# Studies in Secure Multiparty Computation and Applications

Thesis for the Degree of

DOCTOR of PHILOSOPHY

by

**Ran Canetti**

Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science

Submitted to the Scientific Council of
The Weizmann Institute of Science
Rehovot 76100, Israel

June 1995
Revised: March 1996

# Acknowledgements

First, a very special thanks is due to Oded Goldreich, my advisor. On top of being an expert on experts, and a dear friend, he is a devoted advisor, far beyond the ordinary.

Oded has the special property of always searching for the crux of any matter, and disgustedly ridding himself of the rest. Once he sets his mind to a particular goal, he is thoroughly and uncompromisingly dedicated. This, together with his sharpness, his peculiar sense of humor, and his natural good-heartedness, make him a remarkable person indeed.

My interaction with Oded deeply affected my approach to research, and to life in general. Time and again, his unconventional approach first looks odd, and after some thought it becomes clear that his is the direct, simple and natural approach. It also becomes totally unclear how I ever thought otherwise.

His colorful and creative feedback on my writing style has made each one of my drafts a museum piece. His feedback also spiced up my fearful anticipation of their return (which has happened at an amazing speed). I am also thankful for the practical training I received in the art of dodging flying shoes.

During my years of study, I have made some special acquaintances from whom I have learned a lot. Among these let me mention Benny Chor, Amir Herzberg (who is the most practical person I know), Hugo Krawzcyk, and Yishay Mansour.

I have also enjoyed working with, and learned a lot from many many people. A very partial list includes Amotz Bar-Noy, Amos Beimel, Mihir Bellare, Cynthia Dwork, Guy Even, Uri Feige, Shafi Goldwasser, Sandy Irani, Yoram Moses, Moni Naor, Tal Rabin, Baruch Schieber, and Moti Yung.

Next I wish to thank my collaborators on the results that make up this thesis. I have enjoyed, and learned a lot from interacting with them. The chapter on adaptive security in the computational setting (Chapter 3) describes joint work with Uri Feige, Oded Goldreich and Moni Naor. The chapter on asynchronous secure computation (Chapter 4) describes joint work with Oded Goldreich and Michael Ben-Or. The chapter on asynchronous Byzantine agreement (Chapter 5) describes joint work with Tal Rabin. The chapter on Proactive Security (Chapter 6) describes joint work with Amir Herzberg.

I have not found an appropriate list for Dana Ron, but I still thank her for her company, and for sharing a bottle of wine in countless dinners...

A final thanks is to Ronitt, who besides being my source of happiness and sound support, has taught me more than a couple of things about research and life.

.

# Abstract


Consider a set of parties who do not trust each other, nor the channels by which they communicate. Still, the parties wish to correctly compute some common function of their local inputs, while keeping their local data as private as possible. This, in a nutshell, is the problem of secure multiparty computation. This problem is fundamental in cryptography and in the study of distributed computations. It takes many different forms, depending on the underlying network, on the function to be computed, and on the amount of distrust the parties have in each other and in the network.

We study several aspects of secure multiparty computation. We first present new definitions of this problem in various settings. Our definitions draw from previous ideas and formalizations, and incorporate aspects that were previously overlooked.

Next we study the problem of dealing with **adaptive** adversaries. (Adaptive adversaries are adversaries that corrupt parties during the course of the computation, based on the information gathered so far.) We investigate the power of adaptive adversaries in several settings. In particular, we show how to construct adaptively secure protocols for computing any function in a **computational** setting, where the communication channels can be tapped by the adversary, and secure communication is achieved by cryptographic primitives based on the computational limitations of the adversary. We remark that the problem of dealing with adaptive adversaries in a computational setting was considered to be a hard open problem.

Next, we initiate a study of secure multiparty computation in **asynchronous** networks. We consider a completely asynchronous network where the parties are connected via secure channels. In this setting, we present appropriate definitions and construct protocols for securely computing any function. We present a detailed proof of security of our protocols.

In the same asynchronous setting, we apply ideas and techniques of secure multiparty computation to a classical problem in the field of distributed computing, namely the problem of reaching agreement in the presence of Byzantine faults. We present the first asynchronous **Byzantine Agreement** protocol with optimal resilience (i.e., an adversary may corrupt up to $\lceil \frac{n}{3} \rceil - 1$ of the $n$ parties) and polynomial complexity.

Finally we address the problem of maintaining the security of computer systems in the presence of repeated, however transient break-ins. We present a new approach for dealing with this problem. Using our approach, we show how systems can automatically recover from transient break-ins. We introduce mechanisms for maintaining the security of internal data of parties. We use secure multiparty computation as a formal setting for developing and analyzing our mechanisms.

# Table of Contents

CHAPTER 1

# Introduction

Consider a set of parties who do not trust each other, nor the channels by which they communicate. Still, the parties wish to correctly compute some common function of their local inputs, while keeping their local data as private as possible. This, in a nutshell, is the problem of secure multiparty computation. This problem takes many different forms depending on the underlying network, on the function to be computed, and on the amount of distrust the parties have in each other and in the network.

The problem of secure multiparty computation is fundamental in cryptography, as well as relevant to practical cryptographic applications (as demonstrated in the sequel). In particular, almost any known cryptographic setting and problem can be viewed as a special case of this general problem (e.g., encryption, authentication, commitment, signatures, zero-knowledge, and many others). Thus, secure multiparty computation may serve as a general, uniform paradigm for the study of most of cryptography. Furthermore, understanding secure multiparty computation is fundamental in the study of distributed systems in general.

The parties' distrust in each other and in the network is usually modelled via an adversary that has control over some of the parties, and perhaps also over the communication media, or channels. (We call parties controlled by the adversary corrupted.) Many different adversary types (or adversary models) may be considered, each modelling different problems, or addressing a different setting. The requirements from solutions to secure computation problems, as well as the techniques used, differ considerably with the adversary models. In order to be able to present the work done in this field (and, in particular, our work), we briefly sketch some prominent parameters defining adversary models.

**Computational power:** We distinguish between adversaries that are computationally unbounded and adversaries restricted to probabilistic polynomial time (PPT). We remark that throughout this work we assume that the uncorrupted parties are restricted to PPT.

**Control over the communication:** We distinguish three levels of control over the channels (or, alternatively, three levels of abstraction of the channel security). In the most abstract setting the adversary has no access to the channels. That is, each two uncorrupted parties communicate securely without the adversary hearing or affecting the communication. This is the secure channels assumption. Alternatively, we may assume that the adversary can hear all the communication among all parties. Still, the adversary cannot alter the communication. This is the insecure channels assumption. Lastly, if the channels are unauthenticated then the adversary has full control over the communication. That is, on top of hearing the communication the adversary can delete, generate and modify messages at wish. (We sometimes call insecure channels authenticated.)

**Synchrony:** In a synchronous network all parties have a common, global clock. All messages are sent on a clock 'tick', and are received at the next 'tick'. In an asynchronous network no global clock exists. Furthermore, arbitrary (however finite) time may lapse between the sending and receipt of a message. (In particular messages may be received in an order different than the order of sending.) We remark that, although often taken as a parameter of the network, synchrony may be considered as a parameter of the adversary. In particular, setting the actual delays of the messages may be naturally considered as an additional power given to the adversary.

**Number of corrupted parties:** We limit the number of corrupted parties at any given time. An adversary is $t$-limited if at any given time at most $t$ parties are corrupted. A protocol is $t$-resilient if it meets it specifications in the presence of $t$-limited adversaries. $t$-resilient protocols for secure computation are also called $t$-secure.

**Control over corrupted parties:** We distinguish between eavesdropping adversaries that only gather information and do not alter the behavior of corrupted parties, and Byzantine adversaries that may alter the behavior of the corrupted parties in an arbitrary and coordinated way. In asynchronous networks it makes sense to consider also Fail-Stop adversaries. Here the only diversion from the protocol allowed to the corrupted parties is to "crash", that is to stop sending message at some time during the computation. (A crushed party may not resume sending messages.) For security considerations, we assume that faulty parties continue receiving messages and have an output.

**Adaptivity:** By adaptivity we mean the way in which the corrupted parties are chosen. It is simplest to assume that the set of corrupted parties is arbitrary but fixed (ofcourse, the uncorrupted parties do not know the identity of the corrupted parties). We call such adversaries non-adaptive. Alternatively, we may let the adversary choose which parties to corrupt as the computation proceeds, based on the information gathered so far. Once a party is corrupted it remains so for the rest of the computation. We call such adversaries adaptive.

Lastly, we may let the adversary corrupt, in an adaptive way, a different set of parties at different times during the computations. (Here, parties that were once corrupted may become uncorrupted again, and there is no limit on the total number of parties that were corrupted at some time or another during the computation.) Such adversaries are called mobile.

In the sequel we often use the following terminology. A secure channels setting refers to computationally unbounded adversaries with secure channels. A computational setting refers to adversaries restricted to PPT, and insecure channels. Other parameters (e.g., synchrony, adaptivity) may vary.

A great deal of work has been done studying secure multiparty computation. This body of work may be divided to three major efforts as follows. First, ingenious protocols have been devised for securely computing, in several adversary models out of the ones characterized above, *any* function whose inputs are distributed among the parties. Next, ideas and techniques from multiparty computation have been successfully applied to other problems in cryptography and in computer science in general. Another effort aims at finding "good" definitions for secure multiparty computation in different adversary models. By "good" we mean definitions that correctly capture our intuitive notions, and are coherent and usable. Surprisingly, such definitions have proved to be very evasive. In particular, in spite of considerable efforts (and some appreciable results), published definitions have several shortcomings, described in the sequel.

We remark that the current state of knowledge regarding secure multiparty computation is somewhat paradoxical: Constructions solving 'any protocol problem' exist, accompanied by neither a proof nor a good definition. The value and validity of most of these constructions is intuitively obvious. However, precise definitions and proofs are essential. To the best of our knowledge, the first work that contains a definition, a protocol and a full proof for a secure multiparty computation problem is the asynchronous secure computation work presented here. In another part of our work we demonstrate a fundamental problem, namely the adaptive security problem, where precise definition and analysis result in pinpointing the difficulties and lead to a long-sought solution (for some important special cases).

In this work we address several different aspects of secure multiparty computation, ranging over the above efforts. In the rest of the introduction we motivate and overview our contributions, as follows. We first briefly overview, in Section 1.1, some of the prior work in this area that is directly relevant to our work. Next, for $n \in \{2, \ldots, 6\}$, Section 1.$n$ in the introduction motivates Chapter $n$ in the sequel.

In Chapter 2 we address the problem of defining secure multiparty computation. We present new formulations of definitions of secure multiparty computation in different adversary models. In Section 1.2 we briefly and roughly sketch our new ideas, as well as overview some prior definitions on which we base ours.

In Chapter 3 we investigate the extra power of adaptive adversaries over non-adaptive adversaries. We consider two cases: the secure channels setting and the computational setting. (In the secure channels setting, we distinguish two very different variants.) We shed new light on this problem and re-evaluate some common beliefs. In particular, we show how to construct adaptively secure protocols for computing any function in a computational setting.

Next we study security in asynchronous networks. In Chapter 4 we define a notion of secure multiparty computation in asynchronous networks. We also show how any functions can be securely computed in our setting. We present detailed proofs of security of our constructions. In Chapter 5 we apply ideas and techniques of secure multiparty computation to the classical Byzantine agreement problem. We present the first asynchronous Byzantine Agreement protocol with optimal resilience (our protocol is $(\lceil \frac{n}{3} \rceil - 1)$-resilient) and polynomial complexity.

The notion of asynchronous verifiable secret sharing (AVSS) plays a key role in both Chapters 4 and 5. In each of the two chapters we present a very different construction of AVSS. The two constructions have different resilience properties. We elaborate on the differences in the sequel.

In Chapter 6 we address the problem of maintaining the security of computer systems in the presence of repeated, however transient break-ins. We present a new approach, called the **proactive** approach, for dealing with this problem. Using our approach, we show how systems can automatically recover from transient break-ins. We use secure multiparty computation as a formal setting for developing and analyzing our mechanisms.

We conclude by presenting, in Chapter 7, a brief personal view of the work and its merits. We also propose some directions for further research.

## 1.1 Some prior and related work

We briefly overview three works that are immediately relevant to our work. We also mention some other prominent works in secure multiparty computation. (We present other relevant works in the sequel.)

The problem of secure computation was first formulated by Yao for the two-party case in 1982 [Y1]. Five years later, Goldreich, Micali and Wigderson showed how to securely compute any function whose inputs are divided among the parties, in a **computational** setting [GMW]. That is, in [GMW] a synchronous network of $n$ parties is considered, where the communication channels are insecure, and the parties, as well as the adversary, are restricted to PPT. In this model they showed, under the assumption that one-way functions with trapdoor exist, how to construct $n$-secure protocols for computing *any* function, in the presence of eavesdropping adversaries. In the case of Byzantine adversaries they show $(\lceil\frac{n}{2}\rceil - 1)$-secure protocols for computing any function. Their protocols can be shown secure in the presence of non-adaptive adversaries.

Ben-Or, Goldwasser and Wigderson [BGW] (and, independently, Chaum, Crepeau and Damgard [CCD]) study secure multiparty computation in the secure channels setting. They show that: (a) If the adversary is eavesdropping then there exist $(\lceil\frac{n}{2}\rceil - 1)$-secure protocols for computing any function. (b) if the adversary is Byzantine, then any function can be $(\lceil\frac{n}{3}\rceil - 1)$-securely computed. Furthermore, they show that these bounds on the number of corruptions are tight. These protocols can be shown secure in the presence of non-adaptive adversaries. Adaptive security (i.e., security in the presence of adaptive adversaries) is provable in certain variants of this setting. We elaborate on this point in Chapter 3.

Goldwasser and Levin build on a long sequence of works studying the case of Byzantine adversaries limited to PPT, where a majority of the parties may be corrupted [GwL]. Chor and Kushilevitz study secure multiparty computation with corrupted majority of the parties in the secure channels setting [CK]. Goldreich, Goldwasser and Linial study secure multiparty computation in the presence of insecure channels and computationally unlimited adversaries [GGL]. Ostrovsky and Yung study secure multiparty computation in the presence of secure channels and *mobile* adversaries [OY]. Micali and Rogaway [MR], and also Beaver [Be], propose definitions for secure multiparty computation in the secure channels setting, in the presence of adaptive adversaries.

## 1.2 Defining secure multiparty computation

We attempt at formulating coherent, lean, and usable definitions, that adequately capture our intuitive notion of security of protocols for multiparty computation in different adversary models. In the sequel we make extensive use of our definitions, when proving security of our protocols. Our definitions build on previously known ideas. We first present these ideas, as well as a brief critique and comparison of relevant works. Next we present our interpretation and new ideas.

Micali and Rogaway [MR], and independently Beaver [Be] introduced the following methodology for defining secure multiparty computation (or, more specifically, secure evaluation of a function whose inputs are distributed among the parties). First an **ideal model** for secure multiparty computation is formulated. A computation in this ideal model, described below, captures "the highest level of security we can expect from multiparty function evaluation". Next we require that executing a secure protocol $\pi$ for evaluating some function of the parties' inputs in the actual, real-life setting is "equivalent" to evaluating the function in the ideal model.

A computation in this ideal model proceeds as follows. First an **ideal-model adversary** chooses to corrupt a set of parties (either adaptively or non-adaptively), learns their inputs, and possibly modifies it. Next all parties hand their (possibly modified) inputs to an incorruptible **trusted party**. Next the trusted party computes the expected output (i.e., the function value) and hands it to all parties. The uncorrupted parties output whatever they receive from the trusted party. The corrupted parties output some arbitrary function of their joint inputs, random inputs, and the value received from the trusted party. Loosely speaking, executing $\pi$ in the real-life setting is said to be "equivalent" to evaluating the function in the ideal model, if the same effect on the computation achieved by a real-life adversary can be also achieved by an ideal-mode adversary.

The definitions in [MR] and [Be] differ in the notion of equivalence of computations. In [MR] the ideal model remains as a motivating intuition behind their definition, and is not explicitly used as in ours. Instead they specify a list of requirements from a protocol (e.g., correctness of the output, simulatability of the adversary view, and the ability to extract each party's input and output from the transcript of the communication) and informally claim that these requirements amount to equivalence of the real-life computation to a computation in the ideal model.

In [BE] a different approach is pursued. First a general notion of comparing security of protocols is formulated, as follows. Consider two protocols $\alpha$ and $\beta$ for computing the same function. Essentially, protocol $\alpha$ is at least as secure as protocol $\beta$ if an adversary attacking $\alpha$ cannot affect the outputs of the parties more than an adversary attacking $\beta$. (Here some technical **interface** algorithms are used when $\alpha$ and $\beta$ operate in different adversary models.) Next, a protocol for evaluating a function is secure if it is at least as secure as the trivial protocol for evaluating the function in an ideal model similar to ours. Here the interface, combined with the adversary in the ideal model, plays a similar role to our ideal-model adversary that operates via black-box simulation. We remark that, although their approach is general, both Micali and Rogaway and Beaver formalize their definitions only in the private channels setting.

Our definitions use ideas from both works. We first define an ideal-model adversary, based on the above description. Next we require that for *any* real-life adversary $\mathcal{B}$ attacking a secure protocol $\pi$ there *exists* an ideal-model adversary $\mathcal{A}$ that has the same effect on the computation as $\mathcal{B}$, *even though $\mathcal{A}$ operates in the ideal model.* That is, on any inputs for the parties, the random variable describing the outputs of all parties in the ideal model is distributed "similarly" to the random variable describing the outputs of all parties in the real-life model. (The particular notion of similarity, e.g. perfect equality or computational indistinguishability, depends on the specific adversary model.) We emphasize that the complexity of the ideal-model adversary $\mathcal{A}$ should be comparable to the complexity of the real-life adversary $\mathcal{B}$. We elaborate on this point in Chapters 2 and 3. Our definitions incorporate an important additional concern that was left unnoticed by previous definitions. We present this concern in Section 1.2.1 below.

We believe that our simple and straightforward notion of equivalence of computations, which does not restrict the operation of the ideal-model adversary, suffices for defining security. In the sequel we also use a more restrictive and technical notion of equivalence of a real-life computation to a computation in the ideal model. Here we limit the ideal-model adversary $\mathcal{A}$ to black-box simulation of the real-life adversary. (A more precise definition of this simulation is presented in the sequel.) We note that our notion of black-box simulation is less restrictive than the [MR] notion.

We remark that Goldwasser and Levin take a slightly different approach at defining secure multiparty computation [GwL]. First they extract the 'inevitable advantages' of the adversary in the ideal model (we briefly sketch these 'inevitable advantages' below). Next they say that a protocol is **robust** if for any adversary, there exists an "equivalent" adversary that is limited to these 'inevitable privileges', and that has the same effect on the computation. Their notion of robustness of protocols has the advantage that it is independent of the specific function to be computed (except for some technical subtleties ignored in this presentation).

The 'inevitable privileges' of the adversary, extracted from the ideal model, can be sketched as follows. First, the adversary may choose to corrupt parties (either adaptively or non-adaptively). Next, if the adversary is Byzantine then the inputs of the corrupted parties may be modified. (However, this is done without knowledge of the inputs of the uncorrupted parties). Next, the adversary may learn the specified outputs of the corrupted parties. This may inevitably reveal some information on the inputs of the uncorrupted parties. Furthermore, if the adversary is adaptive then it can corrupt parties, after the computation is completed, based on the output of the computation.[1]

### 1.2.1 On semi-honest parties

The problem of secure computation in the presence of adaptive adversaries is intimately related to the following concern. In a distributed scenario where no party is thoroughly trusted, there is no reason to believe that even uncorrupted parties follow their protocols to the dot. Honest parties internally deviate from their protocol in many real-life scenarios, such as users that keep record of their passwords, stock-market brokers that keep records of their clients' orders, *operating systems* that "free" old memory instead of erasing or

[1]It turns out that if a majority of the parties are corrupted then, in addition to the privileges described above, the adversary cannot be prevented from "quitting early", i.e. disrupting the computation at any time. However, this is done without knowing the output with more certainty than the uncorrupted parties. We do not discuss situations of corrupted majority in this work.

take periodic snapshots of the memory (for error recovery purposes), and computers that use pseudorandom generators as their source of randomness instead of truly random bits. Consider for example a protocol in which party $A$ is instructed to choose a random number $r$ for party $B$, hand $r$ to $B$, and then to *erase* $r$ from its own memory. Can $B$ be certain that $A$ no longer knows $r$? Furthermore, can $A$ now convince a third party (or an adversary that decides to corrupt $A$) that he no longer knows $r$?

For this purpose we introduce the notion of a **semi-honest** party.[2] Such a party "appears as honest" (i.e., seems to be following its protocol) from the point of view of an outside observer; however, internally it may somewhat deviate from his protocol. For instance, a semi-honest party may fail to erase some internal data, or use randomness not as instructed. (However, semi-honest parties do *not* collaborate.) We wish to have protocols that are secure even when parties are not thoroughly trusted, or in other words when the uncorrupted parties are semi-honest rather than honest. That is, say that a protocol $\pi'$ is a **semi-honest protocol** for a protocol $\pi$ if a party running $\pi'$ "appears as" an honest party running $\pi$. (We define this notion more precisely in Chapter 2.) When the parties are not thoroughly trusted we want the requirements from $\pi$ to be satisfied even if the uncorrupted parties are running some semi-honest protocol for $\pi$. In the sequel, we consider several alternative notions of semi-honest parties, differing in the "amount of allowed internal deviation" from the protocol.

We distinguish three types of semi-honest behaviour. The most 'benign' (and hardest to prevent) is simply not erasing internal data. We call such parties **non-erasing**. Alternatively, one may consider parties that internally deviate from the protocol in an arbitrary way, as long as the deviation is undetectable by any external test (that represents a collaboration of the other parties). We call such parties **honest-looking**. Finally, we consider parties that deviate from their protocols in a way that is undetectable only by parties *running the protocol.* Such parties are called **weakly honest**. We elaborate on (and present definitions of) the three types of semi-honest parties in Chapter 2.

We remark that the difference between computations in the presence of totally honest parties and computations in the presence of semi-honest parties becomes evident only in the presence of adaptive adversaries.

## 1.3 Adaptively secure computation

We investigate adaptively secure multiparty computation (that is, computation secure in the presence of adversaries that choose which parties to corrupt as the computation proceeds, based on the information gathered so far). Unlike the case of non-adaptive adversaries, which is pretty well understood, the case of adaptive adversaries contains various aspects that were previously overlooked. In particular, unlike folklore belief, proving adaptive security of protocols in both the secure channels and computational settings encounters fundamental difficulties. We investigate these difficulties and provide solutions for some important special cases.

The difference between adaptive and non-adaptive adversaries may be best demonstrated via an example. Consider the following secret sharing protocol, run in the presence of an

[2] We borrow the name from an earlier version of [GMW], where it is used for different purposes.

adversary that may corrupt $t = O(n)$ out of the $n$ parties: *A dealer $D$ chooses at random a small set $S$ of $m = \sqrt{t}$ parties, and shares its secret among these parties using an $m$-out-of-$m$ sharing scheme. In addition $D$ publicizes the set $S$.* Intuitively, this scheme lacks in security since $S$ is public and $|S| \ll t$. Indeed, an adaptive adversary can easily find $D$'s secret, *without corrupting $D$,* by corrupting the parties in $S$. However, any non-adaptive adversary that does not corrupt $D$ learns $D$'s secret only if $S$ happens to be identical to the pre-defined set of corrupted parties. This happens only with exponentially small probability. Consequently, this protocol is secure in the presence of non-adaptive adversaries.

It turns out that the power of an adaptive adversary depends, in a crucial way, on the amount in which *uncorrupted* parties internally deviate from their protocols. Consider a party just corrupted by the adversary, during the course of the computation. If the party is totally honest, then the adversary will see exactly the data specified in the protocol; in particular, any data that was supposed to be erased will be indeed erased. In this case adaptively secure computation can be carried out, using known primitives, in all the settings discussed below [BH]. If, however, the party did not erase old data (or more generally if the party is semi-honest, as informally defined in Section 1.2), then the adversary may see a great deal of other data, such as all the past random choices of the party and all the messages ever received and sent by the party. (The adversary may also see other, more problematic types of internal data. We elaborate on this point in the sequel.) Therefore, the adversary is much more powerful in the presence of semi-honest parties. The more allowed "internal deviation" from the protocol, the stronger the adversary becomes.

We first consider the secure channels setting. Here the [BGW, CCD] protocols can be proven adaptively secure in the presence of non-erasing parties (see Section 1.2.1). Fundamental problems arise when trying to prove adaptive security of protocols in the presence of more general types of semi-honest parties. We sketch these problems.

Finally we concentrate on the computational setting, and on non-erasing parties. Is adaptively secure computation possible in this scenario? This question has remained open since the result of [GMW].

We answer this question in the affirmative. The problems encountered, and our solution, are presented via the following transformation. It is a folklore belief that any secure protocol in the secure channels setting can be transformed into a secure protocol in the computational setting, by encrypting each message using a standard (semantically) secure encryption scheme. This belief can indeed be turned into a proof, provided that only *non-adaptive* adversaries are considered. Major difficulties are encountered when trying to prove this belief in the presence of adaptive adversaries. We show how these difficulties are overcome if a novel protocol for transmission of encrypted data is used, instead of standard encryption. We call such encryption protocols non-committing. (Standard encryption schemes are not non-committing.) We also construct a non-committing encryption protocol, based on the existence of a primitive called common domain trapdoor systems. This primitive exists under the RSA assumption.

Non-committing encryption can be roughly described as follows. Traditional encryption schemes have the extra property that the ciphertext may serve as a commitment of the sender to the encrypted data. That is, suppose that after seeing the ciphertext, a third party requests the sender to *reveal* the encrypted data, and show how it was encrypted and decrypted. Using traditional encryption schemes it may be infeasible (or even impossible) for the sender to demonstrate that the encrypted data was any different than what was indeed transmitted. (In fact, many times encryption is explicitly or implicitly used for commitment.) In a **non-committing** encryption scheme the ciphertext cannot be used to commit the sender (or the receiver) to the transmitted data. That is, a non-committing encryption protocol allows a simulator to generate **dummy ciphertexts** that look like genuine ones, and can be later "opened" as encryptions of either 1 or 0, at wish. We note that communication over absolutely secure channels is trivially non-committing, since the third party sees no "ciphertext".

Our construction of non-committing data transmission requires all parties to participate in the secure transmission of information between two parties. For benefit of other possible applications, we note that our construction can be carried out in two stages; the first stage, which requires the participation of all parties, does not depend on the data to be delivered (which may even be undetermined at this stage), whereas the second stage consists of a single message transmission from the data-sender to the receiver. Our scheme is resilient as long as at least *one* party remains uncorrupted.

## 1.4 Asynchronous secure computation

We initiate a study of security in asynchronous networks. We consider a completely asynchronous network of $n$ parties connected by private channels. There is no global clock, and messages can be arbitrarily delayed on the channels (however, each message sent is eventually received). Furthermore, the order of the messages on a channel need not be preserved.

If the adversary is eavesdropping, then any *synchronous* secure protocol can be run in an *asynchronous* network using any synchronizer (e.g. [Aw]). It can be seen that in this case, the security (in the *synchronous* sense, as defined in Chapter 2) of the protocol is maintained. However, asynchrony combined with the possibility of faults has devastating consequences on the computational capabilities of a network. Fischer, Lynch and Paterson [FLP] showed that deterministic protocols cannot achieve even the basic goal of Consensus in an asynchronous network in the presence of even one Fail-Stop fault. Consequently, every (randomized) protocol reaching Consensus must have some infinite runs (on every input). Chor and Moscovici [CM] characterized the possible "tasks" in the presence of $t$ Fail-Stop faults: roughly speaking, the output of any computation, in the presence of $t$ potential faults, cannot be based on more than $n - t$ of the inputs (since up to $t$ parties may never join the computation).

We define secure computation in this asynchronous setting. Following the methodology of synchronous definitions (see Section 1.2), we first envision an ideal model for secure computation *in our asynchronous setting.* This ideal model is different than the 'synchronous' ideal model. We then say that a real-life computation is secure if it is "equivalent" to a computation in the ideal model. A computation in the ideal model proceeds as follows. Also here , an incorruptible **trusted party** is added to the network. Essentially, in the presence of $t$ potential faults (or corruptions), the trusted party cannot wait to hear from more than $n-t$ parties in the network (since up to $t$ may never join the computation). Instead, the trusted party outputs an "estimation" to the function value, based on the inputs of the parties in some "core" set of size at least $n - t$. This "core" set, chosen by the adversary, should be independent of the inputs of the uncorrupted parties. Furthermore, this "core" set should appear explicitly in the output of the uncorrupted parties (otherwise, the output may have no sense). The corrupted parties should learn nothing from the computation, other than the estimated function value, and the agreed "core" set.

We show that whatever can be computed in this asynchronous setting can be computed in a secure manner. We consider two types of adversaries. First, we show how to $t$-securely compute any function (in the asynchronous sense), provided that the adversary is Fail-Stop, and $n \geq 3t+1$. Next, we show how to $t$-securely compute any function in the presence of Byzantine adversaries, provided that $n \geq 4t+1$. In our protocols, there is no probability of error in the output of the uncorrupted parties. Although infinite runs of our protocols must exist, they occur with probability (or measure) zero.

The resilience of our construction to Fail-Stop adversaries is optimal. That is, we demonstrate functions which cannot be $n/3$-securely computed (or even approximated better than guessing at random) in the presence of Fail-Stop adversaries. The resilience of our construction for Byzantine adversaries is optimal with respect to errorless protocols. That is, we demonstrate functions that cannot be $\lceil \frac{n}{4} \rceil$-securely computed in the presence of Byzantine adversaries, if no errors are allowed. Subsequent to our work Ben-Or, Kelmer and Rabin showed, using very different constructions, how any function can be $(\lceil \frac{n}{3} \rceil - 1)$-securely computed in the presence of Byzantine adversaries, and with exponentially small probability of error [BKR].

Our constructions adapt the [BGW] synchronous constructions to an asynchronous environment. Furthermore, we develop several new tools which may be of separate interest. We describe a constant time, errorless Asynchronous Verifiable Secret Sharing (AVSS) scheme for $n \geq 4t+1$. (different constructions [Fe, CR] have a small probability of error.) Our AVSS scheme employs a method for 'on-line' error correcting of Generalized Reed Solomon codes, as well as a 'specially tailored' approximation scheme for the maximum-clique problem in a graph.

Another tool is a protocol for agreement on a common "core" set of parties. At the onset of this protocol, each party knows of a different set of parties that have completed some stage (e.g., the sharing of their inputs has been successfully completed). The protocol enables the uncorrupted parties to *agree* on a large enough "core" set of parties, such that all the parties in this core set have indeed completed the specified stage.

We present a full proof of security of our constructions. For clarity of presentation, we first prove that our protocols are secure against *non-adaptive* adversaries. Next we modify the proofs to deal with adaptive adversaries.

## 1.5 Asynchronous Byzantine Agreement

The problem of reaching agreement in the presence of faults is one of the most fundamental problems in the field of distributed computing. A particularly interesting variant of this problem, introduced by Pease, Shostak and Lamport [PSL], allows Byzantine adversaries. A standard formulation of this problem, called the Byzantine agreement (BA) problem, follows: design a protocol that allows the uncorrupted parties to agree on a common value. The agreed value should be the input value of one of the uncorrupted parties. We remark that Byzantine agreement is a very limited special case of secure multiparty computation, where privacy of the inputs of the parties need not be kept. Still, privacy-maintaining primitives will play a key role in our construction.

The BA problem was extensively investigated in various adversary models (out of the ones characterized at the beginning of the introduction). We refer the interested reader to the surveys of Fischer [F] and Chor and Dwork [CD]. However, despite extensive research a few important questions have remained open. One of these questions is the focus of this work.

Bounds on the resilience of BA protocols were proved in [PSL]. There, it was showed that agreement cannot be reached by a *deterministic* protocol in an $n$-party *synchronous* network with $\lceil \frac{n}{3} \rceil$ Byzantine faults. Karlin and Yao [KY] generalized this result to *randomized* protocols. These results apply also to *asynchronous* networks. Furthermore, the impossibility result of Fischer, Lynch and Paterson for *deterministic* protocols [FLP] implies that any (randomized) protocol reaching BA must have non-terminating runs. Bracha describes an $(\lceil \frac{n}{3} \rceil - 1)$-resilient asynchronous BA protocol which runs in $2^{\Theta(n)}$ expected time [Br]. Feldman and Micali describe a *synchronous* $(\lceil \frac{n}{3} \rceil - 1)$-resilient BA protocol, which runs in constant expected time [FM]. Feldman [Fe] generalizes the [FM] construction to an asynchronous setting, yielding a constant expected time, $(\lceil \frac{n}{4} \rceil - 1)$-resilient asynchronous BA protocol. All of these works allow computationally unbounded adversaries ([FM, Fe] assume secure channels).

A long standing open question (cf. [FM, CD]) is whether there exists an $(\lceil \frac{n}{3} \rceil - 1)$-resilient *a*synchronous BA protocol with polynomial (time and message) complexity.

We answer this question in the affirmative. We consider a completely asynchronous network of $n$ parties with secure channels, and computationally unlimited, adaptive adversaries. In this setting, we describe a BA protocol that is $(\lceil \frac{n}{3} \rceil - 1)$-resilient. With overwhelming probability all the uncorrupted parties complete our protocol. Given that all the uncorrupted parties have completed the protocol, they do so in constant expected time. The constructions we use in our protocol are of independent interest.

Let us overview the chain of results leading to our result, and sketch the techniques used. Rabin [MRa2] describes an $(\lceil \frac{n}{8} \rceil - 1)$-resilient BA protocol that runs in constant expected time, provided that all the parties have access to a 'global coin' (namely, a *common* source of randomness). Rabin's construction can be used in synchronous as well as asynchronous networks. Bracha [Br] improved the resilience of Rabin's protocol to $\lceil \frac{n}{3} \rceil - 1$. Furthermore, He proposed a very simple (however inefficient) scheme for implementing 'global coin'. (This inefficiency results in exponential running time.) The essence of the [FM] $(\lceil \frac{n}{3} \rceil - 1)$-resilient synchronous BA protocol is an *efficient* scheme for generating such a 'global coin'; once this global coin is generated, the parties proceed in a similar manner to Rabin's and Bracha's protocols. The [FM] protocol for generating this 'global coin' relies heavily on a Verifiable Secret Sharing (VSS) scheme. (The notion of VSS was introduced in [CGMA].) Feldman [Fe] describes an *asynchronous* construction for 'global coin' and BA, given an $r$-resilient *Asynchronous* VSS (AVSS) scheme. This construction is $min(r, \lceil \frac{n}{3} \rceil - 1)$-resilient. $(\lceil \frac{n}{4} \rceil - 1)$-resilient AVSS schemes are presented in [Fe, BCG]. (The [BCG] scheme is presented in Section 4.4.) Up to now, no known AVSS scheme has been more than $(\lceil \frac{n}{4} \rceil - 1)$-resilient.

In this chapter, we construct an $(\lceil \frac{n}{3} \rceil - 1)$-resilient AVSS scheme. We also present a considerably modified version of Feldman's construction for reaching BA given an AVSS scheme. Put together, these constructions constitute an asynchronous $(\lceil \frac{n}{3} \rceil - 1)$-resilient

BA protocol. We note that our $(\lceil \frac{n}{3} \rceil - 1)$-resilient AVSS scheme has applications in other contexts as well (for instance, in [BE, BKR]).

We offer an intuitive exposition of the difficulties encountered in trying to devise an $(\lceil \frac{n}{3} \rceil - 1)$-resilient AVSS scheme. Generally, in an asynchronous network of $n$ parties with $t$ potential faults, a party can never wait to communicate with more than $n - t$ other parties, because the corrupted parties may not cooperate. An AVSS scheme is composed of two phases: (1) a sharing phase, in which a dealer shares a secret among the parties, and each party verifies for himself that a unique secret is defined by the shares, and (2) a reconstruction phase in which the parties reconstruct the secret from its shares. A party $P$ must be able to complete the execution of the sharing phase even if he has communicated with only $n-t$ of the parties. This means that he has verified the existence of a well defined secret only with this subset of the parties, denoted $C_1$. When $P$ proceeds to carry out the reconstruction phase he again can communicate with at most $n - t$ of the parties. Denote this set $C_2$. The set $C_2$ might include parties with whom $P$ has not communicated in the sharing phase, hence $P$ does not know whether their shares are in accordance with the secret defined by the shares of the parties in $C_1$. (Possibly, the parties that are not in $C_1$ did not receive shares at all.) Thus, $P$ can depend only on the parties in the intersection, $C$, of the two sets. Still, $t$ out of the parties in $C$ may be corrupted. As long as $n \geq 4t + 1$, it holds that $|C| \geq 2t + 1$; thus, the majority of the parties in $C$ are uncorrupted. However, when $n = 3t + 1$, it is possible that $|C| = t + 1$. In this case, there might be only a single uncorrupted party in $C$.

We overcome these difficulties by devising a tool, called Asynchronous Recoverable Sharing (A-RS), assuring that, with overwhelming probability, the shares of *all* the parties in the set $C_1$ (defined above) will be available in the reconstruction phase. The A-RS protocol uses a tool presented in [TRa, RB], called Information Checking Protocol. Using A-RS as a primitive, we construct a secret sharing scheme, called Asynchronous Weak Secret Sharing (AWSS). Using AWSS, we construct our AVSS scheme. (Both the AWSS and the AVSS schemes generalize synchronous constructs introduced in [TRa, RB].)

## 1.6 Proactive security: Maintaining security in the presence of transient faults

Traditionally, cryptography is focused on protecting interacting parties (i.e., computers) against *external* malicious entities. Such cryptographic tasks include private communication over insecure channels, authentication of parties, unforgeable signatures, and general multiparty secure computation. An inherent property of all these scenarios is that once a party is corrupted it remains this way.

As computer systems become more complex, *internal* attacks on systems (i.e., attacks that corrupt components within a system) become an even more important security threat (e.g., [LE, St]). Such attacks may be performed, for instance, by internal (human) fraud, operating system weaknesses, or Trojan horse software (e.g. viruses). We use the generic term **break-ins** for all these attacks. Security administrators often find break-ins more alarming than external attacks, such as line tappings.

Break-ins are often temporary, or **transient** (e.g., [ER]). Thus the paradigm of "bad once means bad forever" does not hold here. Still, known solutions to break-ins do not include mechanisms for taking advantage of possible *automatic recovery* of a component, in case that the fault is transient. This approach is contrasted with the traditional approach of fault-tolerance, which relies heavily on the fact that faults are transient, and on the reuse of recovered components. We believe that the idea of recovering and reusing components that have once been corrupted can be extremely useful also for cryptographic purposes. This idea, and in particular the recovery process, is the focus of this work.

We propose a new approach to designing security systems in the presence of perpetual, however transient break-ins. This approach, which we call the proactive approach, may be outlined as follows.

> (a). Distribute the tasks and responsibilities among several components ofthe system. Design the system so that the overall security remains intact as long as at any instance the security of only some fraction of the components is compromised.
> (b). Design a mechanism for automatic recovery of a given component from a break-in, possibly with the help of other components. Recovery will be guaranteed only if the component is no longer corrupted (i.e., controlled by an adversary).
> (c). Apply the automatic recovery mechanism periodically to all components in the system.

This way, the overall security provided by the system remains intact in the presence of break-ins, as long as no large fraction of the components are broken into *all at once* (that is, between two consecutive applications of the recovery mechanism).

The automatic recovery process is at the heart of proactive security. The goal of the recovery process is to ensure that once a component is recovered, it will again contribute to the overall security of the system. This goal is somewhat tricky. Even after the attacker loses control of a component, it still knows the internal data of the component (e.g., the private cryptographic keys). Thus, a first step in the recovery process must be to somehow hand the recovering component some new secrets unknown to the attacker. These secrets can then be used to, say, choose new keys. The obvious way to generate such secrets is to use some source of "fresh", physical randomness. However, such a source may not be readily available. (In the sequel we demonstrate other reasons for not using fresh randomness in each round, even when it is available.) In this paper we show how, using the non-corrupted components in the system, new "pseudorandom" secrets can be generated *without* fresh randomness. In Section 1.6.1 we describe an important application of our PP protocol to secure sign-on mechanisms.

We use multiparty secure computation as a formal setting for our work, as follows. We consider a system (network) of components (parties) where every two parties are connected via a communication channel. (We elaborate below on the security requirements from the channels.) Parties may be temporarily corrupted by a mobile adversary. That is, the adversary may choose to corrupt different parties at different times (i.e., communication rounds), as long as at any given time the number of infected parties is limited. We stress that there may be no party that has never been infected! Secure multiparty computation in the presence of mobile adversaries was previously studied by Ostrovsky and Yung [OY].

We remark that here the notion of semi-honest parties (discussed in Section 1.2) is irrelevant, since all components are programmed and run by the same entity. In fact, erasing old data plays a key role in our constructions.

We assume that, even if the faults are Byzantine, once the adversary has left the party resumes executing its original protocol (while its memory may be corrupted). This assumption is explained as follows. If the adversary can control the protocol after it leaves, then there is little meaning to recovery, and regaining security would be impossible. Furthermore, in practice there are reasonable ways to ensure that the code is not modified, such as physical read-only storage or comparison against backup copies. These techniques are used regularly in many systems.

In this work, we describe a scheme in which the parties use randomness *only at the beginning of the computation.* At each round, the scheme supplies each uncorrupted party with a "fresh" pseudorandom number, unpredictable by the adversary, even if this party was corrupted in previous rounds, and if the adversary knows all the other pseudorandom numbers supplied to any party at any round. In particular, these pseudorandom numbers can be used by a recovering party just as fresh random numbers (e.g., for regaining security). We call such a scheme a proactive pseudorandomness (PP) protocol.

We require the following weak conditions. First, we assume that the adversary is limited to probabilistic polynomial time. Next, we require that in each round of computation there is at least *one* secure party. A party is secure at a given round if it is uncorrupted at this round, and it has a secure channel to a party that was secure in the previous round. This channel has to be secure only during this round.

Our construction is simple, using pseudorandom functions [GGM2, GGM1]. A standard cryptographic tool which is believed to behave as a pseudorandom function family is the Data Encryption Standard (DES). Our construction requires each server to apply DES once per user at each round.

### 1.6.1 Reconstructability and an application to secure sign-on

We describe an important application of our PP protocol to secure sign-on mechanisms.

**Reconstructability.** Pseudorandom generators, being deterministic functions applied to a random seed, have the following advantage over truly random sources. A pseudorandom sequence is reconstructible, in the sense that it is possible to generate exactly the same sequence again by using the same seed. This property is very useful for several purposes, such as repeatable simulations and debugging. Our application to secure sign-on also makes use of this property.

In our setting, we say that a PP protocol is reconstructible if the value generated within each party at each round depends only on the seeds chosen by the parties at the beginning of the computation. In particular, these values should not depend on the adversary.

Reconstructability is not easily achieved for proactive pseudorandomness protocols. In particular, the basic protocol described here is reconstructible only if the adversary is eavesdropping. Fail-Stop adversaries (and also Byzantine adversaries, at the price of slightly compromising the security) could be tolerated by simple modifications.

**An application to Secure Sign-On.** Unix and other operating systems provide security for the passwords by storing only a one-way function of the passwords on disk [MT]. This technique allows authentication of the users, secure against eavesdropping the password file. Session security is not provided if the communication channels are not secure.

When constructing secure LAN systems, it is not realistic to assume that the underlying communication channels are secure. Security mechanisms, therefore, avoid sending the password "on the clear". Instead, they use the user's password to derive a *session key*, with which they secure the communication. In both Kerberos [MNSS] and NetSP / KryptoKnight[BGH$^+$1], this is done by using the password as a key for exchanging a random session key; this method also allows NetSP / KryptoKnight to authenticate the user automatically to additional systems ('single sign on').

However, this mechanism implies that some server must be able to compute the session key itself, using some secret (e.g. the password). This in turn implies that the server has to maintain the password file *secret*. This secrecy requirement is a major 'Achilles heel' of any security system. (Indeed, NetWare 4.0 provides a more complicated and computationally intensive solution, where the server keeps, for each user, an RSA private key encrypted using the user's password. The encrypted private key is sent to the workstation, which decrypts it using the password, and then uses it to derive a session key. This solution only requires the password file remains unmodified, rather than secret.)

We show how a reconstructible proactive pseudorandomness protocol can be used to overcome this weakness, without compromising efficiency. Our solution uses several proactive sign-on servers. The servers run a different copy of our PP protocol for each user. The initial seed of each server $P_i$ is a pseudorandom value derived from the user's password in a straightforward way. Each server sets its key for each time period to be the current output of the PP protocol. The user, knowing all the servers' inputs of this reconstructible computation, can simulate the computation and compute each server's key at any time period *without need for any communication.* Thus, a user can always interact with the server of his choice. The security of our PP protocol makes sure that a mobile adversary does not know the key currently used by a secure server, as long as in each round there exists at least one secure server.

Our solution does not require public key mechanisms. Furthermore, it is valid even if the attacker can modify the login files kept by the servers.

CHAPTER 2

# Defining secure multiparty computation

We present definitions of secure multiparty computation in different adversary models. Good definitions of secure multiparty computation are notoriously hard to formulate, and may become very complex in some settings. (See Section 1.2 for an introductory discussion.) We therefore start with a simple setting: non-adaptive, computationally unbounded adversaries with secure channels, in a synchronous network. (We later distinguish two very different variants of this setting.) Although we do not use this definition in the sequel, its presentation captures many of the ideas needed for defining multiparty secure computation.

Next we concentrate on adaptive adversaries. Here the notion of semi-honest parties, introduced in Section 1.2, is central to our definitions. (The notion of semi-honest parties is irrelevant in the presence of non-adaptive adversaries.) We first define several variants of semi-honest parties, differing in the amount of internal deviation from the protocol. Next we present our definition of adaptively secure computation. We also consider the **computational setting**, where the channels are insecure and the adversary is restricted to probabilistic polynomial time (PPT). Both settings are synchronous. Definitions of secure multiparty computation in other settings can be formulated, using the same methodology. In particular, in Chapter 4 we define secure multiparty computation in an asynchronous setting.

Let us recall the standard definition of computational indistinguishability of distributions.

**Definition 2.1** *Let $\mathcal{A} = \{A_n\}_{n\in\mathbf{N}}$ and $\mathcal{B} = \{B_n\}_{n\in\mathbf{N}}$ be two ensembles of probability distributions. We say that $\mathcal{A}$ and $\mathcal{B}$ are* **computationally indistinguishable** *if for every constant $c > 0$, for every polytime distinguisher $D$ and for all large enough $n$,*

$$|\mathrm{Prob}(D(A_n) = 1) - \mathrm{Prob}(D(B_n) = 1)| < \frac{1}{n^c}.$$

We colloquially say that "$A_n$ and $B_n$ are computationally indistinguishable", or "$A_n \stackrel{c}{\approx} B_n$".

## 2.1 Non-adaptively secure computation

We define non-adaptively secure multiparty computation in the secure channels setting. That is, we consider a synchronous network where every two parties are connected via an absolutely secure communication channel (i.e., the adversary cannot hear, nor alter, messages sent between uncorrupted parties).[1] The adversary is computationally unlimited. We use the standard methodology presented in Section 1.2. Recall that executing a protocol for computing some function is compared to evaluating the function in an ideal model, where a trusted party is used. We substantiate the definition in three steps. First, we give an exact definition of this ideal model. Next, we formulate our (high level) notion of 'real-life' protocol execution. Finally, we describe and formalize our method of comparing computations.

Let $f : D^n \rightarrow D'$ be a function, for some domains $D$ and $D'$. The parties have inputs $\vec{x} = x_1 \ldots x_n \in D^n$ (party $P_i$ has input $x_i$) and wish to compute $f(x_1, \ldots, x_n)$.[2] The ideal-model-adversary $\mathcal{S}$ has a fixed set, $B$, of up to $t$ corrupted parties, and has the inputs of the parties in $B$. The computation in the ideal model proceeds as follows.

**Input substitution stage:** The ideal-model-adversary $\mathcal{S}$ may alter the inputs of the corrupted parties; however, this is done without any knowledge of the inputs of the good parties. Let $\vec{b}$ be the $|B|$-vector of the altered inputs of the corrupted parties, and let $\vec{y}$ be the $n$-vector constructed from the input $\vec{x}$ by substituting the entries of the corrupted parties by the corresponding entries in $\vec{b}$.

**Computation stage:** The parties hand $\vec{y}$ to the trusted party (party $P_i$ hands $y_i$), and receive $f(\vec{y})$ from the trusted party.[3]

**Output stage:** The uncorrupted parties output $f(\vec{y})$, and the corrupted parties output some arbitrary function, computed by the adversary, of the information gathered during the computation in the ideal model. This information consists only of their inputs, their joint random input (and, consequently, the altered input vector $\vec{b}$), and the resulting function value $f(\vec{y})$. We let the $n$-vector $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}) = \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_1 \ldots \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_n$ denote the outputs of the parties on input $\vec{x}$ and adversary $\mathcal{S}$ (party $P_i$ outputs $\text{IDEAL}_{f,\mathcal{S}}(\vec{x})_i$).

In Definitions 2.2 and 2.3 we formally define the output of the parties in the ideal

[1] An immediate interpretation of this model is that of a closed system: there is a fixed number of parties, all of which know each others identity. This is a conceptually simple, however somewhat limiting interpretation.
A more general and realistic interpretation is that of an open system: there is an unbounded number of parties in the network, each with a unique identity. The parties need not be aware of each other *a-priori*. Still, only a limited number of parties actually join the computation. (In most cases, a limit on the number of parties that may join needs to be known in advance.)
The distinction between these two interpretations of the model is merely conceptual, and is not reflected in the definitions in any way. In particular, in both cases the number of parties is taken to be the number of parties that actually join the computation.

[2] A more general formulation allows different parties to compute a different functions of the input. Specifically, in this case the range of $f$ is a $n$-fold Cartesian product and the interpretation is that the $i^{\text{th}}$ party should get the $i^{\text{th}}$ component of $f(\vec{x})$.

[3] In the case where each party computes a different function of the inputs, as discussed in the previous footnote, the trusted party will hand each party its specified output.

model. (These definitions capture the above description, and can be skipped in a first reading.) First, we need two technical notations.

- For a vector $\vec{x} = x_1 \ldots x_n$ and a set $B \subseteq [n]$, let $\vec{x}_B$ denote the vector $\vec{x}$, projected on the indices in $B$.

- For an $n$-vector $\vec{x} = x_1 \ldots x_n$, a set $B \subseteq [n]$, and a $|B|$-vector $\vec{b} = b_1 \ldots b_{|B|}$, let $\vec{x}/_{(B,\vec{b})}$ denote the vector constructed from vector $\vec{x}$ by substituting the entries in $B$ by the corresponding entries from $\vec{b}$.

**Definition 2.2** *Let $S$ be the domain of possible inputs of the parties, and let $\mathcal{R}$ be the domain of possible random inputs. A $t$-*limited ideal-model-adversary *is a triplet $\mathcal{S} = (B, h, O)$, where:*

- *$B$ is the set of corrupted parties.*

- *$h : [n]^* \times S^* \times \mathcal{R} \to S^*$ is the* input substitution *function*

- *$O : S^* \times \mathcal{R} \to \{0,1\}^*$ is an* output *function for the bad parties.*

**Definition 2.3** *Let $f : D^n \to D'$ for some domains $D$ and $D'$ be the computed function, and let $\vec{x} \in D^n$ be an input vector. The* output of computing function $f$ in the ideal model *with adversary $\mathcal{S} = (B, h, O)$, on input $\vec{x}$ and random input $r$, is an $n$-vector $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}) = \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_1 \ldots \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_n$ of random variables, satisfying for every $1 \le i \le n$:*

$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x})_i = \begin{cases} f(\vec{y}) & \text{if} \quad i \notin B \\ O(\vec{x}_B, f(\vec{y}), r) & \text{if} \quad i \in B \end{cases}$$

*where $r$ is the random input of $\mathcal{S}$, and $\vec{y} = \vec{x}/_{(B,h(B,\vec{x}_B,r))}$ is the substituted input vector for the trusted party.*

Next we describe the execution of a protocol $\pi$ in the real-life scenario. The parties engage in a synchronous computation in the secure channels setting, running protocol $\pi$. A computationally unbounded (non-adaptive) $t$-limited real-life adversary controls a fixed set $B$ of corrupted parties. Once the computation is completed, each uncorrupted party outputs whatever it has computed to be the function value. Without loss of generality, we assume that the corrupted parties output their entire view on the computation. The view consists of all the information gathered by the adversary during the computation. Specifically, the view includes the inputs and random inputs of the corrupted parties, and the communication seen by the corrupted parties.

We use the following notation. Let $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{r})$ denote the view of the adversary $\mathcal{A}$ when interacting with parties running protocol $\pi$ on input $\vec{x}$ and random input $\vec{r}$ ($x_i$ and $r_i$ for party $P_i$), as described above. Let $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x})$ denote the random variable describing the distribution of $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{r})$ when $\vec{r}$ is randomly chosen. Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{r})_i$ denote the output of party $P_i$ after running protocol $\pi$ on input $\vec{x} = x_1 \ldots x_n$ and random input $\vec{r} = r_1 \ldots r_n$, and with a real life adversary $\mathcal{A}$. Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x})_i$ denote the random variable describing $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{r})_i$ where $\vec{r}$ is uniformly chosen. Let $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}) = \text{EXEC}_{\pi,\mathcal{A}}(\vec{x})_1 \ldots \text{EXEC}_{\pi,\mathcal{A}}(\vec{x})_n$. (We have $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x})_i = \text{VIEW}_{\pi,\mathcal{A}}(\vec{x}, \vec{r})$ for corrupted parties $P_i$.)

Finally we require that executing a secure protocol $\pi$ for evaluating a function $f$ be equivalent to evaluating $f$ in the ideal model, in the following sense. For any real-life adversary $\mathcal{A}$ there should exist an ideal-model adversary $\mathcal{S}$, such that for every input vector $\vec{x}$, the output vectors $\text{IDEAL}_{f,\mathcal{S}}(\vec{x})$ and $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x})$ are identically distributed.

We require that the complexity of the ideal-model adversary $\mathcal{S}$ be comparable to the complexity of the real-life adversary $\mathcal{A}$. This requirement can be motivated as follows. The ideal-model adversary is an imaginary concept whose purpose is to formalize the following statement: "whatever the adversary learns from interacting with parties running $\pi$, he could have also learned in the ideal model *in roughly the same computational effort.*" If we do not limit the computational power of the ideal-model adversary, we end up with a much weaker notion of security, detached from realistic computations. We remark that, in the presence of adaptive adversaries, bounding the computational power of the ideal-model adversary introduces several previously overlooked problems. We elaborate on these problems in Chapter 3.

In the sequel, whenever we refer to the secure channels setting, we assume that the complexity of the ideal-model adversary is polynomial in the complexity of the real-life adversary. We let the unbounded secure channels setting denote the case where the ideal-model adversary is computationally unbounded.

**Definition 2.4** *Let $f : D^n \rightarrow D$ for some domains $D$ and $D$, and let $\pi$ be a protocol for $n$ parties. We say that $\pi$* (non-adaptively) $t$-securely computes $f$ in unbounded the secure channels setting, *if for any (non-adaptive) $t$-limited real-life adversary $\mathcal{A}$, there exists a (non-adaptive) $t$-limited ideal-model-adversary $\mathcal{S}$, whose running time is polynomial in the running time of $\mathcal{A}$, such that for every input vector $\vec{x}$,*

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

*If the running time of $\mathcal{S}$ is polynomial in the running time of $\mathcal{A}$, then we say that $\pi$* (non-adaptively) $t$-securely computes $f$ in the bounded secure channels setting.

**Remarks:**

- For measuring complexity, we assume that the protocol $\pi$, the simulator $\mathcal{S}$ and the adversary $\mathcal{A}$ are Turing machines that have $n$, the number of parties, as part of their input. We measure the complexity of $\pi$, $\mathcal{S}$ and $\mathcal{A}$ with respect to $n$. (The function $f$ is now a family of functions, where a function corresponds to each value of $n$.)

- We stress that the two *whole* output $n$-vectors must be identically distributed. Some previous definitions of secure computation (e.g., [GMW]) partitioned the Security requirement into two separate requirements: (a) the output of the corrupted parties be equal in both scenarios, and (b) the output of the uncorrupted parties be equal in both scenarios. However, as pointed out in [MR], requiring (a) and (b) does not guarantee secure computation.

## 2.2 Semi-honest parties

We define semi-honest parties (or, equivalently, semi-honest protocols). We consider three alternative notions of semi-honesty. First we take a minimalist approach, in which semi-honest parties deviate from the protocol only by not erasing old data. We call such parties honest-but-non-erasing, or in short non-erasing. More precisely, say that a memory tape is write-once if it consists of memory locations that can be modified only once (from their initial default contents). Non erasing protocols are defined as follows.

**Definition 2.5** *Let $\pi$ and $\pi'$ be $n$-party protocols. We say that $\pi'$ is a* non-erasing *protocol for $\pi$ if $\pi'$ is identical to $\pi$ with the exception that, in addition to the instructions of $\pi$, protocol $\pi'$ copies the contents of each memory location accessed by $\pi$ to a special write once memory tape.*

Alternatively, one may consider semi-honest parties that execute some arbitrary protocol other than the specified one, with the only restriction that no external test (representing the combination of all other parties) can distinguish between such a behaviour and a truly honest behaviour. We call such parties honest-looking. We consider two variants of honest-looking parties: In the secure channels setting the external distinguishing test is computationally unbounded. In the computational setting the external distinguishing test is computationally bounded. More formally, let $\text{COM}_\pi(\vec{x}, \vec{r})$ denote the communication among $n$ parties running $\pi$ on input $\vec{x}$ and random input $\vec{r}$ ($x_i$ and $r_i$ for party $P_i$). Let $\text{COM}_\pi(\vec{x})$ denote the random variable describing $\text{COM}_\pi(\vec{x}, \vec{r})$ when $\vec{r}$ is uniformly chosen. For $n$-party protocols $\rho$ and $\pi$ and an index $i \in [n]$, let $\rho_{/(i,\pi)}$ denote the protocol where party $P_i$ executes $\pi$ and all the other parties execute $\rho$.

**Definition 2.6** *Let $\pi$ and $\pi'$ be $n$-party protocols. We say that $\pi'$ is a* perfectly honest-looking *protocol for $\pi$ if for any input $\vec{x}$, for any $n$-party "test" protocol $\rho$, and for any index $i \in [n]$,*

$$\text{COM}_{\rho_{/(i,\pi)}}(\vec{x}) \stackrel{\text{d}}{=} \text{COM}_{\rho_{/(i,\pi')}}(\vec{x}).$$

*If the test protocol $\rho$ is restricted to probabilistic polynomial time, and $\text{COM}_{\rho_{/(i,\pi)}}(\vec{x}) \stackrel{\text{c}}{\approx} \text{COM}_{\rho_{/(i,\pi')}}(\vec{x})$, then we say that $\pi'$ is a* computationally honest-looking *protocol for $\pi$.*

We stress that here the "test" protocol $\rho$ represents a collaboration of all parties for testing whether $P_i$ is honest.

**Remark:** Note that both the perfect and the computational variants of honest-looking parties can do other "harmful" things, on top of not erasing data. For instance, assume that some one-way permutation $f$, defined on some domain $D$, is known to all parties. When instructed to choose a value $x$ at random from $D$, an honest-looking party can instead choose $y$ at random from $D$ and let $x = f(y)$. Thus, the party cannot be trusted to *not* know $f^{-1}(x)$. Also, let $f_0, f_1$ be a claw-free pair of permutations over $D$. Then, when instructed to choose a random input $r \in D$ for use in its protocol, the party can, on input $\sigma \in \{0, 1\}$, use $f_\sigma(r)$ for random input instead of $r$. (This particular example is very 'disturbing', as will become clear in the Chapter 3.)

An even more permissive approach allows a semi-honest party to deviate arbitrarily from the protocol, as long as his behaviour appears honest to parties *executing the protocol.* We stress that other external tests, not specified in the protocol, may be able to detect such a party as cheating. We call such semi-honest parties weakly-honest. More precisely, here we require that Definition 2.6 is satisfied only with respect to the original protocol $\pi$, rather than with respect to any test protocol $\rho$.

**Definition 2.7** *Let $\pi$ and $\pi'$ be n-party protocols. We say that $\pi'$ is an* perfectly weakly-honest *protocol for $\pi$ if for any input $\vec{x}$ and for any index $i \in [n]$,*

$$\text{COM}_{\pi}(\vec{x}) \stackrel{\text{d}}{=} \text{COM}_{\pi/(i,\pi')}(\vec{x}).$$

*If $\pi$ is restricted to probabilistic polynomial time, and if $\text{COM}_{\pi}(\vec{x}) \stackrel{\text{c}}{\approx} \text{COM}_{\pi/(i,\pi')}(\vec{x})$, then we say that $\pi'$ is a* computationally weakly-honest *protocol for $\pi$.*

The choice of these particular notions of semi-honest parties can be motivated as follows. Non-erasing behaviour is a very simple deviation from the protocol, that is very hard to prevent. Even if the protocol (say, given to parties as a piece of software) is protected against modifications, it is always possible to add an *external* device that copies all memory locations accessed by the protocol to a "safe" memory where the data is kept. Such an external device requires no understanding in the internal structure or in the behaviour of the protocol. Honest-looking parties represent "sophisticated" parties that internally deviate from the protocol in an arbitrary way, but are willing to take no chance that they will *ever* be uncovered (say, by an unexpected audit). Weakly-honest parties represent the most general internal deviation from the protocol that may remain undetected by other parties running the protocol.

## 2.3 Adaptively secure computation in the secure channels setting

We define adaptively secure multiparty computation in the the secure channels setting. We use the same "ideal model methodology" as in Section 2.1, with respect to adaptive adversaries. Here, however, we require that the computation be secure even if the parties run any semi-honest protocol for a given protocol $\pi$.

We start by re-defining how a function is evaluated in the ideal model. Let $f : D^n \to D$ be a function, for some domains $D$ and $D'$. The parties have inputs $\vec{x} = x_1 \ldots x_n \in D^n$ (party $P_i$ has input $x_i$) and wish to compute $f(x_1, \ldots, x_n)$. The ideal-model-adversary $\mathcal{S}$ has no initial input, and is parameterized by $t$, the maximum number of parties it may corrupt. In the sequel we limit the computational power of the adversary. The difference from the non-adaptive ideal-model adversary (Section 2.1) are the two extra adaptive corruption stages.

**First corruption stage:** First, $\mathcal{S}$ proceeds in up to $t$ iterations. In each iteration $\mathcal{S}$ may decide to corrupt some party, based on $\mathcal{S}$'s random input and the information gathered so far. Once a party is corrupted its internal data (that is, its input and random input) become known to $\mathcal{S}$. A corrupted party remains corrupted for the rest of the computation. Let $B$ denote the set of corrupted parties at the end of this stage.

**Input substitution stage:** $\mathcal{S}$ may alter the inputs of the corrupted parties; however, this is done without any knowledge of the inputs of the good parties. Let $\vec{b}$ be the $|B|$-vector of the altered inputs of the corrupted parties, and let $\vec{y}$ be the $n$-vector constructed from the input $\vec{x}$ by substituting the entries of the corrupted parties by the corresponding entries in $\vec{b}$.

**Computation stage:** The parties hand $\vec{y}$ to the trusted party (party $P_i$ hands $y_i$), and receive $f(\vec{y})$ from the trusted party.

**Second corruption stage:** Now that the output of the computation is known, $\mathcal{S}$ proceeds in another sequence of up to $t - |B|$ iterations, where in each iteration $\mathcal{S}$ may decide to corrupt some additional party, based on $\mathcal{S}$'s random input and the information gathered so far (this information now includes the value received from the trusted party). We stress that $\mathcal{S}$ may corrupt at most $t$ parties in the entire computation.

**Output stage:** The uncorrupted parties output $f(\vec{y})$, and the corrupted parties output some arbitrary function, computed by the adversary, of the information gathered during the computation in the ideal model. This information consists only of their inputs, their joint random input (and, consequently, the altered input vector $\vec{b}$), and the resulting function value $f(\vec{y})$. We let the $n$-vector $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}) = \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_1 \ldots \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_n$ denote the outputs of the parties on input $\vec{x}$ and adversary $\mathcal{S}$ (party $P_i$ outputs $\text{IDEAL}_{f,\mathcal{S}}(\vec{x})_i$).

In Definitions 2.8 through 2.10 we formally define the output of the parties in the ideal model. These definitions capture the above description, and can be skipped in a first reading. We use the notations $\vec{x}_B$ and $\vec{x}/_{(B,\vec{b})}$ as in Section 2.1.

**Definition 2.8** *Let $D$ be the domain of possible inputs of the parties, and let $\mathcal{R}$ be the domain of possible random inputs. A $t$-*limited ideal-model-adversary *is a quadruple $\mathcal{S} = (t, b, h, O)$, where:*

- *$t$ is the maximum number of corrupted parties.*
- *$b : [n]^* \times D^* \times \mathcal{R} \to [n] \cup \{\perp\}$ is the* selection *function for corrupting parties (the value $\perp$ is interpreted as "no more parties to corrupt at this stage")*
- *$h : [n]^* \times D^* \times \mathcal{R} \to D^*$ is the* input substitution *function*
- *$O : D^* \times \mathcal{R} \to \{0, 1\}^*$ is an* output *function for the bad parties.*

The sets of corrupted parties are now defined as follows.

**Definition 2.9** *Let $D$ be the domain of possible inputs of the parties, and let $\mathcal{S} = (t, b, h, O)$ be a $t$-limited ideal-model-adversary. Let $\vec{x} \in D^n$ be an input vector, and let $r \in \mathcal{R}$ be a random input for $\mathcal{S}$. The $i$*th set of corrupted parties *in the ideal model $B^{(i)}(\vec{x}, r)$, is defined as follows.*

- $B^{(0)}(\vec{x}, r) = \phi$

- *Let* $b_i \triangleq b(B^{(i)}(\vec{x}, r), \vec{x}_{B^{(i)}(\vec{x},r)}, r)$*. For* $0 \le i < t$*, and as long as* $b_i \ne \perp$*, let*

$$B^{(i+1)}(\vec{x}, r) \triangleq B^{(i)}(\vec{x}, r) \cup \{b_i\}$$

- *Let* $i^*$ *be the minimum between* $t$ *and the first* $i$ *such that* $b_i = \perp$*. Let* $b_i^f \triangleq b(B^{(i)}(\vec{x}, r), \vec{x}_{B^{(i)}(\vec{x},r)}, f(\vec{y}), r)$*, where* $\vec{y}$ *is the substituted input vector for the trusted party.*
  *That is,* $\vec{y} \triangleq \vec{x}/_{(B^{(i^*)}(\vec{x},r), h(B^{(i^*)}(\vec{x},r), \vec{x}_{B^{(i^*)}(\vec{x},r)}, r))}$.
  *For* $i^* \le i < t$*, let*

$$B^{(i+1)}(\vec{x}, r) \triangleq B^{(i)}(\vec{x}, r) \cup b_i^f.$$

In Definition 2.10 we use $B^{(i)}$ instead of $B^{(i)}(\vec{x}, r)$.

**Definition 2.10** *Let* $f : D^n \to D'$ *for some domains* $D$ *and* $D$ *be the computed function, and let* $\vec{x} \in D^n$ *be an input vector. The* output of computing function $f$ in the ideal model *with adversary* $\mathcal{S} = (t, b, h, O)$*, on input* $\vec{x}$ *and random input* $r$*, is an* $n$*-vector* $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}) = \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_1 \ldots \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_n$ *of random variables, satisfying for every* $1 \le i \le n$*:*

$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x})_i = \begin{cases} f(\vec{y}) & \text{if} \quad i \notin B^{(t)} \\ O(\vec{x}_{B^{(t)}}, f(\vec{y}), r) & \text{if} \quad i \in B^{(t)} \end{cases}$$

*where* $B^{(t)}$ *is the* $t^{\text{th}}$ *set of corrupted parties,* $r$ *is the random input of* $\mathcal{S}$*, and* $\vec{y} = \vec{x}/_{(B^{(t)}, h(B^{(t)}, \vec{x}_{B^{(t)}}, r))}$ *is the substituted input vector for the trusted party.*

Next we describe the execution of a protocol $\pi$ in an (adaptive) real-life scenario. The parties engage in a synchronous computation in the secure channels setting, running some semi-honest protocol $\pi'$ for $\pi$ (according to any one of the notions of semi-honesty defined above). A computationally unbounded (adaptive) $t$-limited real-life adversary may choose to corrupt parties at any point during the computation, based on the information known to the previously corrupted parties, and as long as at most $t$ parties are corrupted altogether. Once a party is corrupted the current contents of its memory (as determined by the semi-honest protocol $\pi'$) becomes available to the adversary. From this point on, the corrupted party follows the instructions of the adversary. Once the computation is completed, each uncorrupted party outputs whatever it has computed to be the function value. Also here, we assume that the corrupted parties output their entire view on the computation. The view consists of all the information gathered by the adversary during the computation. Specifically, the view includes the inputs and random inputs of the corrupted parties, the communication seen by the corrupted parties, and the internal data of parties upon corruption.

We let $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x})$ and $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}, \vec{r})$ have an analogous meaning to Section 2.1 in the presence of adaptive adversaries.

**Definition 2.11** *Let* $f : D^n \to D'$ *for some domains* $D$ *and* $D'$*, and let* $\pi$ *be a protocol for* $n$ *parties. We say that* $\pi$ $t$-securely computes $f$ in the unbounded secure channels setting*, if for any semi-honest protocol* $\pi'$ *for* $\pi$ *(according to any of the Definitions 2.5 through 2.7), and for any* $t$*-limited real-life adversary* $\mathcal{A}$*, there exists a* $t$*-limited ideal-model-adversary* $\mathcal{S}$*,* *whose running time is polynomial in the running time of $\mathcal{A}$, such that for every input vector $\vec{x}$,*

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

*If the running time of $\mathcal{S}$ is polynomial in the running time of $\mathcal{A}$, then we say that $\pi$* $t$-securely computes $f$ in the bounded secure channels setting.

**Black-box simulation.** In the sequel we use a more restricted notion of equivalence of computations, where the ideal-model adversary is limited to black-box simulation of the real-life setting. That is, for any semi-honest protocol $\pi'$ for $\pi$ there should exist a ideal-model adversary $\mathcal{S}$ with oracle (or black-box) access to a real-life adversary. This black-box represents the input-output relations of the real-life adversary described above. For concreteness, we present the following description of the "mechanics" of this black-box, representing a real-life adversary. The black-box has a random tape, where the black-box expects to find its random input, and an input-output tape. Once a special `start` input is given on the input-output tape, the interaction on this tape proceeds in iterations, as follows. Initially, no party is corrupted. In each iteration $l$, first the black-box expects to receive the information gathered in the $l$th round. (In the secure channels setting this information consists of the messages sent by the uncorrupted parties to the corrupted parties.) Next black-box outputs the messages to be sent by the corrupted parties in the $l$th round. Next, the black-box may issue several '`corrupt` $P_i$' requests. Such a request should be answered by the internal data of $P_i$, according to protocol $\pi'$. Also, from this point on $P_i$ is corrupted. At the end of the interaction, the output of the real-life adversary is defined as the contents of the random tape succeeded by the history of the contents of the input-output tape during the entire interaction. We let $\mathcal{S}^{\mathcal{A}}$ denote the ideal-model adversary $\mathcal{S}$ with black-box access to a real-life adversary $\mathcal{A}$.

The simulator is restricted to probabilistic polynomial time (where each invocation of the black-box is counted as one operation).[4] Furthermore, we limit the operation of the simulator as follows. We require that the `start` message is sent only once, and that no party is corrupted in the ideal model unless a request to corrupt this party is issued by the black-box.

If Definition 2.11 is satisfied by an ideal-model adversary limited to black-box simulation as described above, then we say that $\pi$ $t$-securely computes $f$ in a simulatable way. In this case we call the ideal-model adversary a black-box simulator, or in short a simulator.

We remark that the only purpose of the technical restrictions imposed on the operation of the simulator is to facilitate proving composition theorems (such as Theorem 3.4). In particular, black-box simulation is the only proof method currently known in the context of secure multiparty computation. The [BGW] protocols for computing any function can be proven secure, in the presence of non-erasing parties, using black-box simulators in probabilistic polynomial time.

[4] For simplicity, we assume that the computed function is polynomially computable. Alternatively, the simulator is polynomial in the complexity of the function.

## 2.4 Adaptively secure computation in the computational setting

We define adaptively secure multiparty computation in the computational setting. That is, we consider a synchronous network where the channels are **insecure**; the adversary sees all messages sent on all channels. (For simplicity we assume that the channels are **authenticated**, namely the adversary cannot *alter* the communication. Authenticity can be achieved via standard primitives.) All parties, as well as the adversary, are restricted to probabilistic polynomial time.[5] Furthermore, we introduce a **security parameter**, determining 'how close' a real-life computation is to a computation in the ideal model. All parties are polynomial also in the security parameter. For simplicity of presentation, we identify the security parameter with $n$, the number of parties.

The framework of defining adaptively secure multiparty computation in this setting is the same as in the secure channels setting (Section 2.3). That is, we compare the real life computation with a computation in the same ideal model, with the exception that here we restrict the ideal-model adversary to probabilistic polynomial time. The execution of a protocol $\pi$ in the real-life scenario, as well as the notations $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x})$ and $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x})$, are also the same as in the secure channels setting, with the exceptions that the real-life adversary is polynomially bounded and sees all the communication between the uncorrupted parties.

We define equivalence of a real-life computation to an ideal-model computation as in the secure channel setting, with the exception that here we use computational indistinguishability as our notion of similarity of distributions. Black-box simulation is defined as in the secure channels setting, with the exception that the information gathered by the adversary in each rounds includes the communication between all parties.

**Definition 2.12** *Let $f : D^n \to D'$ for some domains $D$ and $D'$, and let $\pi$ be a protocol for $n$ parties. We say that $\pi$ $t$-***securely computes** *$f$ in the computational scenario, if for any semi-honest protocol $\pi'$ for $\pi$ (according to any of the Definitions 2.5 through 2.7), and for any $t$-limited real-life adversary $\mathcal{A}$, there exists a polynomially bounded $t$-limited ideal-model-adversary $\mathcal{S}$, such that for every input vector $\vec{x}$,*

$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x}) \stackrel{c}{\approx} \text{EXEC}_{\pi',\mathcal{A}}(\vec{x}).$$

*If $\mathcal{S}$ is restricted to black-box simulation of real-life adversaries then we say that $\pi$ $t$-***simulatably computes** *$f$ in the computational scenario.*

**Remark:** Our convention that the real-life adversary outputs its entire view is important for clarifying how the following difficulty, pointed out in [MR], is settled. Assume that the corrupted parties do not output their inputs and random inputs, and that the function $f$ to be computed is pseudorandom. Then an insecure protocol that allows a uniformly distributed output vector $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x})$, regardless of the parties' inputs, could be considered

[5] For simplicity, we assume that the computed function is polynomially computable. Alternatively, all parties, as well as the real-life adversary and the ideal-model-adversary, should be polynomial in the complexity of the function.

secure since it generates outputs indistinguishable from the parties' outputs in the ideal model (i.e., $\text{IDEAL}_{f,\mathcal{S}}(\vec{x})$). We stress that no generality is lost by using our convention, since Definition 2.12 quantifies over all real-life adversaries.