{aligned}&\text{Prob}[\mathcal{D}(1^k, (\vec{x}, z), \mathcal{E}(k, \vec{x}, z)) = 1 | \mathcal{C}_{\mathcal{E}}(k, \vec{x}, z) = 1] \\ &\qquad \leq \text{Prob}[\mathcal{D}(1^k, (\vec{x}, z), \mathcal{I}(k, \vec{x}, z)) = 1 | \mathcal{C}_{\mathcal{I}}(k, \vec{x}, z) = 1] + k^{-2c}.\end{aligned} \tag{19}$$

It is also easy to see that (15) implies that for sufficiently large k and every $(\vec{x}, z)$,

$$\text{Prob}[\mathcal{C}_{\mathcal{E}}(k, \vec{x}, z) = 0] \leq \text{Prob}[\mathcal{C}_{\mathcal{I}}(k, \vec{x}, z) = 0] + k^{-2c}. \tag{20}$$

Clearly, $\text{Prob}[\mathcal{D}(1^k, a, \mathcal{E}(k, \vec{x}, z)) = 1]$ is equal to

$$\begin{aligned}&\text{Prob}[\mathcal{D}(1^k, (\vec{x}, z), \mathcal{E}(k, \vec{x}, z)) = 1 | \mathcal{C}_{\mathcal{E}}(k, \vec{x}, z) = 0] \cdot \text{Prob}[\mathcal{C}_{\mathcal{E}}(k, \vec{x}, z) = 0] \\ &\qquad + \text{Prob}[\mathcal{D}(1^k, (\vec{x}, z), \mathcal{E}(k, \vec{x}, z)) = 1 | \mathcal{C}_{\mathcal{E}}(k, \vec{x}, z) = 1] \cdot \text{Prob}[\mathcal{C}_{\mathcal{E}}(k, \vec{x}, z) = 1],\end{aligned}$$

which (by (18), (19), and (20)) is at most

$$\begin{aligned}(\mathrm{Prob}[&\mathcal{D}(1^k,(\vec{x},z),\mathcal{I}(k,\vec{x},z))=1|\mathcal{C}_{\mathcal{I}}(k,\vec{x},z)=0]+k^{-2c})\\ &\cdot(\mathrm{Prob}[\mathcal{C}_{\mathcal{I}}(k,\vec{x},z)=0]+k^{-2c})\\ &+(\mathrm{Prob}[\mathcal{D}(1^k,(\vec{x},z),\mathcal{I}(k,\vec{x},z))=1|\mathcal{C}_{\mathcal{I}}(k,\vec{x},z)=1]+k^{-2c})\\ &\cdot(\mathrm{Prob}[\mathcal{C}_{\mathcal{I}}(k,\vec{x},z)=1]+k^{-2c}),\end{aligned}$$

which is at most

$$\begin{aligned}\mathrm{Prob}[\mathcal{D}(1^k,(\vec{x},z),\mathcal{I}(k,\vec{x},z))=1]+4k^{-2c}+2k^{-4c}\\ \le \mathrm{Prob}[\mathcal{D}(1^k,(\vec{x},z),\mathcal{I}(k,\vec{x},z))=1]+k^{-c}\end{aligned}$$

(for sufficiently large k). Therefore, for sufficiently large k,

$$\mathrm{Prob}[\mathcal{D}(1^k,(\vec{x},z),\mathcal{I}(k,\vec{x},z))=1]-\mathrm{Prob}[\mathcal{D}(1^k,(\vec{x},z),\mathcal{E}(k,\vec{x},z))=1]\le k^{-c}.$$

The same argument, starting from inequality (18), can be applied in a symmetric way (i.e., swapping $\mathcal{E}$ and $\mathcal{I}$). Thus we get

$$\left|\mathrm{Prob}[\mathcal{D}(1^k,(\vec{x},z),\mathcal{E}(k,\vec{x},z))=1]-\mathrm{Prob}[\mathcal{D}(1^k,(\vec{x},z),\mathcal{I}(k,\vec{x},z))=1]\right|\le k^{-\omega(1)}.$$

This completes the proof of Lemma 30. □

From Lemma 30 we get the following.

Theorem 31. *Consider the model with an active adversary, secure channels, and no PEC. Suppose that a two-party protocol π non-adaptively $\mathcal{B}$-securely evaluates f with IT (resp., universal, computational) security and computational emulation. Then π adaptively $\mathcal{B}$-securely evaluates f with IT (resp., universal, computational) security and computational emulation.*

Proof. From Lemma 30 we get that for every efficient (i.e., expected polynomial time) $\mathcal{A}$ and $\mathcal{Z}$ there exists an efficient $\mathcal{S}'$ such that $\mathrm{IDEAL}_{f,\mathcal{S}',\mathcal{Z}} \overset{c}{\approx} \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. Thus, as in the proof of Theorem 29, the only non-trivial part is to prove that the expected number of restarts of $\mathsf{sim\_both}_{\mathcal{S}'}$ procedure is small. Let $p=p_{\mathcal{I}}$, the probability that $\mathcal{S}_{\mathrm{na}}$ outputs `error`. If $p=0$, then $\mathsf{sim\_both}_{\mathcal{S}'}$ is never executed, and thus we are done. Otherwise, the expected number of iterations that we need to wait until c is increased by 1 (after it was set to 1 in Step (a) or increased by 1 last time) is equal to $1/p$. Thus the expected number of iterations until c reaches $k+1$ is at most k/p. Since the probability of $\mathcal{S}'$ entering Step 6 is p, the expected number of iterations of $\mathsf{sim\_both}_{\mathcal{S}'}$ is at most k. Thus we are done. □

Along the same lines one can prove a similar theorem for the statistical indistinguishability.

Theorem 32. *Consider the model with an active adversary, secure channels, and no PEC. Suppose that a two-party protocol π non-adaptively $\mathcal{B}$-securely evaluates f with IT (resp., universal, computational) security and statistical emulation. Then π adaptively $\mathcal{B}$-securely evaluates f with IT (resp., universal, computational) security and statistical emulation.*

Proof. The proof is very similar to the proof of Theorem 31. Specifically, one can prove the following "statistical" version of Lemma 30:

$$\text{if} \quad \mathrm{IDEAL}_{f,\mathcal{S}_{\mathrm{na}}} \overset{\mathrm{s}}{\approx} \mathrm{EXEC}_{\pi,\mathcal{A}_{\mathrm{na}}}, \quad \text{then} \quad \mathrm{IDEAL}_{f,\mathcal{S}',\mathcal{Z}} \overset{\mathrm{s}}{\approx} \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \tag{21}$$

(where $\mathcal{S}'$ is identical to the one in Lemma 30). A proof of (21) may be obtained from the proof of Lemma 30 by replacing computational indistinguishability with statistical indistinguishability and dropping the assumption that $\mathcal{D}$ is computationally bounded. □

2.7.2. *Separation with Insecure Channel (with Output to Adversary)*

This section considers the model where the adversary can eavesdrop the channel between the parties even when he did not corrupt any of them. In this model we show a protocol that is non-adaptively secure, but adaptively insecure, as long as the adversary may receive an output from (or give an input to) the trusted party in the ideal model. The example is of a similar nature as the one in the three-party case (Section 2.6). Again, the example holds for any type of emulation.

Consider two parties: a dealer D and a receiver R. Suppose the adversary can corrupt only D, i.e., the adversary structure $\mathcal{B}$ consists of sets $\{\emptyset, \{D\}\}$. Define function f_2 as follows. The only input that it takes is D's input $x \in \{0, 1\}$. It returns no output for D and it outputs x to R and to the adversary. The protocol π_2 works as follows:

1. D sends x to R.
2. D sends x to R.
3. R outputs what he received in Step 2 (ignoring what he received in Step 1).

Claim 33. *The protocol π_2 non-adaptively, $\mathcal{B}$-securely evaluates f_2 with universal security and perfect emulation.*

Proof. Consider a non-adaptive adversary $\mathcal{A}$. There are two possibilities:

- $\mathcal{A}$ does not corrupt D. In this case $\mathcal{S}$ can simply wait until the output stage in which he learns x. He can then simulate the real-life execution of $\mathcal{A}$ against π_2 (and output the output of the simulated $\mathcal{A}$).
- $\mathcal{A}$ corrupts D. In this case $\mathcal{S}$ learns x at the beginning of the ideal execution and again he can simulate the real-life execution. □

Claim 34. *The protocol π_2 is adaptively insecure for evaluating the function f_2, with either universal, IT, or computational security, against active adversary structure $\mathcal{B}$.*

Proof. We show a real-life adversary $\mathcal{A}$ that cannot be simulated by any ideal-process adversary $\mathcal{S}$. Intuitively the aim of the adversary $\mathcal{A}$ is to make R always output 1. In order to do this it waits until in Step 1 the dealer D sends x. If $x = 1$, then it does nothing, otherwise he corrupts D and makes him send 1 in Step 2. Using this strategy $\mathcal{A}$ corrupts D only if it is necessary (i.e., when $x = 0$).

On the other hand suppose that any ideal-process adversary $\mathcal{S}$ wants to achieve the same. Then he has to decide whether to corrupt D before he sends x to the trusted party. However, at this moment $\mathcal{S}$ has no information on x. Therefore he has to do it always (also when $x = 1$). Thus no ideal-process adversary can simulate $\mathcal{A}$. □

We can conclude with the following.

Theorem 35. *If there are exactly two parties, then in the insecure channels setting, for active adversaries, adaptive security is strictly stronger than non-adaptive security.*

Remark: *Other separating examples.* The above separating example relies heavily on the fact that the simulator obtains an explicit output from the trusted party, even if none of the parties are corrupted. Below we briefly sketch a separating example where the adversary receives no explicit output from the trusted party. However, for this example it is important that the adversary be able to provide the trusted party with its own input, even if none of the parties are corrupted.

Consider the following function f, that takes no inputs from the parties, and an input bit b from the adversary. First, f tosses a coin r. If $r = 1$, then the output is b. Otherwise, the output is set to a new random bit s. In any case, both parties receive the same output bit, and the adversary receives no output.

Next, consider the following sketchy description of a protocol, π. First, the parties run a standard coin-tossing protocol to obtain a public, common random bit r. Next, party P_1 sends (encrypted) a randomly chosen bit b to P_2, and P_2 sends (encrypted) a randomly chosen bit s to P_1. Finally, if $r = 1$, then both parties output b. Otherwise, both parties output s.

It can be readily seen that if the adversary is non-adaptive, then π securely evaluates f.[19] On the other hand, consider the following adaptive adversary $\mathcal{A}$. First, $\mathcal{A}$ monitors the conversation between the parties until the bit r is known. Next, it corrupts P_r and sets the output bit of the computation to 0. Clearly, this adversary cannot be emulated in the (adaptive) ideal process.

Remark: *The setting without input or output to adversary.* We leave open the question whether there exists a separating example for the restricted case where the adversary does not provide the trusted party with input nor does it receive explicit output from the trusted party (that is, the original [C1] definition without the PEC requirement). We note

[19] As mentioned in footnote 2, two-party coin-tossing can only be securely realized under a relaxed definition that allows the adversary to abort, namely, stop the computation at any time, possibly after learning its output. (Technically, this relaxation is achieved by modifying the ideal process so as to allow the simulator to decide whether the uncorrupted party (parties) receive their outputs from the trusted party.) This standard relaxation of the definition in the two-party case was adopted also in [G1].

that if the simulator is required to be a one-pass black-box simulator, then we can in fact show an equivalence.

3. Adaptivity versus Non-Adaptivity in the Augmented Definition

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 [DM]. For additional details, please refer to [DM].

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

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 party consists of all messages he sends and receives. However, since without loss of generality the adversary never sends a message from a corrupted player to another corrupted player, the traffic of a corrupted player consists only of messages exchanged with honest players.

π must specify *input and output functions* that map traffic to input and output values for the function f computed. The *effective inputs* $\hat{x}_1^\pi, \dots, \hat{x}_n^\pi$ are determined by applying the input functions to the traffic of each party up to and including CR. So these values are the ones that parties "commit to" as their inputs. The *effective outputs* $\hat{y}_1^\pi, \dots, \hat{y}_n^\pi$ are determined by applying the output functions to the entire traffic of each party.

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

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

The way $\mathcal{A}$ interacts with the protocol is as follows: in each round, $\mathcal{A}$ sees all messages from honest parties 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 parties, on behalf of the corrupted ones. Note that, in particular, this means that the traffic of a corrupted party consists only of messages sent to/received from parties who are honest at the time the message is sent/received.

The definition calls for the 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 $\mathit{History}(\mathcal{A}, \pi)$. 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 parties $\hat{x}_j^{\mathcal{S}}$. 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 parties and the real x_j's for honest parties. The result is $\hat{y}^{\mathcal{S}} = (\hat{y}_1^{\mathcal{S}}, \dots, \hat{y}_n^{\mathcal{S}})$. O sends the results for corrupted parties back to $\mathcal{S}$.
- If P_j is corrupted in or after CR, then O sends $x_j, a_j, \hat{y}_j$ to $\mathcal{S}$.

The random variable $\mathit{View}(\mathcal{A}, \mathcal{S})$ 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., $\hat{y}^{\mathcal{S}} = f(\hat{x}^{\mathcal{S}}, r)$.

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

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,

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

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

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

Lemma 36. *If π computes f securely, then the input and output functions are such that if P_j remains honest up to* CR, *then $\hat{x}_j^{\pi} = x_j$. If P_j is never corrupted, then $\hat{y}_j^{\pi}$ is the* jth *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 $x_j = \hat{x}_j^{\mathcal{S}}$ and $\mathit{History}(\mathcal{A}_j, \mathcal{S}) \equiv \mathit{History}(\mathcal{A}_j, \pi)$. The second follows from $\mathit{History}(\mathcal{A}_j, \mathcal{S}) \equiv \mathit{History}(\mathcal{A}_j, \pi)$ and the fact that the correlation $\hat{y}_j^{\mathcal{S}} = f(\hat{x}^{\mathcal{S}}, r)_j$ 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*

3.2.1. *Preliminaries*

Our goal in this section is to take a non-adaptively secure protocol and look at how it behaves against an adaptive adversary. To do this, we first have to deal with three preliminary issues:

First, one that concerns the input/output functions: if we are given a protocol that is non-adaptively secure, we are of course also given some input/output functions. However, these functions can only be assumed to be defined on traffic that could occur under a non-adaptive attack, in particular traffic where the corrupted set is constant over time. Under an adaptive attack, in each round the traffic consists (for corrupted players) of messages exchanged with the set of parties that are honest at that particular time, and so the input/output functions we are given are not defined on all such traffic-values. This is resolved as follows: say A is the corrupted set at the end of the CR. Then to compute the effective inputs and outputs, we delete from all traffic-values all messages exchanged between players in A, and then evaluate the (non-adaptive) input/output functions on what remains. The output functions are handled in a similar way. In other words, we pretend that A was corrupted from the beginning and evaluate effective input/outputs accordingly. This is natural and seems to be the only meaningful solution that is also general. We assume throughout that input/output functions are transplanted to the adaptive setting in this way.

Second, 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: a 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 the 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. Note that a halting adversary does not necessarily halt prematurely, it only has the option to do so.

If protocol π is secure in the above definition, then the simulator guaranteed by the definition can also simulate the (possibly partial) history generated by any halting adversary, since no rewinding occurs. Conversely, an ordinary adversary is also a halting one (which happens to never halt prematurely). So we see that protocol π is secure according to 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.

A third a final preliminary issue is that we need one more concept that does not appear in the original definition. To explain this, consider the following two scenarios for a non-adaptively secure protocol π:

- Non-adaptive adversary $\mathcal{A}$ attacks the protocol, corrupting set A.
- Non-adaptive adversary $\mathcal{A}'$ attacks the protocol, corrupting set A' where $A \subset A'$. However, $\mathcal{A}'$ follows the strategy of $\mathcal{A}$, i.e., he lets the parties in $A' \setminus A$ play honestly, ignoring completely their internal data.

For each given set of random coins used in the protocol, all parties make exactly the same moves in the two scenarios. It therefore seems intuitively reasonable that the effective inputs should also be the same. Indeed, this is the case for all known examples of general multi-party computation protocols. However, the original definition does not ensure this: for players in $A' \setminus A$, for instance, the input functions are evaluated on messages exchanged with all players in the first scenario, and in the second only on a smaller set of messages, namely, those exchanged with parties outside A'.

In order to prove that non-adaptive security implies adaptive, we will need that effective inputs are indeed the same in such cases, so we now define a condition on the input functions that precisely ensures this:

Definition 37. Consider a protocol π and any non-adaptive adversary $\mathcal{A}$. Consider any possible partial history H of $\mathcal{A}, \pi$ that runs until the end of the committal round. Let A be the corrupted set in H, and let x be the set of effective inputs defined by H. Let A' be any corruptible set with $A \subset A'$, and let x' be the set of effective inputs obtained by applying the input functions to the traffic obtained by ignoring all messages exchanged between players in A'. The input functions are said to be *consistent* iff it always holds that $x = x'$.

Based on the above discussion, the consistency requirement seems very natural, and indeed one might argue that consistency should be required in the definition. However, this is a quite subjective question. What is important for us is that consistency is a property that can be easily verified for a concrete protocol, and that assuming consistency, non-adaptive security implies adaptive security, as we shall see.

Nevertheless, for completeness, we show that some consistency requirement is necessary by showing an example protocol with inconsistent input functions that separates non-adaptive and adaptive security:

We have three parties called D, P_1, P_2. D has input bit b, and the function to be computed outputs b for P_1, no one else has output. D or P_1 or both can be corrupted. The protocol goes as follows:

1. D chooses at random b_1, b_2 such that $b = b_1 \oplus b_2$. He sends b_1 to P_1 and b_2 to P_2. P_1 sends a 0 to D. This round is also the committal round.
2. P_2 sends b_2 to P_1, who sends a 0 to D and computes the XOR of b_1,b_2 as his result. D sends a 0 to P_1.

The input function is defined as follows: first recall that these functions generally get the traffic of a player P as input, where the traffic is the messages exchanged with all players if P is honest, and otherwise only messages exchanged with honest players. Since in each round, we have a message from D to P_1 and from P_1 to D, we can tell from D's traffic in round 1 and P_1's traffic in round 2, whether $\{D, P_1\}$ or a smaller set is corrupted, just by checking if data exchanged between D and P_1 is present or not in the traffic.

So we can define D's input function as follows: If only D, only P_1, or no one is corrupt, then the input is the XOR of what D sent to P_1 and P_2. If P_1 and D are corrupt, then the input is always 0. The output function of P_1 is defined by: if only P_1, only D, or no one is corrupt, then the output is the XOR of what P_1 received from D and from P_2. Else the output is always 0.

This example is slightly counterintuitive because it is (and must be) based on defining the in and output functions in a somewhat unreasonable way, which is nevertheless allowed by the definition. In particular, we note that these functions are indeed inconsistent according to Definition 37: if D is corrupt and sends 1 to P_1 and 0 to P_2, then the effective input is 1, but if we restrict to what P_2 receives, the effective input is 0. Moreover, the output function is defined such that P_1's effective output is forced to 0 if both D and P_1 are corrupt at round 2. Therefore, an adaptive adversary can corrupt D, make the effective input be 1, and then afterwards corrupt P_1 and create a mismatch with the input. However, because this strategy can only be used by an adaptive adversary, the protocol is still non-adaptively secure.

In more detail, we can argue as follows. The protocol is non-adaptively secure: if no one is corrupted, then security is trivial. If only D is corrupt, then the simulator only has to generate the 0's sent to D in rounds 1 and 2, and the effective inputs and outputs match as they should. If only P_1 is corrupt, then the simulator sends a random bit to P_1 on behalf of D in the first round, gets the result from the oracle, and sends a bit to P_1 on behalf of P_2 in the second round such that the XOR of what P_1 has seen is the right result (and a 0 on behalf of D). If P_1 and D are corrupt, then the simulator looks at what D sends to P_2 in round 1 and sends this to P_1 in round 2. Effective input and output are always 0 and in particular are equal as they should be.

However, the protocol is adaptively insecure: consider an adversary who corrupts D from the start and sends bits b_i to P_i, such that $b_1 \oplus b_2 = 1$. Then after the CR he corrupts P_1. Clearly, in an attack on the real protocol, the effective input of D is 1, whereas the effective output of P_2 is 0. Hence any simulator is forced to to give 1 as the effective input to the oracle, who will compute 1 as output, so this never matches the effective output of the real protocol.

3.2.2. *The Equivalence Proof*

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,

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

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 38. *Let π be a protocol which is non-adaptively secure under the augmented definition, and which has consistent input functions. Then π is adaptively secure.*

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 coins $r_{\mathcal{A}}$ and auxiliary input α:[20]

[20] 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.

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 learned from corrupting P_j (and fresh random input).
2. Start executing the protocol, initially letting the parties 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 party 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 parties 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 parties in B, whereas if u ends before the protocol is complete, $Aview(u)$ ends in the some round where $\mathcal{A}$ requested corrupting some party 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 coins 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.

We first give an intuitive explanation of the idea behind the simulation: From the beginning, $\mathcal{A}$ has not corrupted any parties. So we can create the start of a history by running ($\mathcal{A}_\emptyset$, S_0) (recall that $\mathcal{A}_\emptyset$ runs $\mathcal{A}$ "in the background"). This will stop as soon as $\mathcal{A}$ corrupts the first party P_j. Say this happens in round i. Let v be the view of $\mathcal{A}$ we obtain from this. Recall that $\mathcal{S}_0$ provides perfect emulation. This means that in real life when $\mathcal{A}$ attacks π, we could (with the same probability) obtain a history where, up to round i, $\mathcal{A}$ has seen view v and all parties including P_j have been honest.

Now, by construction of $\mathcal{A}_{\{P_j\}}$ this same history up to round i can also be realized by $\mathcal{A}_{\{P_j\}}$ attacking F: the only difference is that from the beginning $\mathcal{A}_{\{P_j\}}$ and not the jth party runs the honest algorithm of P_j. Again by assumption on $\mathcal{S}_0$, the history can also be realized by $\mathcal{A}_{\{P_j\}}$ interacting with $\mathcal{S}_0$.

We can therefore (by exhaustive search over the random coins) generate a random history of $\mathcal{S}_0$ interacting with $\mathcal{A}_{\{P_j\}}$, conditioned on the event that the view v for $\mathcal{A}$ is produced in the first i rounds (and, moreover, this can be done without rewinding $\mathcal{A}$). This process may be inefficient, but this is no problem since we consider IT security here. Once we succeed, we let ($\mathcal{S}_0, \mathcal{A}_{\{P_j\}}$) continue to interact until they halt, i.e., we extend the history until the protocol is finished or $\mathcal{A}$ corrupts the next party (say $P_{j'}$). In the former case we are done, and otherwise we continue in the same way with $\mathcal{A}_{\{P_j, P_{j'}\}}$.

Once we finish the CR, the effective inputs will be determined, and we will get resulting outputs from the oracle. Note here that since we only consider one-pass black-box simulation, we will never need to rewind back past the CR, which might otherwise create problems since then A could change its mind about the effective inputs. Thus the one-pass black-box requirement is also essential for the proof.

Here follows a more formal description of the algorithm of $\mathcal{S}$:

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 parties 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 may 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 provide the access it needs to an oracle (which we call O_0 to distinguish from the oracle O that $\mathcal{S}$ uses), $\mathcal{A}_B$ also needs black-box access to $\mathcal{A}$. We describe how to emulate O_0 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 coins 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 parties in B from $\mathcal{S}_0$ and send these to $\mathcal{A}_B$. Now we simply want to run the algorithm of $\mathcal{A}_B$ to compute the actions in this round of parties in B. However, 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 the 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 2(a) (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 $\mathit{Aview}(u)$, 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 party 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 $w = \mathit{Aview}(u)$ 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 the next step.
4. If we reach this step, w ends prematurely because $\mathcal{A}$ requested corrupting 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 Theorem 38 it is clearly enough to show the following:

Claim. *For any (non-halting) adversary $\mathcal{A}$, any fixed random coins and auxiliary input for $\mathcal{A}$, and for any fixed input x and random coins r for the oracle O, the algorithm for $\mathcal{S}$ terminates; furthermore, conditioned on the data we fixed, $\mathit{History}(\mathcal{A}, \pi) \equiv \mathit{History}(\mathcal{A}, \mathcal{S})$.*

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 39. *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 $\mathit{History}(\mathcal{A}_B, \mathcal{S}_0)$, 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 $\mathit{Aview}(u)$.*

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 the 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 party $P_j \notin B'$. 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. 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 ith 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 ith 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 the prefix, and otherwise query the real adversary $\mathcal{A}$. For the simulation of O_0, we split 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 the first time 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 to the end of or beyond the CR, and determines some effective inputs $\hat{x}'$, namely, the input functions applied to v' for parties in B' and the real inputs for the other parties. Moreover, v' is a prefix of v, so if we let u be the current history produced by $\mathcal{S}_0$ when the query to O_0 is made, it is clear that u is consistent with v' in the sense that $Aview(u)$ equals v' truncated after the CR.

  Now, u determines some effective inputs of parties in B', from messages they exchange with parties outside B, since this is the currently corrupted set. Similarly, v' determines some effective inputs of parties in B', from messages they exchange with parties outside B'. However, since u is consistent with v', $B' \subset B$ and by the consistency of the input functions, these two sets of effective inputs are the same. All parties outside B' play honestly until the end of the CR, and so their effective inputs equal the real inputs in all cases, by Lemma 36. Hence the inputs on which the oracle 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 40. *Fix any corruptible set B. Let $Distr_B$ be the distribution obtained from the distribution of $History(\mathcal{A}, \pi)$ by truncating every history such that it ends at the* *first point where $\mathcal{A}$ corrupts a party not in B (no truncation if $\mathcal{A}$ never corrupts a party 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 $\mathcal{A}_B$. $\square$

We now return to the proof of the main 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 ith party (no truncation if $\mathcal{A}$ corrupts less than i parties). Then $Distr_{n+1}$ is the distribution of $History(\mathcal{A}, \pi)$, since $\mathcal{A}$ cannot corrupt more than the total number of parties. Therefore our claim follows once we show the following lemma:

Lemma 41. *Running the algorithm for $\mathcal{S}$ for at most i iterations produces a history with $Distr_i$ as distribution, for any $i \geq 1$.*

Proof. Some notation first. Let X be any of the distributions of histories we considered so far—so X may be, for instance, $Distr_B$, $Distr_i$, or $History(\mathcal{A}_B, \mathcal{S}_0)$; and let v be a possible (partial) view of $\mathcal{A}$. Then $X(v)$ means X conditioned on the event that v is a prefix of the view of $\mathcal{A}$ specified by an outcome of X.

To prove the lemma, we use induction on i. The basis of the reduction ($i = 1$) follows immediately from Lemmas 39 and 40 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 $i - 2$ parties. In the cases where the ith 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 ith 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 party. Note that we actually only have to focus on the value of v, since v, being the complete view of $\mathcal{A}$, determines the values of $B, a_B, x_B, \hat{y}_B^{\mathcal{S}}$.

We therefore only have to show that this ith iteration of $\mathcal{S}$ will produce a history that is distributed according to $Distr_i(v)$.

It is 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 party or completes the protocol.

Now, by Lemma 40, $Distr_B(v) = History(\mathcal{A}_B, \mathcal{S}_0)(v)$ (the distribution produced by $\mathcal{S}_0$ interacting with $\mathcal{A}_B$ when we condition on v). Finally by Lemma 39, $History(\mathcal{A}_B, \mathcal{S}_0)(v)$ in turn equals the distribution produced by the ith iteration of $\mathcal{S}$, when that iteration starts from values $B, v, a_B, x_B, \hat{y}_B^{\mathcal{S}}$. $\square$

Acknowledgment

We are grateful to an anonymous referee for comments and suggestions that greatly improved this paper.

References

[B1] D. Beaver, Secure Multi-Party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority, *J. Cryptology*, vol. 4, pages 75–122, 1991.

[B2] D. Beaver, Plug and Play Encryption, *Proc. CRYPTO '97*, pages 75–89, 1997.

[B3] P. Billingsley, *Probability and Measure*, 2nd edition, Wiley, New York, 1986.

[BC] G. Brassard and C. Crepeau, Nontransitive Transfer of Confidence: A Perfect Zero-Knowledge Interactive Protocol for SAT and Beyond, *Proc. 27th FOCS*, pages 188–195, 1986.

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

[BH] D. Beaver and S. Haber, Cryptographic Protocols Provably Secure Against Dynamic Adversaries, *Proc. Eurocrypt '92*, pages 307–323, 1992.

[C1] R. Canetti, Security and Composition of Multiparty Cryptographic Protocols, *J. Cryptology*, vol. 13, no. 1, pages 143–202, 2000. On-line version at http://philby.ucsd.edu/cryptolib/1998/98-18.html.

[C2] R. Canetti, Universally Composable Security: A New Paradigm for Cryptographic Protocols, *Proc. 42nd FOCS*, pages 136–145, 2001. Full version available at http://eprint.iacr.org/2000/067.

[CCD] D. Chaum, C. Crepeau, and I. Damgaard, Multi-Party Unconditionally Secure Protocols, *Proc. 20th STOC*, pages 11–19, 1988.

[CDG] D. Chaum, I. Damgaard, and J. van de Graaf, Multi-Party Computations Ensuring Privacy of Each Party's Input and Correctness of the Result, *Proc. CRYPTO '87*, pages 87–119, 1987.

[CDM] R.Cramer, I. Damgaard, and U. Maurer, General Secure Multi-Party Computation from Any Linear Secret-Sharing Scheme, *Proc. EuroCrypt* 2000, pages 316–334, 2000.

[CFGN] R. Canetti, U. Feige, O. Goldreich, and M. Naor, Adaptively Secure Computation, *Proc. 28th STOC*, pages 639–648, 1996. Fuller version in MIT-LCS-TR #682, 1996.

[CHP] D. Chaum, E. v. Heijst, and B. Pfitzmann, Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer, *Proc. CRYPTO '91*, pages 470–484, 1991.

[CK] B. Chor and E. Kushilevitz, A Zero-One Law for Boolean Privacy, *SIAM J. Disc. Math.*, vol. 4, pages 36–47, 1991. Preliminary version in *Proc. 21st STOC*, pages 62–72, 1989.

[CS] R. Cramer and V. Shoup, A Practical Public-Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack, *Proc. CRYPTO '98*, pages 13–25, 1998.

[DDN] D. Dolev, C. Dwork, and M. Naor, Non-Malleable Cryptography, *SIAM J. Computing*, vol. 30, no. 2, pp. 391–437, 2000. Preliminary version in *STOC'91*.

[DM] Y. Dodis and S. Micali, Parallel Reducibility for Information-Theoretically Secure Computation, *Proc. CRYPTO* 2000, pages 74–92, 2000.

[DN] I. Damgaard and J. Nielsen, Improved Non-Committing Encryption Schemes Based on a General Complexity Assumption, *Proc. CRYPTO* 2000, pages 432–450, 2000.

[G1] O. Goldreich, Secure Multi-Party Computation, 1998. Available at http://philby.ucsd.edu.

[G2] O. Goldreich, *Foundations of Cryptography*, Cambridge University Press, Cambridge, 2001.

[GL] S. Goldwasser and L. Levin, Fair Computation of General Functions in Presence of Immoral Majority, *Proc. CRYPTO '90*, pages 77–93, 1990.

[GM] S. Goldwasser and S. Micali, Probabilistic encryption, *J. Comput. Systems Sci.*, vol. 28, no. 2, pages 270–299, April 1984.

[GMW] O. Goldreich, S. Micali, and A. Wigderson, How to Play Any Mental Game, *Proc. 19th STOC*, pages 218–229, 1987.

[K] E. Kushilevitz, Privacy and Communication Complexity, *SIAM J. Discrete Math.*, vol. 5, no. 2, pages 273–284, 1992. Preliminary version in *Proc. 29th FOCS*, 1989.

[KKMO] J. Kilian, E. Kushilevitz, S. Micali, and R. Ostrovsky, Reducibility and Completeness in Private Computations, *SIAM J. Computing*, vol. 29, pages 1189–1208, 2000. Preliminary versions by Kilian in *Proc. 23rd STOC*, 1991, and by Kushilevitz, Micali, and Ostrovsky in *Proc. 35th FOCS*, 1994.

[MR] S. Micali and P. Rogaway, Secure Computation, unpublished manuscript, 1992. Preliminary version in *Proc. CRYPTO '91*, pages 392–404, 1991.

[P] T. P. Pedersen, Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing, *Proc. CRYPTO '91*, pages 129–140, 1991.

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

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

[S] A. Sahai, Non-Malleable, Non-Interactive Zero Knowledge and Adaptive Chosen Ciphertext Security, *Proc*. 40*th FOCS*, pages 543–553, 1999.

[Y1] A. Yao, Protocols for Secure Computation, *Proc*. 23*rd FOCS*, pages 160–164, 1982.

[Y2] A. Yao, How to generate and exchange secrets, *Proc*. 27*th FOCS*, pages 162–167, 1986.