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 π 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 π 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 adversary is required to very closely mimic the operation of the real-life adversary, down to precise details. In particular, the ideal-model adversary is limited to creating a simulated environment for the real-life adversary (that looks the same as a real environment), via a special type of black-box simulation. In [Be] a different approach is pursued. First a general notion of comparing security of protocols is formulated, as follows. Consider two protocols α and β for computing the same function. Essentially, protocol α is at least as secure as protocol β if an adversary attacking α cannot affect the outputs of the parties more than an adversary attacking β. (Here some technical "interface" algorithms are used when α and β operate in different adversary models.) Next, a protocol for evaluating a function is secure if it at least as secure as the trivial protocol for evaluating the function in the ideal model. We remark that, although their approach is general, both Micali and Rogaway and Beaver formalize their definitions only in the secure 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 π 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 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

[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.

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 π' is a **semi-honest protocol** for a protocol π if a party running π' "appears as" an honest party running π. (We define this notion more precisely in Chapter 2.) When the parties are not thoroughly trusted we want the requirements from π to be satisfied even if the uncorrupted parties are running some semi-honest protocol for π. 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 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

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

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^{th} party should get the i^{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 $\vec{x} = x_1 \dots x_n$ 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 $\vec{x} = x_1 \dots x_n$, a set $B \subseteq [n]$, and a $|B|$-vector $\vec{b} = b_1 \dots b_{|B|}$, 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 $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}) = \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_1 \dots \text{IDEAL}_{f,\mathcal{S}}(\vec{x})_n$ 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 π in the real-life scenario. The parties engage in a synchronous computation in the secure channels setting, running protocol π. 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 π 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 π on input $\vec{x} = x_1 \dots x_n$ and random input $\vec{r} = r_1 \dots r_n$, 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 $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x}) = \text{EXEC}_{\pi,\mathcal{A}}(\vec{x})_1 \dots \text{EXEC}_{\pi,\mathcal{A}}(\vec{x})_n$. (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 π 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 π, 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 $f : D^n \to D$ for some domains D and D, and let π be a protocol for n parties. We say that π* **(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 π **(non-adaptively) t-securely computes f in the bounded secure channels setting**.

Remarks:

- For measuring complexity, we assume that the protocol π, 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 π, $\mathcal{S}$ 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 π and π' be n-party protocols. We say that π' is a* non-erasing *protocol for π if π' is identical to π with the exception that, in addition to the instructions of π, protocol π' copies the contents of each memory location accessed by π 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 π 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 ρ and π and an index $i \in [n]$, let $\rho_{/(i,\pi)}$ denote the protocol where party P_i executes π and all the other parties execute ρ.

Definition 2.6 *Let π and π' be n-party protocols. We say that π' is a* perfectly honest-looking *protocol for π if for any input $\vec{x}$, for any n-party "test" protocol ρ, 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 ρ 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 π' is a computationally honest-looking *protocol for π.*

We stress that here the "test" protocol ρ 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 π, rather than with respect to any test protocol ρ.

Definition 2.7 *Let π and π' be n-party protocols. We say that π' is an* perfectly weakly-honest *protocol for π 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 π 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 π' is a computationally weakly-honest *protocol for π.*

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 π.

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* $0 \leq i < t$*, and as long as* $b_i \neq \perp$*, 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 $i^* \leq i < t$*, 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* $1 \leq i \leq n$*:*

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

where $B^{(t)}$ *is the* t^{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 π in an (adaptive) real-life scenario. The parties engage in a synchronous computation in the secure channels setting, running some semi-honest protocol π' for π (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 π') 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* π *be a protocol for* n *parties. We say that* π t-securely computes f in the unbounded secure channels setting*, if for any semi-honest protocol* π' *for* π *(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 π 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 π' for π 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 lth 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 lth 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 π'. 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 π 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 π 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 π be a protocol for n parties. We say that π t-***securely computes** *f in the computational scenario, if for any semi-honest protocol π' for π (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 π 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.

Chapter 3

Adaptively secure computation in the computational setting

We study secure multiparty computation in the presence of adaptive adversaries (see Section 1.3 for an introductory discussion). Although the emphasis of this chapter is on the computational setting, we also sketch the state of affairs in the secure channels setting. We believe that understanding adaptively secure computation in the computational setting is easier if the secure channels setting is first considered.

We first present, in Section 3.1, an overview of the problems encountered when trying to prove adaptive security of protocols, first in the secure channels setting, and then in the computational settings. We also sketch our solution for the computational setting. Next, in Section 3.2 we define a tool, called non-committing encryption, that is central in our solution for the computational setting. In Section 3.3 we present our construction for the case of non-erasing parties. We first show how, given a non-committing encryption scheme, any adaptively secure protocol in the secure channels setting can be transformed into an adaptively secure protocol in the computational setting. Next we describe our construction of a non-committing encryption scheme. In Section 3.4 we suggest a construction for the case of honest-looking parties.

3.1 The problems in proving adaptive security: informal presentation

3.1.1 The secure channels setting

The state-of-the-art with respect to adaptive computation in the secure channels setting can be briefly summarized as follows. Adaptively secure protocols for computing any function exist in the presence of non-erasing parties (e.g., [BGW, CCD]). However, in contrast with popular belief, not every *non-adaptively* secure protocol is also *adaptively* secure in the presence of non-erasing parties. Furthermore, current techniques are insufficient for proving adaptive security of any protocol for computing a non-trivial function in the presence of honest-looking parties.

A standard construction of an ideal-model-adversary, $\mathcal{S}$, operates via black-box interaction with the real-life adversary $\mathcal{A}$. (The exact "mechanics" of the black-box representing $\mathcal{A}$ are specified in Section 2.3.) More specifically, let π' be a semi-honest protocol for π. $\mathcal{S}$ runs the black-box representing $\mathcal{A}$ on a simulated interaction with a set of parties running π'. $\mathcal{S}$ corrupts (in the ideal model) the same parties that $\mathcal{A}$ corrupts in the simulated interaction, and outputs whatever $\mathcal{A}$ outputs. From the point of view of $\mathcal{A}$, the interaction simulated by $\mathcal{S}$ should be distributed identically to an authentic interaction with parties running π'. It is crucial that $\mathcal{S}$ be able to run a successful simulation based only on the information available to it in the ideal model, and in particular *without knowing the inputs of uncorrupted parties.* We restrict our presentation to this methodology of proving security of protocols, where $\mathcal{S}$ is restricted to probabilistic polynomial time. We remark that no other proof method is known in this context. In the sequel we often call the ideal-model-adversary $\mathcal{S}$ a simulator.

Following the above methodology, the simulator that we construct has to generate simulated messages from the uncorrupted parties to the corrupted parties. In the non-adaptive case the set of corrupted parties is fixed and known to the simulator. Thus the simulator can corrupt these parties, in the ideal model, before the simulation starts. In the adaptive case the corrupted parties are chosen by the simulated adversary $\mathcal{A}$ as the computation unfolds. Here the simulator corrupts a party, in the ideal model, only when the simulated adversary decides on corrupting that party. Thus the following extra problem is encountered. Consider a currently uncorrupted party P. Since $\mathcal{S}$ does not know the input of P, it may not know which messages should be sent by P to the corrupted parties. Still, $\mathcal{S}$ has to generate some dummy messages to be sent by the simulated P to corrupted parties. When the simulated adversary $\mathcal{A}$ later corrupts P it expects to see P's internal data. The simulator should now be able to present internal data for P that is consistent with P's newly-learned input and with the messages previously sent by P, according to *the particular semi-honest protocol* π' run by P. It turns out that this can be done for the [BGW] protocols for computing any function in the presence of non-erasing parties. Thus, the [BGW] protocols are adaptively secure *in the presence of non-erasing parties.* We stress, however, that not every protocol which is secure against non-adaptive adversaries is also secure against adaptive adversaries.[1]

In face of honest-looking parties. Even more severe problems are encountered when honest-looking parties are allowed, as demonstrated by the following example. Consider a protocol β that instructs each party, on private input σ, to just publicize a uniformly and independently chosen value r in some domain D and terminate. Let f_0, f_1 be a claw-free pair of permutations over D. Then, on input $\sigma \in \{0,1\}$, an honest-looking party can 'commit' to its input by publicizing $f_\sigma(r)$ instead of publicizing r. Now, if this honest-looking variant of β is shown secure via an efficient black-box simulation as described above, then the constructed simulator can be used to find claws between f_0 and f_1. Similar honest-looking protocols can be constructed for the [BGW, CCD] protocols. Consequently, if claw-free pairs of permutations exist then adaptive security of the [BGW, CCD] protocols, *in the presence of honest-looking parties*, cannot be proven via black-box simulation.

[1] See example in the third paragraph of the Introduction.

3.1.2 Adaptive security in the computational setting

In this subsection we sketch the extra difficulty encountered in constructing adaptively secure protocols *in the computational setting*, and outline our solution for non-erasing parties. Consider the following folklore methodology for constructing secure protocols in the computational setting. Start with an adaptively secure protocol π *resilient against non-erasing parties in the secure channels setting*, and construct a protocol $\tilde{\pi}$ by encrypting each message using a standard encryption scheme. We investigate the security of $\tilde{\pi}$ in the computational setting.

Proving that $\tilde{\pi}$ is non-adaptively secure. We first sketch how $\tilde{\pi}$ can be shown *non-adaptively* secure in the computational setting, assuming that π is non-adaptively secure in the secure channels setting. Let $\mathcal{S}$ be the ideal-model-adversary (simulator) associated with π in the secure channels setting. (We assume that $\mathcal{S}$ operates via "black-box simulation" of the real-life adversary $\mathcal{A}$ as described above.) We wish to construct, in the computational setting, a simulator $\tilde{\mathcal{S}}$ for $\tilde{\pi}$. The simulator $\tilde{\mathcal{S}}$ operates just like $\mathcal{S}$, with the following exception. In the computational setting the real-life adversary expects to see the ciphertexts sent between uncorrupted parties. (In the secure channels setting the adversary does not see the communication between uncorrupted parties.) Furthermore, the real-life adversary expects that the messages sent to corrupted parties be encrypted. $\tilde{\mathcal{S}}$ will imitate this situation as follows. First each message sent to a corrupted party will be appropriately encrypted. Next, the simulated uncorrupted parties will exchange dummy ciphertexts. (These dummy ciphertexts can be generated as, say, encryptions of the value '0'.) The validity of simulator $\tilde{\mathcal{S}}$ can be shown to follow, in a straightforward way, from the validity of $\mathcal{S}$ and the security of the encryption scheme in use.

Problems with proving adaptive security. When adaptive adversaries are considered, the construction of a simulator $\tilde{\mathcal{S}}$ in the computational setting encounters the following problem which is a more severe version of the problem encountered in the secure channels setting. Consider an uncorrupted party P. Since $\tilde{\mathcal{S}}$ does not know the input of P, it does not know which messages should be sent by P to other *un*corrupted parties.[2] Still, $\tilde{\mathcal{S}}$ has to generate dummy ciphertexts to be sent by the simulated P to uncorrupted parties. These dummy ciphertexts are seen by the simulated adaptive adversary. When the simulated adversary later corrupts P, it expects to see all of P's internal data, as specified *by the semi-honest protocol* π'. Certainly, this data may include the cleartexts of all the ciphertexts sent and received by P in the past, including the random bits used for encryption and decryption, respectively. Thus, it may be the case that some specific dummy ciphertext c was generated as an encryption of '0', and the simulated P now needs to "convince" the adversary that c is in fact an encryption of '1' (or vice versa). This task is impossible if a standard encryption scheme (i.e., an encryption scheme where no ciphertext can be a legal encryption of both '1' and '0') is used.

[2] There is also the easier problem of generating the messages sent by P to corrupted parties. This was the problem discussed in the previous subsection. However, our hypothesis that $\mathcal{S}$ is a simulator for the secure channel model means that $\mathcal{S}$ is able to generate these cleartext messages. Thus, all that $\tilde{\mathcal{S}}$ needs to do is encrypt the messages it has obtained from $\mathcal{S}$.

We remark that Beaver and Haber [BH] have suggested to solve this problem as follows. Instruct each party to *erase* (say, at the end of each round) all the information involved with encrypting and decrypting of messages. If the parties indeed erase this data, then the adversary will no longer see, upon corrupting a party, how past messages were encrypted and decrypted. Thus the problem of convincing the adversary in the authenticity of past ciphertexts no longer exists. Consequently, such "erasing" protocols can be shown adaptively secure in the computational setting. However, this approach is clearly not valid in the presence of semi-honest parties. In particular, it is not known whether the [BH] protocols (or any other previous protocols) are secure in the presence of non-erasing parties.

Sketch of our solution. We solve this problem by constructing, in the multi-party computational setting, an encryption protocol that serves as an alternative to standard encryption schemes, and enjoys an additional property roughly described as follows. The additional property is that one can efficiently generate dummy ciphertexts that can later be "opened" as encryptions of either '0' or '1', at wish. (Here the word 'ciphertext' is used to denote all the information seen by the adversary during the execution of the protocol.) These dummy ciphertexts are different and yet computationally indistinguishable from the valid encryptions of '0' (or '1') produced in a real communication. We call such encryption protocols non-committing.[3]

Let $\mathcal{E}^{(0)}$ (resp., $\mathcal{E}^{(1)}$) denote the distribution of encryptions of the value 0 (resp., 1) in a public-key encryption scheme. For simplicity, suppose that each of these distributions is generated by applying an efficient deterministic algorithm, denoted $A^{(0)}$ (resp., $A^{(1)}$), to a uniformly selected n-bit string.[4] In a traditional encryption scheme (with no decryption errors) the supports of $\mathcal{E}^{(0)}$ and $\mathcal{E}^{(1)}$ are disjoint (alas $\mathcal{E}^{(0)}$ and $\mathcal{E}^{(1)}$ are computationally indistinguishable). In a non-committing encryption scheme, the supports of $\mathcal{E}^{(0)}$ and $\mathcal{E}^{(1)}$ are not disjoint but the probability that an encryption (of either '0' or '1') resides in their intersection, denoted I, is negligible. Thus, decryption errors occur only with negligible probability. However, it is possible to efficiently generate a distribution $\mathcal{E}^{\mathrm{amb}}$ which assumes values in I so that this distribution is computational indistinguishable from both $\mathcal{E}^{(0)}$ and $\mathcal{E}^{(1)}$.[5] Furthermore, each "ambiguous ciphertext" $c \in I$ is generated together with two random looking n-bit strings, denoted r_0 and r_1, so that $A^{(0)}(r_0) = A^{(1)}(r_1) = c$. That is, the string r_0 (resp., r_1) may serve as a witness to the claim that c is an encryption of '0' (resp., '1').

Using a non-committing encryption protocol, we resolve the simulation problems which were described above. Firstly, when transforming π into $\tilde{\pi}$, we replace every bit transmission of π by an invocation of the non-committing encryption protocol. This allows us to generate dummy ciphertexts for messages sent between uncorrupted parties so that at a later stage we can substantiate for each such ciphertext both the claim that it is an encryption of '0' and the claim that it is an encryption of '1'. We stress that although dummy ciphertexts appear

[3] This "non-committing property" is reminiscent of the "Chameleon blobs" of [Br]. Those are *commitment* schemes where the recipient of a commitment c can generate by himself de-commitments of c to both 0 and 1. Here we consider *encryption* schemes where an adversary can generate by himself ciphertexts which can be opened both as encryptions of 1 and as encryptions of 0.

[4] This is an over simplification. Actually, each of these algorithms is also given an n-bit encryption key.

[5] Consequently, it must be that $\mathcal{E}^{(0)}$ and $\mathcal{E}^{(1)}$ are computationally indistinguishable. Thus, a non-committing encryption scheme is also a secure encryption scheme in the traditional sense.

with negligible probability in a real execution, they are computationally indistinguishable from a uniformly generated encryption of either '0' or '1'. Thus, using a non-committing encryption protocol we construct *adaptively secure protocols* for computing any (recursive) function *in the computational model in the presence of non-erasing parties.* Finally, we construct a non-committing encryption protocol based on the intractability of inverting the RSA, or more generally based on the existence of common-domain trapdoor systems (see Definition 3.5). Thus, we get

Theorem 3.1 *If common-domain trapdoor systems exist, then there exist secure protocols for computing any (recursive) function in the computational setting, in the presence of non-erasing parties and adaptive adversaries that corrupt less than a third of the parties.*

We remark that, using standard constructs (e.g., [RB]), our protocols can be modified to withstand adversaries that corrupt less than half of the parties.

Dealing with honest-looking parties. We also sketch a solution for the case of honest-looking parties, assuming, in addition to the above, also the existence of a "trusted dealer" at a pre-computation stage. The dealer hands each party P a truly random string r_P, to be used as random input. Next, the dealer hands the other parties shares of r_P, so that a coalition of all parties other than P can reconstruct r_P. These shares enable us to "force" each party to send messages according to the specification of the protocol. We stress that this result does not hold if an initial (trusted) set-up is not allowed.

3.2 Defining non-committing encryption

We present a concise definition of a non-committing encryption protocol in our multi-party scenario. First define the bit transmission function $\text{BTR} : \{0,1,\perp\}^n \rightarrow \{0,1,\perp\}^n$. This function is parameterized by two identities of parties (i.e., indices $s, r \in [n]$), with the following interpretation. $\text{BTR}_{s,r}$ describes the secure transmission of a bit from party P_s (the sender) to party P_r (the receiver). That is, for $\vec{x} = x_1, \ldots, x_n \in \{0,1,\perp\}^n$ let

$$\text{BTR}_{s,r}(\vec{x})_i = \begin{cases} x_s & \text{if } \ i = r \\ \perp & \text{otherwise} \end{cases}$$

where $\text{BTR}_{s,r}(\vec{x})_i$ is the i^{th} component of the vector $\text{BTR}_{s,r}(\vec{x})$. We are interested in input vectors $\vec{x}$ where x_s (i.e., the senders input) is in $\{0,1\}$. All other inputs are assumed to be $\perp$.

Definition 3.2 *Let $s, r \in [n]$ and $s \neq r$. A protocol ε is a t-resilient (in the presence of $\mathcal{T}$-semi-honest parties and adaptive adversaries),* non-committing encryption *protocol (from P_s to P_r) if ε t-securely computes $\text{BTR}_{s,r}$, in a simulatable way, in the computational model, in the presence $\mathcal{T}$-semi-honest parties and an adaptive adversary.*

It may not be immediately evident how Definition 3.2 corresponds to the informal description of non-committing encryptions, presented in Section 3.1.2. A closer look, however, will show that the requirements from the simulator associated with a non-committing encryption protocol (according to Definition 3.2) imply these informal descriptions. In particular, in the case where the simulated adversary corrupts the sender and receiver only after the last communication round, the simulator has to first generate some simulated communication between the parties, without knowing the transmitted bit. (This communication serves as the "dummy ciphertext".) When the sender and/or the receiver are later corrupted, the simulator has to generate internal data that correspond to any value of the transmitted bit.

3.3 A solution for non-erasing parties

We show that any function can be securely computed in the computational setting, in the presence of adaptive adversaries and non-erasing parties. In Subsection 3.3.1 we show how, using a non-committing encryption protocol, a simulatable protocol for computing some function f in the computational setting can be constructed from any simulatable protocol for computing f in the secure channels setting. In Subsection 3.3.2 we present our construction of non-committing encryption. We use the following result, attributed to [BGW, CCD], as our starting point:[6]

Theorem 3.3 *The* [BGW, CCD] *protocols for computing any function of n inputs are $(\lceil\frac{n}{3}\rceil - 1)$-securely computable in a simulatable way,* in the secure channels setting, *in the presence of non-erasing parties and adaptive adversaries.*

3.3.1 Adaptively secure computation given non-committing encryption

Theorem 3.4 *Let f be an n-ary function, $t < n$ and π be a protocol that t-securely computes f in a simulatable way* in the secure channels setting, *in the presence of non-erasing parties and adaptive adversaries. Suppose that $\varepsilon_{s,r}$ is a t-resilient non-committing encryption protocol, resilient to non-erasing parties and adaptive adversaries, for transmission from P_s to P_r. Let $\tilde{\pi}$ be the protocol constructed from π as follows. For each bit σ transmitted by π from party P_s to party P_r, protocol $\tilde{\pi}$ invokes a copy of a $\varepsilon_{s,r}$ for transmitting σ. Then $\tilde{\pi}$ t-securely computes f, in a simulatable way* in the computational setting, *in the presence of non-erasing parties and adaptive adversaries.*

Proof (sketch): Let π' be a non-erasing protocol for π and let $\mathcal{S}$ be a simulator for π' *in the secure channels setting.* For simplicity we assume that in protocol π, as well as in the interaction generated by $\mathcal{S}$, each party sends on bit to each other party in each round. Let δ be the (computational-model) simulator that corresponds to the non-erasing protocol ε' for the non-committing encryption protocol ε. Given these two different simulators, we construct a simulator $\tilde{\mathcal{S}}$ for protocol $\tilde{\pi}$ in the computational setting. The simulator $\tilde{\mathcal{S}}$ will be a modification of $\mathcal{S}$ and will use several copies of δ as subroutines.

Recall that $\mathcal{S}$ is supposed to interact with a black-box representing a real-life adversary *in the secure channels setting.* That is, at each round $\mathcal{S}$ generates all the messages sent from uncorrupted parties to corrupted parties. Furthermore, whenever the black-box decides to

[6] A proof of this result can be extracted from Chapter 4, which deals with the more involved asynchronous model.

corrupt some party P, machine $\mathcal{S}$ generates internal data for P which is consistent with P's input and with the messages previously sent by P to corrupted parties.

The simulator $\tilde{\mathcal{S}}$, interacts with a black box representing an arbitrary real-life adversary *in the computational setting*, denoted $\tilde{\mathcal{A}}$. The simulator $\tilde{\mathcal{S}}$ is identical to $\mathcal{S}$ with the exception that for each bit sent in the interaction simulated by $\mathcal{S}$, the simulator $\tilde{\mathcal{S}}$ invokes a copy of δ and $\tilde{\mathcal{S}}$ incorporates the outputs of the various copies of δ in its (i.e., $\tilde{\mathcal{S}}$'s) communication with $\tilde{\mathcal{A}}$. Likewise, $\tilde{\mathcal{S}}$ extracts the transmitted bits from the invocations of δ corresponding to message transmissions from corrupted parties to uncoruppted ones. (The way $\tilde{\mathcal{S}}$ handles these invocation will be discussed below.) At this point we stress that $\tilde{\mathcal{A}}$ is the only adversary that $\tilde{\mathcal{S}}$ needs to simulate and to this end it "emulates" real-life adversaries of its choice for the copies of δ. In particular, when $\mathcal{S}$ asks to corrupt some party P, the simulator $\tilde{\mathcal{S}}$ corrupts the same party P. When $\mathcal{S}$ generates P's view *in the secure channel setting*, $\tilde{\mathcal{S}}$ will complete this view into P's view *in the computational setting* by using the various copies of δ.

We describe how $\tilde{\mathcal{S}}$ handles the various copies of δ. As stated above, $\tilde{\mathcal{S}}$ emulates a real-life adversary for each copy of δ using the communication tapes by which this copy is supposed to interact with its black-box/adversary. The information that δ expects to receive form its black box is extracted, in the obvious manner, from the information that $\tilde{\mathcal{S}}$ receives from $\tilde{\mathcal{A}}$. That is, $\tilde{\mathcal{S}}$ hands δ the messages, sent by the corrupted parties, that are relevant to the corresponding invocation of ε'. Furthermore, all the past and current requests for corrupting parties (issued by $\tilde{\mathcal{A}}$) are handed over to δ. The partial view received from each copy of δ is used in the emulation of the corresponding black-box (of this δ-copy) as well as incorporated in the information handed by $\tilde{\mathcal{S}}$ to $\tilde{\mathcal{A}}$. When $\tilde{\mathcal{A}}$ asks to corrupt some party P, the simulator $\tilde{\mathcal{S}}$ emulates a 'corrupt P' request to each copy of δ and obtains the internal data of P in the corresponding sub-protocol ε which it (i.e., $\tilde{\mathcal{S}}$) hands to $\tilde{\mathcal{A}}$ (along with the information obtained by $\mathcal{S}$ – the secure channel simulator). Finally, observe that $\delta = \delta_{s,r}$ (where P_s and P_r are the designated sender and receiver) also expects to interact with parties in the ideal-model. This interaction consists of issuing 'corrupt' requests and obtaining the internal data (of the ideal model). This interaction is (also) emulated by $\tilde{\mathcal{S}}$ as follows. Whenever δ wishes to corrupt a party P which is either P_s or P_r, the simulator $\tilde{\mathcal{S}}$ finds out which bit, σ, was supposed to be sent in this invocation of $\varepsilon'_{r,s}$ and passes σ to $\delta_{r,s}$. We stress that σ is available to $\tilde{\mathcal{S}}$ since at this point in time P has already been corrupted and furthermore $\tilde{\mathcal{S}}$ (which mimics $\mathcal{S}$) has already obtained P's view in the secure channel setting. (Here we use Definitions 2.12 and 3.2 which guarantee that δ corrupts a party only if this party is already corrupted by δ's black box. We also use the fact that $\tilde{\mathcal{S}}$ is playing δ's black box and is issuing a 'corrupt P' request only after receiving such a request from $\tilde{\mathcal{A}}$ and having simulated this corruption as $\mathcal{S}$.) In case P is neither P_s not P_r the simulator $\tilde{\mathcal{S}}$ passes $\perp$ (as P's input) to δ.

Let $\tilde{\pi}'$ be a non-erasing protocol for $\tilde{\pi}$ and $\tilde{\mathcal{A}}$ be as above (i.e., an arbitrary real-life adversary in the computational setting). We claim that $\tilde{\mathcal{S}}^{\tilde{\mathcal{A}}}$ (i.e., the ideal-model adversary $\tilde{\mathcal{S}}$ with black-box access to $\tilde{\mathcal{A}}$) properly simulates the execution of $\tilde{\pi}'$. We need to show that for any adversary $\tilde{\mathcal{A}}$ and for any input $\vec{x}$ we have

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

Here we present only a rough sketch of the proof of this claim. The plan is to construct a real-life adversary $\mathcal{A}$ in the secure channels setting, and prove the following sequence of equalities by which the above claim follows:

$$\text{IDEAL}_{f,\tilde{\mathcal{S}}^{\tilde{\mathcal{A}}}}(\vec{x}) \stackrel{\text{d}}{=} \text{IDEAL}_{f,\mathcal{S}^{\mathcal{A}}}(\vec{x}) \stackrel{\text{d}}{=} \text{EXEC}_{\pi',\mathcal{A}}(\vec{x}) \stackrel{\text{c}}{\approx} \text{EXEC}_{\tilde{\pi}',\tilde{\mathcal{A}}}(\vec{x}) \qquad (3.1)$$

Regardless of what $\mathcal{A}$ is, the second equality follows immediately from the hypothesis that $\mathcal{S}$ is a simulator for π' (the non-erasing protocol for π) *in the secure channels setting.* It remains to construct $\mathcal{A}$ so that the other two equalities hold.

The real-life adversary $\mathcal{A}$ of the secure channel setting will operate via a simulation of $\tilde{\mathcal{A}}$ (the real-life adversary of the computational setting), imitating the simulation carried out by $\tilde{\mathcal{S}}$. That is, for each bit communicated by π, machine $\mathcal{A}$ will invoke a copy of δ while emulating an adversary in accordance with $\tilde{\mathcal{A}}$. In particular, $\tilde{\mathcal{A}}$ will be given all ciphertexts sent in the open as well as all internal data of corrupted parties (regardless if these parties were corrupted before, during or after the 'real' transmission). Furthermore, when $\tilde{\mathcal{A}}$ corrupts a party P, machine $\mathcal{A}$ corrupts P and hands $\tilde{\mathcal{A}}$ the internal data of P, along with the outputs of the relevant copies δ, just as $\tilde{\mathcal{S}}$ does. At the end of the computation $\mathcal{A}$ outputs whatever $\tilde{\mathcal{A}}$ outputs (that is, $\mathcal{A}$ outputs $\tilde{\mathcal{A}}$'s view of the computation). It follows from the definition of $\mathcal{A}$ that the execution of $\mathcal{S}$, with black-box access to $\mathcal{A}$, is in fact identical to the execution of $\tilde{\mathcal{S}}$ with black-box access to $\tilde{\mathcal{A}}$. Thus, $\text{IDEAL}_{f,\tilde{\mathcal{S}}^{\tilde{\mathcal{A}}}}(\vec{x}) \stackrel{\text{d}}{=} \text{IDEAL}_{f,\mathcal{S}^{\mathcal{A}}}(\vec{x})$ which establishes the first equality in Eq. (3.1).

It remains to show that $\text{EXEC}_{\pi',\mathcal{A}}(\vec{x}) \stackrel{\text{c}}{\approx} \text{EXEC}_{\tilde{\pi}',\tilde{\mathcal{A}}}(\vec{x})$. Essentially the difference between these two executions is that $\text{EXEC}_{\pi',\mathcal{A}}(\vec{x})$ is a real-life execution in the secure channel setting which is augmented by invocations of δ (performed by $\mathcal{A}$), whereas $\text{EXEC}_{\tilde{\pi}',\tilde{\mathcal{A}}}(\vec{x})$ is a real-life execution in the computational setting in which honest parties use the encryption protocol ε'. However, the security of ε means that invocations of δ are indistinguishable from executions by ε' (both in presence of adaptive adversaries). Using induction on the number of rounds, one thus establishes the last equality of Eq. (3.1). $\square$

3.3.2 Constructing non-committing encryption

Before describing our non-committing encryption protocol, let us note that one-time-pad is a valid non-committing encryption protocol.[7] The drawback of this trivial solution is that it requires an initial set-up in which each pair of parties share a random string of length at least the number of bits they need to exchange. Such an initial set-up is not desirable in practice and does not resolve the theoretically important problem of dealing with a setting in which *no secret information is shared a-priori.*

Our scheme uses a collection of trapdoor permutations together with a corresponding hard-core predicate [BM, Y2, GrL]. Actually, we need a collection of trapdoor permutation with the additional property that they are many permutations over the same domain.

[7] Assume that each pair of parties share a sufficiently long secret random string, and each message is encrypted by bitwise xor-ing it with a new segment of the shared random string. Then Definition 3.2 is satisfied in a straightforward way. Specifically, the simulated message from the sender to the receiver (i.e., the dummy ciphertext), denoted c, can be uniformly chosen in $\{0,1\}$. When either the sender or the receiver are corrupted, and the simulator has to demonstrate that c is an encryption of a bit σ, the simulator claims that the corresponding shared random bit was $r = c \oplus \sigma$. Clearly r is uniformly distributed, regardless of the value of σ.

Furthermore, we assume that given a permutation f over a domain D (but not f's trapdoor), one can efficiently generate at random another permutation f' over D together with the trapdoor of f'. Such a collection is called a common-domain trapdoor system.

Definition 3.5 *A* common-domain trapdoor system *is an infinite set of finite permutations* $\{f_{\alpha,\beta} : D_\alpha \stackrel{1\text{-}1}{\mapsto} D_a\}_{(\alpha,\beta)\in P}$*, where* $P \subseteq \{0,1\}^* \times \{0,1\}^*$*, so that*

- domain selection: *There exists a probabilistic polynomial-time algorithm* G_1 *so that on input* 1^n*, algorithm* G_1 *outputs a description* $\alpha \in \{0,1\}^n$ *of domain* D_α*.*
- function selection: *There exists a probabilistic polynomial-time algorithm* G_2 *so that on input* α*, algorithm* G_2 *outputs a pair* $(\beta, t(\beta))$ *so that* $(\alpha,\beta) \in P$*. (*β *is a description of a permutation over* D_α *and* $t(\beta)$ *is the corresponding trapdoor.)*
- domain sampling: *There exists a probabilistic polynomial-time algorithm* S *that on input* α*, uniformly selects an element of* D_a*.*
- function evaluation: *There exists a polynomial-time algorithm* F *that on inputs* $(\alpha,\beta) \in P$ *and* $x \in D_\alpha$ *returns* $f_{\alpha,\beta}(x)$*.*
- function inversion: *There exists a polynomial-time algorithm* I *that on inputs* $(\alpha, t(\beta))$ *and* $y \in D_\alpha$*, where* $(\alpha,\beta) \in P$*, returns* $f^{-1}_{\alpha,\beta}(y)$*.*
- one-wayness: *For any probabilistic polynomial-time algorithm* A*, the probability that on input* $(\alpha,\beta) \in P$ *and* $y = f_{\alpha,\beta}(x)$*, algorithm* A *outputs* x *is negligible (in* n*), where the probability distribution is over the random choices of* $\alpha = G_1(1^n)$*,* $\beta = G_2(\alpha)$*,* $x = S(\alpha)$ *and the coin tosses of algorithm* A*.*

Remarks:

- The standard definition of trapdoor permutations can be derived from the above by replacing the two selection algorithms, G_1 and G_2, by a single algorithm G that on input 1^n generates a pair $(\beta, t(\beta))$ so that β specifies a domain D_β as well as a permutation f_β over this domain (and $t(\beta)$ is f_β's trapdoor). Thus, the standard definition does not guarantee any structural resemblance among domains of different permutations. Furthermore, it does not allow to generate a new permutation with corresponding trapdoor for a given domain (or given permutation). Nevertheless some popular trapdoor permutations can be formulated in a way which essentially meets the requirements of a common-domain trapdoor system.
- Common-domain trapdoor systems can be constructed based on an arbitrary family of trapdoor permutations, $\{f_\beta : D_\beta \stackrel{1\text{-}1}{\mapsto} D_\beta\}$, with the extra property that the domain of any permutation, generated on input 1^n, has non-negligible density inside $\{0,1\}^n$ (i.e., $|D_\beta| \geq \frac{1}{\text{poly}(|\beta|)} \cdot 2^{|\beta|}$). We construct a common-domain family where the domain is $\{0,1\}^n$ and the permutations are natural extensions of the given permutations. That is, we let $G_1(1^n) = 1^n$, $G_2(1^n) = G(1^n)$ and extend f_β into g_β so that $g_\beta(x) = f_\beta(x)$ if $x \in D_\beta$ and $g_\beta(x) = x$ otherwise. This yields a collection of "common-domain" permutations, $\{g_\beta : \{0,1\}^{|\beta|} \stackrel{1\text{-}1}{\mapsto} \{0,1\}^{|\beta|}\}$, which are weakly one-way. Employing amplification techniques (e.g., [Y2, GILVZ]) we obtain a proper common-domain system.

In the sequel we refer to common-domain trapdoor systems in a less formal way. We say that two one-way permutations, f_a and f_b, are a **pair** if they are both permutations over the same domain (i.e., $a = (\alpha, \beta_1)$ and $b = (\alpha, \beta_2)$, where the domain is D_α). We associate the permutations with their descriptions (and the corresponding inverse permutations with their trapdoors). Finally, as stated above, we augment any common-domain trapdoor system with a hard-core predicate, denoted B. (That is, B is polynomial-time computable, but given (f_a and) $f_a(x)$ is it infeasible to predict $B(x)$ with non-negligible advantage over 1/2.)

Outline of our scheme. The scheme consists of two stages. In the first stage, called the **key generation** stage, the parties arrive at a situation where the sender has two trapdoor permutations f_a, f_b of a common-domain system, the trapdoor of only *one* of which is known to the receiver. Furthermore, the simulator will be able to generate, in a simulated execution of the protocol, two trapdoor permutations with the same distribution as in a real execution and such that the trapdoors of both permutations are known. (The simulator will later open dummy ciphertexts as either '0' or '1' by claiming that the decryption key held by the receiver is either f_a^{-1} or f_b^{-1}. The correspondence between $\{0, 1\}$ and $\{a, b\}$ will be chosen at random by the simulator and never revealed). The key generation stage is independent of the bit to be transmitted (and can be performed before this bit is even determined).

Our most general implementation of this stage, based on any common-domain system, requires participation of all parties. It is described in Section 3.3.2. In the implementations based on the RSA and DH assumptions (see Section 3.3.3) the key-generation stage consists of only one message sent from the receiver to the sender.

The second stage, in which the actual transmission takes place, consists of only one message sent from the sender to the receiver. This stage consists of **encryption** and **decryption** algorithms, invoked by the sender and the receiver respectively.

We first present, in Section 3.3.2, the encryption and decryption algorithms as well as observations that will be instrumental for the simulation. In Section 3.3.2 we present the key generation protocol. (A reader that is satisfied with a construction based on specific number theoretic assumptions may, for simplicity, skip Section 3.3.2 and read Section 3.3.3 instead.) Finally we show that these together constitute the desired non-committing encryption protocol.

Encryption and decryption

Let f_a and f_b be two randomly selected permutations over the domain D, and let B be a hard-core predicate associated with them. The scheme uses a security parameter, k, which can be thought to equal $\log_2 |D|$.

Encryption: to encrypt a bit $\sigma \in \{0, 1\}$ with encryption key (f_a, f_b), the sender proceeds as follows. First it chooses $x_1, \ldots, x_{8k}$ at random from D, so that $B(x_i) = \sigma$ for $i = 1, ..., 5k$ and $B(x_i) = 1 - \sigma$ otherwise (i.e., for $i = 5k + 1, ..., 8k$). For each x_i it computes $y_i = f_a(x_i)$. These x_i's (and y_i's) are **associated with** f_a (or with a). Next, it repeats the process with respect to f_b. That is, $x_{8k+1}, \ldots, x_{16k}$ are chosen at random from D, so that $B(x_i) = \sigma$ for $i = 8k + 1, ..., 13k$ and $B(x_i) = 1 - \sigma$ otherwise, and $y_i = f_b(x_i)$ for $i = 8k + 1, ..., 16k$. The latter x_i's (and y_i's) are **associated with** f_b (or with b). Finally, the sender applies a random re-ordering (i.e., permutation) $\phi : [16k] \to [16k]$ to $y_1, \ldots, y_{16k}$ and send the resulting vector, $y_{\phi(1)}, \ldots, y_{\phi(16k)}$, to the receiver.

Decryption: upon receiving the ciphertext $y_1, \ldots, y_{16k}$, when having private key f_r^{-1} (where $r \in \{a, b\}$), the receiver computes $B(f_r^{-1}(y_1)), \ldots, B(f_r^{-1}(y_{16k}))$, and outputs the majority value among these bits.

Correctness of decryption. Let us first state a simple technical claim.

Claim 3.6 *For all but a negligible fraction of the α's and all but a negligible fraction of permutation pairs f_a and f_b over D_α,*

$$|\mathrm{Prob}(B(f_b^{-1}(f_a(x))) = B(x)) - \frac{1}{2}| \quad \textit{is negligible} \tag{3.2}$$

where the probability is taken uniformly over the choices of $x \in D_\alpha$.

Proof: Assume for contradiction that the claim does not hold. Then, without loss of generality, there exists a positive polynomial p so that for infinitely many n's, we have

$$\mathrm{Prob}\left(|\{y \in D_\alpha : B(f_b^{-1}(y)) = B(f_a^{-1}(y))\}| > (\frac{1}{2} + \frac{1}{p(n)}) \cdot |D_\alpha|\right) > \frac{1}{p(n)}$$

when f_a and f_b are independently generated from $\alpha = G_1(1^n)$. This means that for these (α, a, b)'s $B(f_a^{-1}(y))$ gives a non-trivial prediction for $B(f_b^{-1}(y))$. Intuitively this cannot be the case and indeed this lead to contradiction as follows.

Given $a = (\alpha, \beta) \in P$ and $y \in D_\alpha$ we may predict $B(f_a^{-1}(y))$ as follows. First we randomly generate a new permutation. f_b, over D_α, together with its trapdoor. Next we test to see if indeed $B(f_a^{-1}(z))$ is correlated with $B(f_b^{-1}(z))$. (The testing is done by uniformly selecting polynomially many x_i's in D_α, computing $z_i = f_a(x_i)$, and comparing $B(f_a^{-1}(z_i)) = B(x_i)$ with $B(f_b^{-1}(z_i))$.) If a non-negligible correlation is detected then we output $B(f_b^{-1}(y)$ (as our prediction for $B(f_a^{-1}(y)))$. Otherwise we output a uniformly selected bit. (Note that $|\mathrm{Prob}(B(x) = 1) - \frac{1}{2}|$ must be negligible otherwise a constant function contradicts the hard-core hypothesis.) □

From this point on, we assume that the pair (f_a, f_b) satisfies Eq. (3.2).

Lemma 3.7 *Let $\vec{y} = y_1, \ldots, y_{16k}$ be a random encryption of a bit σ. Then with probability $1 - 2^{-\Omega(k)}$ the bit decrypted from $\vec{y}$ is σ.*

Proof: Assume without loss of generality that the private key is f_a^{-1}. Then, the receiver outputs the majority value of the bits $B(f_a^{-1}(y_1)), \ldots, B(f_a^{-1}(y_{16k}))$. Recall that $8k$ of the y_i's are associated with f_a. Out of them, $5k$ (of the y_i's) satisfy $B(f_a^{-1}(y_i)) = B(x_i) = \sigma$, and $3k$ satisfy $B(f_a^{-1}(y_i)) = B(x_i) = 1 - \sigma$. Thus, the receiver outputs $1 - \sigma$ only if at least $5k$ out of the rest of the y_i's (that is, the y_i's associated with f_b) satisfy $B(f_a^{-1}(y_i)) = 1 - \sigma$. However, Eq. (3.2) implies that $|\mathrm{Prob}(B(f_a^{-1}(y_i) = \sigma) - \frac{1}{2}|$ is negligible for each y_i associated with f_b. Thus only an expected $4k$ of the y_i's associated with f_b satisfy $B(f_a^{-1}(y_i)) = 1 - \sigma$. Using a large deviation bound, it follows that decryption errors occur with probability $2^{-\Omega(k)}$. □

Simulation assuming knowledge of both trapdoors. In Lemma 3.9 (below) we show how the simulator, knowing the trapdoors of both f_a and f_b, can generate "dummy ciphertexts" $\vec{z} = z_1, \ldots, z_{16k}$ that can be later "opened" as encryptions of both 0 and 1. Essentially, the values $B(f_a^{-1}(z_i))$ and $B(f_b^{-1}(z_i))$ for each z_i are carefully chosen so that this "cheating" is possible. We use the following notations. Fix an encryption key (f_a, f_b). Let the random variable $\lambda_\sigma = (\sigma; \vec{x}, \phi; \vec{y}; r, f_r^{-1})$ describe a *legal encryption and decryption process* of the bit σ. That is:

- $\vec{x} = x_1, \ldots, x_{16k}$ is a vector of domain elements chosen at random as specified in the encryption algorithm.
- ϕ is a random permutation on $[16k]$.
- $\vec{y} = y_1, \ldots, y_{16k}$ is generated from $\vec{x}$ and ϕ as specified in the encryption algorithm.
- r is uniformly chosen in $\{a, b\}$ and f_r^{-1} is the inverse of f_r. (Note that the decrypted bit is defined by the majority of the bits $B(f_r^{-1}(y_i)$.)

We remark that the information seen by the adversary, after the sender and receiver are corrupted, includes either λ_0 or λ_1 (but not both).

Let us first prove a simple technical claim, that will help us in proving Lemma 3.9. Let BIN_m denote the binomial distribution over $[m]$.

Claim 3.8 *There exists an efficiently samplable distribution μ over $\{0, 1, ..., 4k\}$ so that the distribution $\tilde{\mu}$ constructed by sampling an integer from μ and adding $2k$ is statistically close to BIN_{8k}. That is, the statistical distance between $\tilde{\mu}$ and BIN_{8k} is $2^{-\Omega(k)}$.*

Proof: Let $\text{BIN}_{8k}(i)$ denote the probability of i under BIN_{8k} (i.e., $\text{BIN}_{8k}(i) = \binom{8k}{i} \cdot 2^{-8k}$). We construct the distribution μ (over $\{0, 1, ..., 4k\}$) so that $\text{Prob}(\mu = i) = \text{BIN}_{8k}(i + 2k)$ for $i = 1, ..., 4n$ and $\text{Prob}(\mu = 0)$ equals the remaining mass of BIN_{8k} (i.e., it equals $\sum_{i=0}^{2k} \text{BIN}_{8k}(i) + \sum_{i=6k+1}^{8k} \text{BIN}_{8k}(i)$).

It can be easily seen that each $i \in \{2k+1, ..., 6k\}$ occurs under $\tilde{\mu}$ with exactly the same probability as under BIN_{8k}. Integers i such that $i < 2k$ or $i > 6k$ have probability 0 under $\tilde{\mu}$ (whereas $2k$ is more likely to occur under $\tilde{\mu}$ than under BIN_{8k}). Thus, the statistical distance between $\tilde{\mu}$ and BIN_{8k} equals the probability, under BIN_{8k}, that i is smaller than $2k$ or larger than $6k$. This probability is bounded by $2^{-\Omega(k)}$. □

Lemma 3.9 *Let (f_a, f_b) be the public key, and assume that both f_a^{-1} and f_b^{-1} are known. Then it is possible to efficiently generate $\vec{z}, \vec{x}^{(0)}, \vec{x}^{(1)}, \phi^{(0)}, \phi^{(1)}, r^{(0)}, r^{(1)}$, such that:*

1. $(0; \vec{x}^{(0)}, \phi^{(0)}; \vec{z}; r^{(0)}, f_{r^{(0)}}^{-1}) \stackrel{c}{\approx} \lambda_0$.
2. $(1; \vec{x}^{(1)}, \phi^{(1)}; \vec{z}; r^{(1)}, f_{r^{(1)}}^{-1}) \stackrel{c}{\approx} \lambda_1$.

Here $\stackrel{c}{\approx}$ stands for 'computationally indistinguishable'. We stress that the *same* dummy ciphertext, $\vec{z}$, appears in both (1) and (2).

Proof: Before describing how the dummy ciphertext $\vec{z}$ and the rest of the data are constructed, we summarize, in Figure 3-1, the distribution of the hard-core bits, $B(f_a^{-1}(Y_1)), ..., b(F_a^{-1}(y_{16k}))$ and

$B(f_b^{-1}(y_1)), ..., B(f_b^{-1}(y_{16k}))$, with respect to a real encryption $y_{\phi(1)}, \ldots, y_{\phi(16k)}$ of the bit $\sigma = 0$. Here $\widetilde{\text{BIN}}_{8k}$ denotes the distribution of the number of '1's in $B(f_b^{-1}(y_i))$ for

	$I=\{1,...,8k\}$	$I=\{8k+1,...,16k\}$
$\forall i \in I$	$y_i = f_a(x_i)$	$y_i = f_b(x_i)$
$\sum_{i\in I} B(f_a^{-1}(y_i)) =$	$3k$	$\tilde{\text{BIN}}_{8k}$
$\sum_{i\in I} B(f_b^{-1}(y_i)) =$	$\tilde{\text{BIN}}_{8k}$	$3k$

Figure 3-1: The distribution of the $B(f_s^{-1}(y_i))$'s with respect to λ_0, where $s \in \{a,b\}$. (The case of λ_1 is similar, with the exception that 5k is replaced for 3k.)

$i = 1,...,8k$. Eq. (3.2) implies that the statistical difference between BIN_{8k} and $\tilde{\text{BIN}}_{8k}$ is negligible. The distribution of $B(f_a^{-1}(y_i))$ for $i = 8k+1,...,16k$ is similar. Given only λ_0 (or only λ_1), only three-quarters of the $B(f_s^{-1}(y_i))$'s, $i \in [16k]$ and $s \in \{a,b\}$, are known. Specifically, consider $\lambda_\sigma = (\sigma;\vec{x},\phi;\vec{y};r,f_r^{-1})$, and suppose that $r = a$. Then all the $B(f_a^{-1}(y_i))$'s can be computed using f_a^{-1}. In addition, for $i = 8k+1,...,16k$, $B(f_b^{-1}(y_i)) = B(x_i)$ is known too. However, for $i \in [8k]$, $B(f_b^{-1}(y_i)) = B(f_b^{-1}f_a(x_i)))$ is not known and in fact it is (computationally) unpredictable (from λ_σ). A similar analysis holds for $r = b$; in this case the unpredictable bits are $B(f_a^{-1}(y_i)) = B(f_a^{-1}f_b(x_i)))$ for $i = 8k+1,...,16k$.

Initial construction and conditions: Keeping the structure of λ_σ in mind, we construct $\vec{z}$, along with $\vec{x}^{(0)}$, $\vec{x}^{(1)}$, $\phi^{(0)}$, $\phi^{(1)}$, $r^{(0)}$ and $r^{(1)}$, as follows. First, we select uniformly a bijection, ρ, of $\{0,1\}$ to $\{a,b\}$ (i.e., either $\rho(0) = a$ and $\rho(1) = b$ or the other way around) and set $r^{(0)} = \rho(0)$ and $r^{(1)} = \rho(1)$. Next, we choose, in the way described below, two binary vectors $\vec{\gamma}^{(0)} = \gamma_1^{(0)},\ldots,\gamma_{16k}^{(0)}$ and $\vec{\gamma}^{(1)} = \gamma_1^{(1)},\ldots,\gamma_{16k}^{(1)}$. We choose random values $v_1,\ldots,v_{16k}$ such that $\gamma_i^{(0)} = B(f_{\rho(0)}^{-1}(v_i))$ and $\gamma_i^{(1)} = B(f_{\rho(1)}^{-1}(v_i))$, for each $i \in [16k]$. We uniformly select a permutation ψ over $[16k]$ and let the permuted vector $v_{\psi(1)},\ldots,v_{\psi(16k)}$ be the dummy ciphertext $\vec{z} = (z_1,...,z_{16k})$. It remains to determine $\phi^{(0)}$ and $\phi^{(1)}$, which in turn determine $\vec{x}^{(0)}$ and $\vec{x}^{(1)}$ so that $x_i^{(\sigma)} = f_a^{-1}(z_{(\phi^{(\sigma)})^{-1}(i)})$ for $i \in [8k]$ and $x_i^{(\sigma)} = f_b^{-1}(z_{\phi^{(\sigma)}(i)})$ otherwise. This should be done so that both permutations $\phi^{(0)}$ and $\phi^{(1)}$ are uniformly (but not necessarily independently) distributed and so that the known $B(f_s^{-1}(y_i^{(\sigma)}))$'s match the distribution seen in a legitimate encryption of σ. We stress that $(\sigma;\vec{x}^{(\sigma)},\phi^{(\sigma)};\vec{z};r^{(\sigma)},f_{r^{(\sigma)}}^{-1})$ should appear as a valid encryption of σ. In particular, for each $\sigma \in \{0,1\}$ there should exist a permutation $\psi^{(\sigma)}$ $(= (\phi^{(\sigma)})^{-1} \circ \phi)$ over $[16k]$ so that[8]

1. $\gamma_{\psi^{(\sigma)}(i)}^{(\rho^{-1}(a))} = B(f_a^{-1}(v_{\psi^{(\sigma)}(i)})) = B(f_a^{-1}(z_{\phi^{(\sigma)}(i)})) = B(x_i^{(\sigma)}) = \sigma$, for $i = 1,...,5k$.
   (E.g., if $\rho(0) = a$ this means $\gamma_{\psi^{(\sigma)}(i)}^{(0)} = \sigma$.)

2. $\gamma_{\psi^{(\sigma)}(i)}^{(\rho^{-1}(a))} = B(f_a^{-1}(v_{\psi^{(\sigma)}(i)})) = B(f_a^{-1}(z_{\phi^{(\sigma)}(i)})) = B(x_i^{(\sigma)}) = 1-\sigma$, for $i = 5k+1,...,8k$.
   (E.g., if $\rho(0) = a$ this means $\gamma_{\psi^{(\sigma)}(i)}^{(0)} = 1-\sigma$.)

[8] In each of the following five conditions, the first equality is by the construction of the v_i's, the second equality is by the definition of the z_i's, and the third equality represents the relation between $\vec{x}^{(\sigma)}$, $\vec{z}$ and $\phi^{(\sigma)}$ that holds in a valid encryption (of σ). In conditions (1) through (4), the last equality represents the relation between $\vec{x}^{(\sigma)}$ and σ that holds in a valid encryption of σ. In condition (5), the last equality represents the information computable from $\vec{z}$ using (the trapdoor) $f_{r^{(\sigma)}}^{-1}$. Here we refer to the inverses of the z_i's which are not $x_i^{(\sigma)}$'s. The hard-core value of these inverses should be uniformly distributed.

3. $\gamma^{(\rho^{-1}(b))}_{\psi^{(\sigma)}(i)} = B(f_b^{-1}(v_{\psi^{(\sigma)}(i)})) = B(f_b^{-1}(z_{\phi^{(\sigma)}(i)})) = B(x_i^{(\sigma)}) = \sigma$, for $i = 8k+1, ..., 13k$. (E.g., if $\rho(0) = a$ this means $\gamma^{(1)}_{\psi^{(\sigma)}(i)} = \sigma$.)

4. $\gamma^{(\rho^{-1}(b))}_{\psi^{(\sigma)}(i)} = B(f_b^{-1}(v_{\psi^{(\sigma)}(i)})) = B(f_b^{-1}(z_{\phi^{(\sigma)}(i)})) = B(x_i^{(\sigma)}) = 1-\sigma$, for $i = 13k+1, ..., 16k$. (E.g., if $\rho(0) = a$ this means $\gamma^{(1)}_{\psi^{(\sigma)}(i)} = 1-\sigma$.)

5. Let $I = [8k]$ if $\rho(\sigma) = b$ and $I = \{8k+1, ..., 16k\}$ otherwise. Then, $\gamma^{(\sigma)}_{\psi^{(\sigma)}(i)} = B(f_{\rho(\sigma)}^{-1}(v_{\psi^{(\sigma)}(i)})) = B(f_{\rho(\sigma)}^{-1}(z_{\phi^{(\sigma)}(i)})) = B(f_{\rho(\sigma)}^{-1}(f_{\rho(1-\sigma)}(x_i^{(\sigma)})))$ equals σ with probability negligibly close to $\frac{1}{2}$, for $i \in I$.
(E.g., for $\rho(0) = a$ and $\sigma = 0$ we have $\text{Prob}(\gamma^{(0)}_{\psi^{(\sigma)}(i)} = 1) \approx \frac{1}{2}$ for $i = 8k+1, ..., 16k$, whereas for $\rho(0) = a$ and $\sigma = 1$ we have $\text{Prob}(\gamma^{(1)}_{\psi^{(\sigma)}(i)} = 1) \approx \frac{1}{2}$ for $i = 1, ..., 8k$.)

This allows setting $\phi^{(\sigma)} = \psi \circ (\psi^{(\sigma)})^{-1}$ so that $x^{(\sigma)}_{\phi^{(\sigma)}(i)}$ is "mapped" to z_i while $\phi^{(\sigma)}$ is uniformly distributed (i.e., $x_i^{(\sigma)} = f_a^{-1}(v_{\psi^{(\sigma)}(i)}) = f_a^{-1}(z_{\psi^{-1}(\phi^{(\sigma)}(i))}) = f_a^{-1}(z_{(\phi^{(\sigma)})^{-1}(i)})$ for $i \in [8k]$ and $x_i^{(\sigma)} = f_b^{-1}(z_{\phi^{(\sigma)}(i)})$ otherwise).
Initial setting of $\vec{\gamma}^{(0)}$, $\vec{\gamma}^{(1)}$, $\psi^{(0)}$ and $\psi^{(1)}$: The key issue is how to select $\vec{\gamma}^{(0)}$ and $\vec{\gamma}^{(1)}$ so that the five condition stated above hold (for both $\sigma = 0$ and $\sigma = 1$). As a first step towards this goal we consider the four sums

$$S_1^\sigma \stackrel{\text{def}}{=} \sum_{i=1}^{8k} \gamma^{(\rho^{-1}(a))}_{\psi^{(\sigma)}(i)} \quad , \quad S_2^\sigma \stackrel{\text{def}}{=} \sum_{i=8k+1}^{16k} \gamma^{(\rho^{-1}(b))}_{\psi^{(\sigma)}(i)} \quad , \quad S_3^\sigma \stackrel{\text{def}}{=} \sum_{i=1}^{8k} \gamma^{(\rho^{-1}(b))}_{\psi^{(\sigma)}(i)} \quad , \quad S_4^\sigma \stackrel{\text{def}}{=} \sum_{i=8k+1}^{16k} \gamma^{(\rho^{-1}(a))}_{\psi^{(\sigma)}(i)}$$

The above conditions imply $S_1^\sigma = S_2^\sigma = 5k\cdot\sigma + 3k\cdot(1-\sigma) = 3k + 2k\sigma$ as well as $S_3^\sigma \stackrel{\text{d}}{=} \tilde{\text{BIN}}_{8k}$ if $\rho(\sigma) = b$ and $S_4^\sigma \stackrel{\text{d}}{=} \tilde{\text{BIN}}_{8k}$ otherwise. (Note that S_3^σ, S_4^σ and $\tilde{\text{BIN}}_{8k}$ are random variables.)

To satisfy the above summation conditions we partition $[16k]$ into 4 equal sized subsets denoted I_1, I_2, I_3, I_4 (e.g., $I_1 = [4k]$, $I_2 = \{4k+1, ..., 8k\}$, $I_3 = \{8k+1, ..., 12k\}$ and $I_4 = \{12k+1, ..., 16k\}$). This partition induces a similar partition on the $\gamma_i^{(0)}$'s and the $\gamma_i^{(1)}$'s. The $\gamma_i^{(0)}$'s and the $\gamma_i^{(1)}$'s in each set are chosen using four different distributions which satisfy the conditions summarized in Figure 3-2. Suppose $\rho(0) = a$. Then, we may set

	$I = I_1$	$I = I_2$	$I = I_3$	$I = I_4$
$\sum_{i\in I}\gamma_i^{(0)} \stackrel{\text{d}}{=}$	$3k$	0	$2k$	μ
$\sum_{i\in I}\gamma_i^{(1)} \stackrel{\text{d}}{=}$	μ	$4k$	$2k$	k

Figure 3-2: The distribution of the $\gamma^{(0)}$'s and $\gamma^{(1)}$'s. (μ is as in Claim 3.8.)

$\psi^{(0)}([8k]) = I_1 \cup I_2$ and $\psi^{(0)}(\{8k+1, ..., 16k\}) = I_3 \cup I_4$, and $\psi^{(1)}([8k]) = I_1 \cup I_3$ and $\psi^{(1)}(\{8k+1, ..., 16k\}) = I_2 \cup I_4$, where $\pi(I) = J$ means that the permutation π maps the elements of the set I onto the set J. (It would have been more natural but less convenient to write $(\psi^{(1)})^{-1}(I_1 \cup I_3) = [8k]$ and $(\psi^{(1)})^{-1}(I_2 \cup I_4) = \{8k+1, 16k\}$.) We claim that, for each $\sigma \in \{0,1\}$, the above setting satisfies the three relevant summation conditions. Consider, for example, the case $\sigma = 0$ (depicted in Figure 3-3). Then, $S_1^0 = \sum_{i=1}^{8k}\gamma_i^{(0)} = 3k$ and $S_2^0 = \sum_{i=8k+1}^{16k}\gamma_i^{(1)} = 3k$ as required. Considering $S_4^0 = \sum_{i=8k+1}^{16k}\gamma_i^{(0)}$ we observe that it is distributed as $2k + \mu = \tilde{\mu}$ (of Claim 3.8) which in turn is statistically close to $\tilde{\text{BIN}}_{8k}$. We

		$I = \{1,...,8k\} = (\psi^{(0)})^{-1}(I_1 \cup I_2)$	$I = \{8k+1,...,16k\} = (\psi^{(0)})^{-1}(I_3 \cup I_4)$
$\sum_{i \in I} \gamma_i^{(0)}$	$=$	$S_1^0 = 3k + 0 = 3k$	$S_4^0 = 2k + \mu \stackrel{\mathrm{d}}{=} \tilde{\text{BIN}}_{8k}$
$\sum_{i \in I} \gamma_i^{(1)}$	$=$	no condition	$S_2^0 = 2k + k = 3k$

Figure 3-3: Using $\psi^{(0)}$ the $\gamma_i^{(0)}$'s and $\gamma_i^{(1)}$'s satisfy the summation conditions S_1^0, S_2^0 and S_4^0.

stress that the above argument holds for any way of setting the $\psi^{(\cdot)}$'s as long as they obey the equalities specified (e.g., for any bijection $\pi : I_1 \cup I_2 \stackrel{1\text{-}1}{\mapsto} I_1 \cup I_3$, we are allowed to set $\psi^{(1)}(i) = \pi(i)$ for all $i \in I_1 \cup I_2$). The case $\sigma = 1$ follows similarly; here $S_1^1 = \sum_{i \in I_1 \cup I_3} \gamma_i^{(0)} = 5k$, $S_2^1 = \sum_{i \in I_2 \cup I_4} \gamma_i^{(1)} = 5k$ and $S_3^1 = \sum_{i \in I_1 \cup I_3} \gamma_i^{(1)} = \mu + 2k$ (see Figure 3-4). In case

		$I = \{1,...,8k\} = (\psi^{(1)})^{-1}(I_1 \cup I_3)$	$I = \{8k+1,...,16k\} = (\psi^{(1)})^{-1}(I_2 \cup I_4)$
$\sum_{i \in I} \gamma_i^{(0)}$	$=$	$S_1^1 = 3k + 2k = 5k$	no condition
$\sum_{i \in I} \gamma_i^{(1)}$	$=$	$S_3^1 = \mu + 2k \stackrel{\mathrm{d}}{=} \tilde{\text{BIN}}_{8k}$	$S_2^1 = 4k + k = 5k$

Figure 3-4: Using $\psi^{(1)}$ the $\gamma_i^{(0)}$'s and $\gamma_i^{(1)}$'s satisfy the summation conditions S_1^1, S_2^1 and S_3^1.

$\rho(0) = b$ we set $\psi^{(0)}([8k]) = I_3 \cup I_4$, $\psi^{(0)}(\{8k+1,...,16k\}) = I_1 \cup I_2$, $\psi^{(1)}([8k]) = I_2 \cup I_4$ and $\psi^{(1)}(\{8k+1,...,16k\}) = I_1 \cup I_3$. The claim that, for each $\sigma \in \{0,1\}$, the above setting satisfies the three relevant summation conditions, is shown analogously.

Refinement of $\vec{\gamma}^{(0)}$, $\vec{\gamma}^{(1)}$, $\psi^{(0)}$ and $\psi^{(1)}$: However, the above summation conditions do not guarantee satisfaction of all the five conditions. In particular, we must use permutations $\psi^{(\cdot)}$ which guarantee the correct positioning visible bits within the $8k$-bit long block. That is, we must have

$$
\begin{aligned}
(\gamma^{(\rho^{-1}(a))}_{\psi^{(\sigma)}(1)}, ..., \gamma^{(\rho^{-1}(a))}_{\psi^{(\sigma)}(8k)}) &= (\sigma^{5k}, (1-\sigma)^{3k}) \\
(\gamma^{(\rho^{-1}(a))}_{\psi^{(\sigma)}(8k+1)}, ..., \gamma^{(\rho^{-1}(a))}_{\psi^{(\sigma)}(16k)}) &= (\sigma^{5k}, (1-\sigma)^{3k})
\end{aligned}
$$

that is, equality between the sequences and not merely equality in the number of 1's. Clearly there is no problem to set the $\psi^{(\cdot)}$'s so that these equalities hold and thus Conditions (1) through (4) are satisfied. It is left to satisfy Condition (5).

Suppose that $\rho(\sigma) = a$. In this case the third summation requirement guarantees $\sum_{i=8k+1}^{16k} \gamma^{(\sigma)}_{\psi^{(\sigma)}(i)} \stackrel{\mathrm{d}}{=} \tilde{\text{BIN}}_{8k}$. This is indeed consistent with the requirement that these $\gamma^{(\sigma)}_{\psi^{(\sigma)}(i)}$'s are almost uniformly and independently distributed. But this is not sufficient. In particular, we also need $\sum_{i \in J} \gamma^{(\sigma)}_{\psi^{(\sigma)}(i)} \stackrel{\mathrm{d}}{=} \tilde{\text{BIN}}_{3k}$, where $J = \{8k < i \leq 16k : \gamma^{(1-\sigma)}_{\psi^{(\sigma)}(i)} = 1 - \sigma\}$ and furthermore the above sum needs to be independent of $\sum_{i \in \{8k+1,...,16k\} - J} \gamma^{(\sigma)}_{\psi^{(\sigma)}(i)}$ (which in turn should be statistically close to BIN_{5k}). Let us start with the case $\sigma = 0$. In this case we need

$$\sum_{i \in J} \gamma_i^{(0)} \stackrel{\mathrm{d}}{=} \tilde{\text{BIN}}_{3k}, \tag{3.3}$$

where $J = \{i \in I_3 \cup I_4 : \gamma_i^{(1)} = 1\}$, and this sum needs to be independent of $\sum_{i \in I_3 \cup I_4 - J} \gamma_i^{(0)}$. By Figure 3-2 we have $|J \cap I_3| = 2k$. We further restrict the distributions $\gamma_i^{(0)}$'s and $\gamma_i^{(1)}$'s so that in part I_3 the four possible outcomes of the pairs $(\gamma_i^{(0)}, \gamma_i^{(1)})$ are equally likely (e.g., for exactly k integers $i \in I_3$ we have $(\gamma_i^{(0)}, \gamma_i^{(1)}) = (0,0)$). Consider $J' = J \cap I_4$ (note $|J'| = k$). To satisfy Eq. (3.3) we construct a random variable $\mu' \in \{0, 1, ..., k\}$ (analogously to Claim 3.8) so that $p_j \stackrel{\rm def}{=} \text{Prob}(\mu' = j) = \text{BIN}_{3k}(k+j)$ for $j \in [k]$ (with the rest of the mass on $\mu' = 0$) and constrain the $\gamma_i^{(0)}$'s to satisfy $\text{Prob}(\sum_{i \in J'} \gamma_i^{(0)} = j) = p_j$. We get $\sum_{i \in J} \gamma_i^{(0)} = k + \mu' \stackrel{\rm d}{=} \tilde{\text{BIN}}_{3k}$ (analogously to Claim 3.8). A minor problem occurs: the new restriction on the $\gamma_i^{(0)}$'s conditions $\sum_{i \in I_4 - J'} \gamma_i^{(0)}$ which we want to be distributed as some $\mu'' \stackrel{\rm d}{=} \text{BIN}_{5k} - 2k$ and independently of μ' (the reason being that $\mu' + \mu''$ should be distributed equally to μ). However this condition has a negligible effect since we can sample μ' and μ and set the $\gamma_i^{(0)}$'s accordingly, getting into trouble only in case $\mu < \mu'$ which happens with negligible probability (since $\text{Prob}(\mu < \mu') < \text{Prob}(\mu < k) = 2^{-\Omega(k)}$).

The case $\sigma = 1$ gives rise to the requirement

$$\sum_{i \in J} \gamma_i^{(1)} \stackrel{\rm d}{=} \tilde{\text{BIN}}_{3k}, \tag{3.4}$$

where $J = \{i \in I_1 \cup I_3 : \gamma_i^{(0)} = 0\}$, and this sum needs to be independent of $\sum_{i \in I_1 \cup I_3 - J} \gamma_i^{(1)}$. To satisfy Eq. (3.4) we restrict the $\gamma_i^{(1)}$'s in $J' \stackrel{\rm def}{=} J \cap I_1$ analogously to satisfy $\sum_{i \in J'} \gamma_i^{(1)} = \mu'$. Finally, we observe that generating the $\gamma_i^{(0)}$'s and $\gamma_i^{(1)}$'s at random so that they satisfy the above requirements makes them satisfy Condition (5).

Beyond the five conditions. In the above construction we have explicitly dealt with conditions which obviously have to hold for the construction to be valid. We now show that indeed this suffices. Namely, we claim that

$$(\sigma; \vec{x}^{(\sigma)}, \phi^{(\sigma)}; \vec{z}; r^{(\sigma)}, f^{-1}_{r^{(\sigma)}}) \stackrel{\rm c}{\approx} \lambda_\sigma = (\sigma; \vec{x}, \phi; \vec{y}; r, f_r^{-1}). \tag{3.5}$$

Consider the case $\sigma = 0$. Both $r^{(0)}$ and r are uniformly chosen in $\{a, b\}$ and so we consider, w.l.o.g., $r = r^{(0)} = a$. Furthermore, $\phi^{(0)}$ is a random permutation and $f_a(x_i^{(0)}) = z_{\phi^{(0)}}$ for $i = 1, ..., 8k$, and $f_b(x_i^{(0)}) = z_{\phi^{(0)}}$ for $i = 8k+1, ..., 16k$, which matches the situation w.r.t ϕ, $\vec{x}$ and $\vec{y}$. It remains to compare the distributions of $B(f_s^{-1}(\cdot))$'s, $s \in \{a, b\}$, with respect to $\vec{x}^{(0)}$ and with respect to $\vec{x}$. By the above analysis we know that the entries corresponding to $s = a$ and to $(s = b) \wedge (i \leq 8k)$ are distributed similarly in the two cases. Thus, we need to compare $B(f_b^{-1}(f_a(x_1^{(0)}))), ..., B(f_b^{-1}(f_a(x_{8k}^{(0)})))$ and $B(f_b^{-1}(f_a(x_1))), ..., B(f_b^{-1}(f_a(x_{8k})))$. Recall that the x_i's are selected at random subject to $B(x_i) = 0$ for $i = 1, ..., 5k$ and $B(x_i) = 1$ for $i = 5k+1, ..., 8k$. An analogous condition is imposed on the $x_i^{(0)}$'s but in addition we also have $B(f_b^{-1}(f_a(x_i^{(0)}))) = 1$ for $i = 1, ..., 4k$, and some complicated conditions on $B(f_b^{-1}(f_a(x_i^{(0)}))) = 1$, for $i = 4k+1, ..., 8k$ (i.e., the distribution of 1's here is governed by μ and furthermore in the first k elements the number of 1's is distributed identically to μ'). Thus, distinguishing $\vec{x}$ from $\vec{x}^{(0)}$ amounts to distinguishing, given $f_a, f_b : D \mapsto D$ and the trapdoor for f_a (but not for f_b), between the two distributions

1. $(u_1, ..., u_{8k})$, where the u_i's are independently selected so that $B(u_i) = 0$ if $i \in [5k]$ and $B(u_i) = 1$ otherwise; and

2. $(w_1, ..., w_{8k})$, where the w_i's are uniformly selected under the conditions

   - $B(w_i) = 0$ if $i \in [5k]$ and $B(u_i) = 1$ otherwise,

- $B(f_b^{-1}(f_a(w_i))) = 1$ for $i \in [4k]$,
- $\sum_{i=4k+1}^{5k} B(f_b^{-1}(f_a(w_i))) = \mu'$, and
- $\sum_{i=5k+1}^{8k} B(f_b^{-1}(f_a(w_i))) = \mu''$, for some $\mu'' \stackrel{\mathrm{d}}{=} \mu - \mu'$.

We claim that distinguishing these two distributions yields a contradiction to the security of the hard-core predicate B. Suppose, on the contrary that an efficient algorithm A can distinguish these two distributions. Using a hybrid argument we construct an algorithm A' which distinguishes the the uniform distribution over $D' \stackrel{\text{def}}{=} \{x \in D : B(x) = \tau\}$ and a distribution over D' that is uniform over both $D'_0 \stackrel{\text{def}}{=} \{x \in D' : B(f_b^{-1}(f_a(x))) = 0\}$ and $D'_1 \stackrel{\text{def}}{=} \{x \in D' : B(f_b^{-1}(f_a(x))) = 1\}$, where τ is a bit which can be efficiently determined. (We stress that the latter distribution is not uniform on D' but rather uniform on each of its two parts.) Without loss of generality, we assume $\tau = 0$. It follows that A' must distinguish inputs uniformly distributed in D'_0 from inputs uniformly distributed in D'_1. We now transform A' into an algorithm, A'', that distinguishes a uniform distribution over $\{y \in D : B(f_b^{-1}(y)) = 0\}$ from a uniform distribution over $\{y \in D : B(f_b^{-1}(y)) = 1\}$. On input $y \in D_\alpha$ and $f_b : D \mapsto D$, algorithm A'' first generates another permutation f_a, over D, together with the trapdoor for f_a. Next, it computes $x = f_a^{-1}(y)$ and stop (outputting 0) if $B(x) = 1$ (i.e., $x \notin D'$). Otherwise, A'', invokes A' on x and outputs $A'(x)$. In this case $B(f_b^{-1}(f_a(x))) = B(f_b^{-1}(y))$ (and $B(x) = 0$) so the output will be significantly different in case $B(f_b^{-1}(y))) = 0$ and in case $B(f_b^{-1}(y))) = 1$. We observe that $\mathrm{Prob}(B(x) = 0) \approx \frac{1}{2}$ (otherwise a constant function violates the security of B), and conclude that one can a random y with $B(f_b^{-1}(y)) = 0$ from a random y with $B(f_b^{-1}(y)) = 1$ (which contradicts the security of B). □

Key generation

We describe how the keys are generated, based on any common-domain trapdoor system. We use Oblivious Transfer [MRa1, EGL] in our constructions. Oblivious Transfer (OT) is a protocol executed by a sender S with inputs s_1 and s_2, and by a receiver R with input $\tau \in \{1, 2\}$. After executing an OT protocol, the receiver should know s_τ, and learn nothing else. The sender S should learn nothing from participating in the protocol. In particular S should not know whether R learns s_1 or s_2. We are only concerned with the case where R is uncorrupted and non-erasing.

We use the implementation of OT described in [GMW] (which in turn originates in [EGL]). This implementation has an additional property, discussed below, that is useful in our construction. For self containment we sketch, in Figure 3-5, the [GMW] protocol for OT of one bit.

It can be easily verified that the receiver outputs the correct value of σ_τ in Step 4. Also, if the receiver is semi-honest in the non-erasing sense, then it cannot predict $\sigma_{3-\tau}$ with more than negligible advantage over $\frac{1}{2}$. [9] The sender view of the interaction is uncorrelated with the value of $\tau \in \{1, 2\}$. Thus it learns nothing from participating in the protocol.

The important additional property of this protocol is that, in a simulated execution of the protocol, the simulator can learn both σ_1 and σ_2 by uniformly selecting $z_1, z_2 \in D$, and

[9] This statement does not hold if R is semi-honest only in the honest-looking sense. Ironically, this 'flaw' is related to the useful (non-committing) feature discussed below.

Oblivious Transfer (OT)

The parties proceed as follows, using a trapdoor-permutations generator and the associated hard-core predicate $B()$.

1. On input $\sigma_1, \sigma_2 \in \{0,1\}$, the sender generates a one-way trapdoor permutation $f : D \to D$ with its trapdoor f^{-1}, and sends f to the receiver.
2. On input $\tau \in \{1,2\}$, the receiver uniformly selects $x_1, x_2 \in D$, computes $y_\tau = f(x_\tau)$, sets $y_{3-\tau} = x_{3-\tau}$, and sends (y_1, y_2) to the sender.
3. Upon receiving (y_1, y_2), the sender sends the pair $(\sigma_1 \oplus B(f^{-1}(y_1)), \sigma_2 \oplus B(f^{-1}(y_2)))$ to the receiver.
4. Having received (b_1, b_2), the receiver outputs $s_\tau = b_\tau \oplus B(x_\tau)$ (as the message received).

Figure 3-5: The [GMW] Oblivious Transfer protocol

having the receiver R send $f(z_1), f(z_2)$ (in Step 2). Furthermore, if R is later corrupted, then the simulator can "convince" the adversary that R received either σ_1 or σ_2, at wish, by claiming that in Step 2 party R chose either $(x_1, x_2) = (z_1, f(z_2))$ or $(x_1, x_2) = (f(z_1), z_2)$, respectively.

In Figure 3-6 we describe our key generation protocol. This protocol is valid as long as at least one party remains uncorrupted.

Simulation (Adaptive security of the encryption protocol)

Let ε denote the combined encryption and decryption protocols, preceded by the key generation protocol.

Theorem 3.10 *Protocol ε is an $(n-1)$-resilient non-committing encryption protocol for n parties, in the presence of non-erasing parties.*

Proof (sketch): Let P_r be the sender and let P_s be the receiver. Recall that a non-committing encryption protocol is a protocol that securely computes the bit transmission function, $\text{BTR}_{s,r}$, in a simulatable way. Let ε' be a non-erasing protocol for ε. We construct a simulator $\mathcal{S}$ such that $\text{IDEAL}_{\text{BTR}_{s,r},\mathcal{S}^{\mathcal{A}}}(\sigma) \stackrel{\text{d}}{=} \text{EXEC}_{\varepsilon',\mathcal{A}}(\sigma)$, for any $(n-1)$-limited adversary $\mathcal{A}$ and for any input $\sigma \in \{0,1\}$ of P_s.

The simulator $\mathcal{S}$ proceeds as follows. First an invocation of the key generation protocol ε_{G} is simulated, in such a way that $\mathcal{S}$ knows both trapdoors f_a^{-1} and f_b^{-1}. (This can be done using the additional property of the [GMW] Oblivious Transfer protocol, as described above.) For each party P that $\mathcal{A}$ corrupts during this stage, $\mathcal{S}$ hands $\mathcal{A}$ the internal data held by P in the simulated interaction. We stress that as long as at least one party remains uncorrupted, the adversary knows at most *one* of f_a^{-1}, f_b^{-1}. Furthermore, as long as P_r remains uncorrupted, the adversary view of the computation is independent of whether P_r has f_a^{-1} or f_b^{-1}.

Once the simulation of the key generation protocol is completed, $\mathcal{S}$ instructs the trusted party in the ideal model to notify P_r of the function value. (This value is P_s's input, σ.) If

key-generation (ε_{G})

For generating an encryption key (f_a, f_b) known to the sender, and a decryption key f_r^{-1} known only to the receiver (R), where r is uniformly distributed in $\{a, b\}$.

1. The receiver generates a common domain D_α and sends α to all parties.
2. Each party P_i generates two trapdoor permutations over D_α, denoted f_{a_i} and f_{b_i}, and sends (f_{a_i}, f_{b_i}) to R. The trapdoors of f_{a_i} and f_{b_i} are kept secret by P_i.
3. The receiver R chooses uniformly $\tau \in \{1, 2\}$ and invokes the OT protocol with each party P_i for a number of times equal to the length of the description of the trapdoor of a permutation over α. In all invocations the receiver uses input τ. In the j^{th} invocation of OT, party P_i acting as sender uses input (σ_1, σ_2), where σ_1 (resp., σ_2) is the j^{th} bit of the trapdoor of f_{a_i} (resp., f_{b_i}). (Here we use the convention by which, without loss of generality, the trapdoor may contain all random choices made by G_2 when generating the permutation. This allows R to verify the validity of the data received from P_i.)
4. Let H be the set of parties with which all the OT's were completed successfully. Let f_a be the composition of the permutations f_{a_i}'s for $P_i \in H$, in some canonical order, and let f_b be defined analogously (e.g., a is the concatenation of the a_i with $i \in H$). Let $r = a$ if $\tau = 1$ and $r = b$ otherwise. The trapdoor to f_r is known only to R (it is the concatenation of the trapdoors obtained in Step 3).
5. R now sends the public key (f_a, f_b) to the sender.

Figure 3-6: The key generation protocol

at this point either P_s or P_r is corrupted, then $\mathcal{S}$ gets to know the encrypted bit. In this case $\mathcal{S}$ generates a true encryption of the bit σ, according to the protocol. If neither P_s nor P_r are corrupted, then $\mathcal{S}$ generates the values $\vec{z}, \vec{x}^{(0)}, \vec{x}^{(1)}\phi^{(0)}, \phi^{(1)}, r^{(0)}, r^{(1)}$ as in Lemma 3.9, and lets $\vec{z}$ be the ciphertext that P_s sends to P_r in the simulated interaction.

If at this stage $\mathcal{A}$ corrupts some party P which is not the sender or the receiver, then $\mathcal{S}$ hands $\mathcal{A}$ the internal data held by P in the simulated interaction. If $\mathcal{A}$ corrupts P_s, then $\mathcal{S}$ corrupts P_s in the ideal model and learns σ. Next $\mathcal{S}$ hands $\mathcal{A}$ the values $\vec{x}^{(\sigma)}, \phi^{(\sigma)}$ for P_s's internal data. If $\mathcal{A}$ corrupts P_r, then $\mathcal{S}$ corrupts P_r in the ideal model, learns σ, and hands $\mathcal{A}$ the value $f^{-1}_{r^{(\sigma)}}$ for P_s's internal data.

The validity of the simulation follows from Lemma 3.9 and from the properties of the [GMW] Oblivious Transfer protocol. □

3.3.3 Alternative implementations of non-committing encryption

We describe two alternative implementations of our non-committing encryption scheme, based on the RSA and DH assumptions, respectively. These implementations have the advantage that the key generation stage can be simplified to consist of a single message sent from the receiver to the sender.

An implementation based on RSA. We first construct the following common-domain trapdoor system. The common domain, given security parameter n, is $\{0,1\}^n$. A permutation over $\{0,1\}^n$ is chosen as follows. First choose a number N uniformly from $[2^{n-1} \ldots 2^n]$, *together with its factorization* (via Bach's algorithm [Ba]). Next choose a prime $2^n < e < 2^{n+1}$. (This way, we are assured that $gcd(e, \phi(N)) = 1$, where $\phi()$ is Euler's totient function, even if the factorization of N is not known.) Let $f_N(x) = x^e (\mathrm{mod} N)$ if $x < N$ and $f_N(x) = x$ otherwise. With non-negligible probability N is a product of two large primes. Thus, this construction yields a collection of common-domain permutations which are weakly one-way. Employing an amplification procedure (e.g., [Y2, GILVZ]) we obtain a proper common-domain system.

This common-domain trapdoor system can be used as described in Section 3.3.2. However, here the key-generation stage can be simplified considerably. Observe that it is possible to choose a permutation from the above distribution *without knowing its trapdoor.* Specifically, this is done by choosing the numbers N of the different instances of f_N in the direct way, without knowing their factorization. Thus, the receiver will choose two trapdoor permutations f_a, f_b, where only the trapdoor to f_r (i.e., f_r^{-1}) is known, $r \in_{\mathrm{R}} \{a, b\}$. Both f_a, f_b are now sent to the sender, who proceeds as in Section 3.3.2. In a simulated execution the simulator will choose both f_a and f_b together with their trapdoors.[10]

An implementation based on DH. Consider the following construction. Although it fails to satisfy Definition 3.5, it will be 'just as good' for our needs. The common domain, given security parameter n, is a prime $2^{n-1} < p < 2^n$ where the factorization of $p-1$ is known. Also, a generator g of Z_p^* is fixed. p and g are publicly known. All computations are done modulo p. To choose a permutation over Z_p^*, choose an element $v \in_{\mathrm{R}} \mathcal{Z}_{p-1}^*$ and let $f_v(x) = x^v$. The public description of f_v is $y \triangleq g^v$. The 'trapdoor' is $u \triangleq v^{-1}(\mathrm{mod} p-1)$.

[10] A similar idea was used in [DP].

This construction has the following properties:

- Although it is hard to compute f_v if only p, g, y are known, it is easy to generate random elements $x \in_{\mathrm{R}} Z_p^*$ together with $f_v(x)$: choose $z \in_{\mathrm{R}} \mathcal{Z}_p^*$, and set $x = g^z$ and $f_v(x) = y^z$. (This holds since $f_v(x) = x^v = g^{zv} = y^z$.)
- If u is known then it is easy to compute $f_v^{-1}(x) = x^u$.
- An algorithm A that inverts f_v given only p, g, y can be easily transformed into an algorithm A' that given p, g, g^α, g^β outputs $g^{\alpha\beta}$ (that is, into an algorithm that contradicts the Diffie-Hellman (DH) assumption). Specifically, Assume that $A(p, g, g^v, x^v) = x$. Then, on input p, g, g^α, g^β, algorithm A' will run $A(p, g^\alpha, g, g^\beta)$ to obtain $g^{\alpha\beta}$.
- It is possible to choose a permutation from the above distribution *without knowing its trapdoor.* Specifically, this is done by uniformly choosing numbers $y \in_{\mathrm{R}} Z_p^*$ until a generator is found. (It is easy to decide whether a given y is a generator of Z_p^* when the factorization of $p - 1$ is known.)

Note that both in the encryption process and in the simulation it is not necessary to compute the permutations f on arbitrary inputs. It suffices to be able to generate random elements x in the domain together with their function value $f(x)$. Thus, this construction is used in a similar way to the previous one.

A concluding remark to Section 3.3. Our solutions for non-erasing parties may appear somewhat unsatisfactory since they are based on 'trusting' the receiver to choose trapdoor permutations without knowing the trapdoor, whereas the permutation can be chosen together with its trapdoor by simple 'honest-looking' behavior. Recall, however, that if honest-looking parties are allowed then no (non-trivial) protocol can be proven adaptively secure (via black-box simulation if claw-free pairs exist). We do not see a meaningful way to distinguish between the 'honest-looking behavior' that foils the security of our constructions and the 'honest-looking behavior', described in Section 3.1.1, that foils provability of the adaptive security of any protocol.

3.4 Honest-looking parties

Our construction for honest-looking parties assumes the existence of a "trusted dealer" at a pre-computation stage. The dealer chooses, for each party P, a truly random string r_P, and hands r_P to P, to be used as random input. (We call r_P a **certified random input** for P.) Next, the dealer generates $n-1$ shares of r_P, so that r_P can be reconstructed from all $n-1$ shares, but any subset of $n-2$ shares are independent of r_P. Finally the dealer hands one share to each party *other than P.*

Now, all parties are able to jointly reconstruct r_P, and thus verify whether P follows its protocol. Consequently, if party P is honest-looking (i.e., P does not take any chance of being caught cheating), then it is forced to use r_P exactly as instructed in the protocol. Party P is now limited to non-erasing behavior, and the construction of Section 3.3 applies. (We note that the use of certified random inputs does not limit the simulator. That is, upon corruption of party P, the simulator can still compute some convenient value r'_P to be used as P's random input, and then "convince" the adversary that the certified random input of P was r'_P. The adversary will not notice anything wrong since it will never have all the shares of the certified random input.)

CHAPTER 4

Asynchronous secure computation

We study secure multiparty computation in asynchronous networks. (See an introductory presentation in Section 1.4.) We first present, in Section 4.1, our definition of asynchronous secure multiparty computation. We consider the bounded variant of the secure channels setting, in the presence of adaptive adversaries and non-erasing parties. Here we also describe our conventions for presenting asynchronous protocols. Next we present, in Section 4.2, asynchronous primitives used in our protocols. In Sections 4.3 and 4.5 we describe our construction of the cases of Fail-Stop and Byzantine adversaries, respectively. In Section 4.4 we define and construct an Asynchronous Verifiable Secret Sharing (AVSS) scheme, that is a key tool in our construction for Byzantine adversaries. In Section 4.6 we demonstrate the optimality of our constructions.

4.1 Preliminaries

4.1.1 The asynchronous model

Consider an asynchronous network of n parties, where every two parties are connected via a reliable and private communication channel. Messages sent on a channel can be arbitrarily delayed; however, each message sent is eventually received. Furthermore, the order in which messages are received on a channel may be different from the order in which they were sent[1].

It is convenient to regard a computation in our model as a sequence of **steps**. In each step a single party is active. The party is activated by receiving a message; it then performs an internal computation, and possibly sends messages on its outgoing channels. We consider the order of the steps as controlled by an adversarial entity, called a **scheduler**. We allow computationally unbounded schedulers. The privacy of the channels is modelled by considering only **oblivious** schedulers. The only information known to these schedulers is the origin and destination (and, possibly, the length) of each message sent[2]. More formally,

[1]For simplicity, we assume that messages are not duplicated. Duplication can be prevented by standard methods, e.g., using counters

[2]Private channels are useless in the presence of schedulers that have access to the *contents* of the messages:

an **oblivious scheduler** is a function$D : (\mathbf{N} \times [n]^2)^* \rightarrow \mathbf{N}$, with the following interpretation. Given the list of $\{$*length, source, destination*$\}$ of the i first messages sent in an execution, in some standard global ordering, function D specifies the serial number of the next message to be delivered. (The scheduler must deliver every message exactly once, and cannot deliver unsent messages.) In the sequel we incorporate the scheduler within the adversary.

We distinguish two types of adversaries: **Fail-Stop** and **Byzantine**. If the adversary is Fail-stop, then the corrupted parties may stop sending messages at some time during the computation; however, we assume that the corrupted parties continue to receive messages and have an output. If the adversary is Byzantine, then the corrupted parties may deviate from their protocols in any way. We consider adaptive adversaries.

4.1.2 Defining secure asynchronous computation

We use the 'ideal model methodology', described in Section 1.2 and Chapter 2, for defining asynchronous secure computation. The 'asynchronous version' of the ideal model should reflect the special properties of the asynchronous setting. In particular, the following property of the asynchronous setting remains unchanged, even in the presence of a trusted party. In an asynchronous network with t potential corruptions, the uncorrupted parties (as well as the trusted party) cannot wait to communicate with more than $n-t$ parties before deciding on the output of a computation, since up to t parties may never join the computation. Consequently, unlike synchronous computations, the output of an asynchronous computation can be based only on some 'core' subset, of size at least $n-t$, of the inputs. Furthermore, the t inputs that were left out are not necessarily inputs of corrupted parties: they can be inputs of slow, however uncorrupted parties.[3] [4]

Therefore, we suggest the following asynchronous ideal model. Evaluating a function f in this model proceeds in the same way as in the synchronous ideal model (see Section 2.3 on page 21), with the only exception that the computation stage is modified as follows.

Computation stage (asynchronous version): The adversary chooses an arbitrary 'core' set of size at least $n-t$; this subset, denoted C, is independent of the inputs of the uncorrupted parties. A 'scheduler' delivers only the messages of the parties in C to the trusted party.

Upon receiving the (possibly substituted) inputs of the parties in subset C, the trusted party computes some predefined "approximation" to the function value, based on C and the inputs of the parties in C. (For concreteness, we use the following "approximation": set the inputs of the parties in $\bar{C}$ to 0, and compute the original function.) In order for the output to make more sense, the trusted party outputs the subset C as well as the approximated function value.

for instance, such a scheduler can set the delivery order of the messages so that the first bit in the first message received by party P_1 will be the same as the fifth bit in the first message that party P_2 sent on its private channel to P_3... thus 'breaking' the privacy of this channel.

[3] Chor and Moscovici describe this property of the asynchronous model in more detail [CM]. Furthermore, they give an exact characterization of the achievable 'tasks' in the presence of a given number of Fail-Stop corruptions, when the privacy of the inputs is disregarded.

[4] A consequence of this phenomenon is that no asynchronous computation can be "equivalent" to a computation in the *synchronous* trusted party scenario.

Consequently, the output of the parties in this ideal model is as follows. On input $\vec{x} = x_1, \ldots, x_n$ of the parties (input x_i to party P_i), let C be the set described above, and let $\vec{y}$ denote the (possibly modified) inputs of the parties in C. Let f_C denote the "approximation" of the function f, made by the trusted party. Then, the uncorrupted parties output $(C, f_C(\vec{y}))$. The corrupted parties output an arbitrary function of the information gathered by the adversary in the computation; this information consists of the identity and inputs of the corrupted parties, their random inputs, and the resulting approximation $f_C(\vec{y})$. Also here, 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 ideal-model adversary $\mathcal{S}$ (party P_i outputs $\text{IDEAL}_{f,\mathcal{S}}(\vec{x})_i$).

In Definitions 4.1 through 4.3 we present notations for the output of the parties in the asynchronous ideal model. (These notations capture the above description in an analogous way to Definitions 2.8 through 2.10 in the synchronous setting, and can be skipped in a first reading.) We use the same technical notations as in Section 2.3:

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

- 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 $\vec{b}$.

Using these notations, the *approximation of function f based on a subset $C \subseteq [n]$* is defined by $f_C(\vec{x}) = f(\vec{x}/_{(\bar{C},\vec{0})})$.

Definition 4.1 *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 asynchronous ideal-model-adversary *is a quintuple $\mathcal{S} = (t, b, h, c, O)$, 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*

- *$c : A^* \times \mathcal{R} \to \{C \subseteq [n] | \; |C| \geq n - t\}$ is a* core set selection *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 4.2 *Let D be the domain of possible inputs of the parties, and let $\mathcal{S} = (t, b, h, c, O)$ 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* ith set of corrupted parties *in the ideal model, denoted $B^{(i)}(\vec{x}, r)$, 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 $0 \leq i < t$, and as long as $b_i \neq \perp$, 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 $b_i^f \triangleq b(B^{(i)}(\vec{x}, r), \vec{x}_{B^{(i)}(\vec{x},r)}, f_C(\vec{y}), r)$, where $\vec{y}$ is the substituted input vector for the trusted party, and C is the selected core subset. That is, $\vec{y} \triangleq \vec{x}/_{(B^{(i^*)}(\vec{x},r), h(B^{(i^*)}(\vec{x},r), \vec{x}_{B^{(i^*)}(\vec{x},r)}, r))}$, and $C \triangleq c(\vec{x}_{B^{(i^*)}(\vec{x},r)}, r)$.*

  For $i^* \leq i < t$, let
$$B^{(i+1)}(\vec{x}, r) \triangleq B^{(i)}(\vec{x}, r) \cup b_i^f.$$

In Definition 4.3 we use $B^{(i)}$ instead of $B^{(i)}(\vec{x}, r)$.

Definition 4.3 *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 asynchronous ideal model *with adversary $\mathcal{S} = (t, b, h, c, O)$, 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 $1 \leq i \leq n$:*
$$\text{IDEAL}_{f,\mathcal{S}}(\vec{x})_i = \begin{cases} (C, f_C(\vec{y})) & \text{if} \quad i \notin B^{(t)} \\ O(\vec{x}_{B^{(t)}}, f_C(\vec{y}), r) & \text{if} \quad i \in B^{(t)} \end{cases}$$
where r is the random input of $\mathcal{S}$, $B^{(t)}$ is the t^{th} set of corrupted parties, $\vec{y} = \vec{x}/_{(B^{(t)}, h(B^{(t)}, \vec{x}_{B^{(t)}}, r))}$ is the substituted input vector for the trusted party, and $C \triangleq c(\vec{x}_{(B^{(t)}}, r)$ is the selected core subset.

Next we describe the execution of a protocol π in the asynchronous real-life scenario, where the uncorrupted parties run some semi-honest protocol π' for π, and in the presence of a computationally unbounded, adaptive t-limited asynchronous real-life adversary $\mathcal{A}$. (We incorporate the scheduler, described in Section 4.1.1, in $\mathcal{A}$.) The computation proceeds as follows. Initially, and upon the receipt of a message, each party sends messages according to π'. The adversary (playing the scheduler) sees the sender, receiver and length of each message sent. It also sees the contents of the messages sent to corrupted parties. After the sending of each message, the adversary may decide to deliver one of the messages that were sent and not yet delivered. It may also decide to corrupt some parties. Upon corruption of a party P, the adversary sees all the data kept by P according to π'. As in Section 2.3, we assume that the corrupted parties output the adversary view of the comutation (i.e., all the information gathered by the adversary during the computation). We let $\text{VIEW}_{\pi,\mathcal{A}}(\vec{x})$ and $\text{EXEC}_{\pi,\mathcal{A}}(\vec{x})$ have an analogous meaning to that of Section 2.3, with respect to the asynchronous setting.

In Definition 4.4 below we concentrate on the bounded variant of the (asynchronous) secure channels setting. (See Section 2.1 for a presentation of the variants.)

Definition 4.4 *Let $f : D^n \to D'$ for some domains D and D', and let π be a protocol for n parties that runs in* PPT. *We say that π* asynchronously t-securely computes f in the bounded secure channels setting, *if the following requirements hold for any semi-honest protocol π' for π (according to one of the Definitions 2.5 through 2.7), and for any asynchronous t-limited real-life adversary $\mathcal{A}$:*

Termination: *On all inputs, all the uncorrupted parties complete execution of the protocol with probability 1.*

Security: *there exists an asynchronous t-limited ideal-model-adversary $\mathcal{S}$, whose complexity is polynomial in the complexity of $\mathcal{A}$, such that for every input vector $\vec{x}$,*

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

Remarks:

- We remark that the Termination property is implicit in the security property (parties that did not complete the protocols do not have an output). We explicitly require Termination to stress the importance and delicacy of this requirement in an asynchronous setting. In particular, we note that Consensus is a (very limited) special case of secure computation. The [FLP] impossibility result for *deterministic* protocols implies that in an asynchronous network with potential faults there must exist non-terminating runs of any (randomized) protocol reaching Consensus. Thus, we only require a secure protocol to terminate with probability (or measure) 1.
- If only Fail-Stop adversaries are allowed, then the real-life adversary $\mathcal{A}$ only specifies: (a) the conditions upon which the corrupted parties should stop sending messages, and (b) the output of the corrupted parties. Consequently, the input substitution function h of the trusted-party adversary are the identity function.

4.1.3 Writing asynchronous protocols

We review the way in which an asynchronous computation is carried out, and describe some useful writing conventions for asynchronous protocols. In Section 4.1.1 we described an asynchronous computation as a sequence of **steps**. This is a global, 'external' view of the computation. From the point of view of a party, a computation is a sequence of **cycles**. A cycle is initiated upon receiving a message; it consists of executing an internal computation, and possibly sending messages to other parties. Once a cycle is completed, the party waits for the next message. The set of instructions to be executed upon the receipt of a message is called an **asynchronous protocol**. These instructions may depend on the internal state of the party, and on the contents of the message received.

However, for clarity of presentation and analysis, our description of an asynchronous protocol is somewhat different: we partition the protocol into several modules, called **sub-protocols**. Each sub-protocol is first presented, and sometimes analyzed, as if it were the only protocol run by the party. Sub-protocols are combined by letting one sub-protocols 'call' other sub-protocols during its execution. We present our interpretation of this writing style of asynchronous protocols.

The party keeps track of the sub-protocols that are currently 'in execution': initially, only one predefined sub-protocol is 'in execution'; when some sub-protocol is called by another sub-protocol, it is added to the sub-protocols 'in execution'.

We assume that each message applies to one sub-protocol only. Thus, Upon the receipt of a message, the party invokes a cycle of the relevant sub-protocol. If the received message refers to a sub-protocol which is not 'in execution', then the party keeps the message; once the sub-protocol has started, a cycle is invoked for each one of the kept messages.

We associate input and output values with each sub-protocol; furthermore, the output of one sub-protocol may be the input of another sub-protocol. However, the latter sub-protocol may be in execution before the former sub-protocol has completed updating its output; consequently, the input of the latter sub-protocol may be changed during its execution. We call such input a dynamic input; namely, unlike regular input, dynamic input variables may have different input values in different cycles in the same run of a sub-protocol.

The variables we use for dynamic input will only have items added to them. It is convenient to regard such variables as monotonically increasing sets. Namely, let $\mathcal{U}$ be such a variable, and let $\mathcal{U}^{(c)}$ denote the set held in $\mathcal{U}$ when this set is of size c for the first time; then, if $c < d$ we have $\mathcal{U}^{(c)} \subseteq \mathcal{U}^{(d)}$. We call these variables accumulative sets. (In the sequel, we will use calligraphic letters (e.g., $\mathcal{U}$) to denote accumulative sets.)

Using the above conventions, the shorthand "Set $a = \pi(b)$" stands for: "(a) call sub-protocol π, with b for input; (b) Let a denote its output". Note that b may be a dynamic input of π; similarly, the output a may be updated as long as sub-protocol π is in execution.

Another shorthand used is as follows. Let *condition* denote some relation (e.g. 'there are at least k elements in the accumulative set'). We use "Wait until *condition*" to denote "If *condition* holds, continue to the next instruction. Else, end this cycle." In the sequel we do not distinguish between protocols and sub-protocols.

4.2 Primitives

In this section we describe primitives used in our constructions, in both the Fail-Stop and the Byzantine cases. We first define the requirements of each primitive, and then describe (or give a reference to) an implementation.

4.2.1 Byzantine Agreement

We present the standard definition of asynchronous Byzantine Agreement. (When the adversary is Fail-Stop, this primitive is sometimes called Consensus.)

Definition 4.5 *Let π be an n-party protocol where each party has binary input. Protocol π is a t-*resilient Byzantine Agreement *protocol if the following hold, for every input, every scheduler, and every coalition of up to t corrupted parties.*

- **Termination.** *With probability 1, all the uncorrupted parties eventually complete the protocol (i.e., terminate locally).*
- **Correctness.** *All the uncorrupted parties that complete the protocol have an identical output. Furthermore, if all the uncorrupted parties have the same input, denoted σ, then all the uncorrupted parties output σ.*

Feldman [Fe] describes an $(\lceil \frac{n}{4} \rceil - 1)$-resilient asynchronous BA protocol, running in constant expected time[5]. $(\lceil \frac{n}{3} \rceil - 1)$-resilient Consensus can be reached by substituting the AVSS scheme in Chapter 5 by a simple secret sharing scheme. (For instance, use the scheme described in Section 4.3.)

[5] We define the running time of an asynchronous protocol in Section 4-A.

4.2.2 Broadcast

We present a definition of a broadcast protocol.

Definition 4.6 *Let π be an n-party protocol initialized by a special party (called the sender), having input m (the message to be broadcast). Protocol π is a t-*resilient Broadcast *protocol if the following hold, for every input, scheduler, and coalition of up to t corrupted parties.*

- **Termination.** *1. If the sender is uncorrupted, then all the uncorrupted parties eventually complete the protocol.*

  2. If any uncorrupted party completes the protocol, then all the uncorrupted parties eventually complete the protocol.

- **Correctness.** *If the uncorrupted parties complete the protocol, then they do so with a common output m^*. Furthermore, if the sender is uncorrupted then $m^* = m$.*

We stress that the Termination property of Broadcast is much weaker than the Termination property of Byzantine Agreement: for Broadcast, we do not require that the uncorrupted parties complete the protocol, in case that the sender is corrupted.

In Figure 4-1 we describe a simple Broadcast protocol for the Fail-Stop model.

Protocol BC

Code for the sender (on input m):

1. Send (MSG, m) to all the parties, and complete the protocol with output m.

Code for the other parties:

2. Upon receiving the *first* (MSG, m) or (ECHO, m) message, send (ECHO, m) to all the parties and complete the protocol with output m.

Figure 4-1: A simple broadcast protocol for Fail-stop adversaries

Proposition 4.7 *Protocol BC is an n-resilient Broadcast protocol for Fail-stop adversaries.*

Proof: If the sender is uncorrupted, then all uncorrupted parties receive a (MSG, m) message and thus complete the protocol with output m. If a uncorrupted party completed the protocol, with output m, then it has sent an (ECHO, m) message to all the parties, thus every uncorrupted party will complete the protocol with output m. □

Bracha [Br] describes an $(\lceil \frac{n}{3} \rceil - 1)$-resilient Broadcast protocol for the Byzantine setting. For self-containment, we present Bracha's Byzantine Broadcast (BB) protocol in Figure 4-2.

Convention: in the sequel, we use "party P received an m broadcast" to shorthand "party P completed a Broadcast protocol with output m". We assume that the identity of the sender appears in m.

Protocol BB

Code for the sender (on input m):

1. send message (MSG, m) to all the parties.

Code for party P_i:

2. Upon receiving a message (MSG, m), send (ECHO, m) to all the parties.
3. Upon receiving $n-t$ messages (ECHO, m') that agree on the value of m', send (READY, m') to all the parties.
4. Upon receiving $t+1$ messages (READY, m') that agree on the value of m', send (READY, m') to all the parties.
5. Upon receiving $n-t$ messages (READY, m') that agree on the value of m', send (OK, m') to all the parties and accept message m' as a broadcast message.

Figure 4-2: Bracha's Broadcast protocol

4.2.3 Agreement on a Core Set

We first illustrate a setting in which an Agreement on a Core Set (ACS) primitive is needed. Assume each party initiates a Broadcast of some value; next, the parties wish to agree on a *common* 'core' set of at least $n-t$ parties whose Broadcast has been successfully completed. Clearly, each uncorrupted party can wait to (locally) complete $n-t$ Broadcasts, and keep the senders of these Broadcasts. However, if more than $n-t$ Broadcasts have been (globally) completed, then we cannot be sure that all the uncorrupted parties keep the same set of parties. For this purpose, the parties use the Agreement on a Core Set (ACS) primitive: the parties' output of this primitive, when used in this context, is a common set of at least $n-t$ parties whose Broadcast has been completed.

We proceed to formally define the properties required of an ACS primitive. Recall the definition of an accumulative set (described in Section 4.1.3, on page 53): we let $\mathcal{U}^{(c)}$ denote the value of variable $\mathcal{U}$ in cycle c of the protocol; An accumulative set $\mathcal{U}$ is a variable holding a set, such that for every two cycles c and d, if c precedes d then $\mathcal{U}^{(c)} \subseteq \mathcal{U}^{(d)}$.

Definition 4.8 *Let $m, M \in \mathbf{N}$ (in out context, we use $m = n-t$ and $M = n$), and let $\mathcal{U}_1 \ldots \mathcal{U}_n \subseteq [M]$ be a collection of accumulative sets, so that party P_i has $\mathcal{U}_i$. We say that the collection is (m,t)-uniform, if the following hold, for every scheduler and every coalition of up to t corrupted parties.*

- *For every uncorrupted party P_i there exists a cycle c such that $|\mathcal{U}_i^{(c)}| \geq m$.*
- *For every two uncorrupted parties P_i and P_j, if $k \in \mathcal{U}_i^{(a)}$ for some cycle a within party P_i, then there exists a cycle b within party P_j so that $k \in \mathcal{U}_j^{(b)}$ (namely, P_i and P_j will eventually have $\mathcal{U}_i = \mathcal{U}_j$).*

Definition 4.9 *Let $m \leq M$, and let π be an n-party protocol with parameters m, M, where the input of every uncorrupted party P_i is an accumulative set $\mathcal{U}_i$. Protocol π is a t-resilient* protocol for Agreement on a Core Set *(with parameters m, M), if the following hold, for every coalition of up to t corrupted parties, and every scheduler.*

- **Termination.** *If the collection $\mathcal{U}_1 \ldots \mathcal{U}_n$ is (m,t)-uniform, then with probability 1 all the uncorrupted parties eventually complete the protocol.*
- **Correctness.** *All the uncorrupted parties complete the protocol with a common output $C \subseteq [M]$ so that $|C| \geq m$. Furthermore, every uncorrupted party has $C \subseteq \mathcal{U}_i^*$, where $\mathcal{U}_i^*$ is the value of $\mathcal{U}_i$ upon the completion of protocol π.*

Before presenting our construction, let us remark that in [BKR] a simpler construction of an $(\lceil \frac{n}{3} \rceil - 1)$-reslient protocol for agreement on a core set is described. Furthermore, the [BKR] protocol runs in constant expected time. Let $n \geq 3t+1$. Our construction, with parameters m and M, consists of two phases:

Phase I: In the first phase, each party first waits until its dynamic input is of size m; then, it performs $\log_2 n$ iterations. In each iteration, the party sends the current contents of its dynamic input to all the other parties; then, the party collects the sets sent by the other parties in this iteration, and waits until its dynamic input contains $n-t$ such sets; then, the party continues to the next iteration. It will be shown below that the intersection of the dynamic inputs of all the uncorrupted parties that have completed this phase is of size at least m. [6]

Phase II: In the second phase, the parties concurrently run M Byzantine Agreement protocols; in the cth Byzantine Agreement, the parties decide whether element $c \in [M]$ will be in the agreed set C. Namely, each party's input to the cth Byzantine Agreement is 1 iff c belongs to the party's dynamic input; the element c belongs to the set C, iff the (common) output of the cth Byzantine Agreement is 1. The properties of Byzantine Agreement assure us that the set C contains the intersection of the dynamic inputs of all the parties that have completed the first step; therefore, the set C is large enough.

Remark: Our construction applies to both the Fail-Stop and the Byzantine cases, using the appropriate agreement primitive (either BA or Consensus).

Protocol ACS is described in Figure 4-3.

Proposition 4.10 *Protocol $ACS[m, M]$ is a $min(r, \lceil \frac{n}{3} \rceil - 1)$-resilient protocol for Agreement on a Core Set in a network of n parties, where r is the resilience of the agreement protocol used.*

Proof: We first assert the Termination condition. Assume that the accumulative sets $\mathcal{U}_1 \ldots \mathcal{U}_n$ define an (m,t)-uniform collection. We show that every uncorrupted party P_i will eventually complete the protocol.

Clearly, Step 1 will be completed. It can be seen, by induction on the number of iterations, that every iteration in Step 2 will be completed: in each iteration r, party P_i will eventually receive the (r, S_j) message from every uncorrupted party. Furthermore, for every

[6] This problem of assuring a large intersection of the sets of a uniform collection was previously addressed by [Fe], as well as [BE]. Both works present partial solutions to this problem: let $m = n - t$; Feldman [Fe] made sure that the intersection of the sets held by all the uncorrupted parties is of size $\Theta(n)$ (but not necessarily m); Ben-Or and El-Yaniv [BE] made sure that the intersection of the sets held by $n-2t$ uncorrupted parties is of size at least m.

Protocol ACS$[m, M](\mathcal{U}_i)$

Party P_i acts as follows, on inputs m, M and accumulative set $\mathcal{U}_i$.

1. Wait until $|\mathcal{U}_i| \geq m$.
2. For $0 \leq r \leq \lfloor \log n \rfloor$ do
   (a) Send $(r, \mathcal{U}_i)$ to all the parties.
   (b) Let $F_i^r = \{P_j |$ an (r, S_j) message was received from $P_j\}$.
   Wait until $S_j \subseteq \mathcal{U}_i$ for at least $n - t$ parties $P_j \in F_i^r$.
   (Note that sets S_j that were not contained in $\mathcal{U}_i$ when they were received, may later be contained in $\mathcal{U}_i$ if elements are added to it.)
3. Run M Byzantine Agreement protocols $BA_1 \ldots BA_M$, with input 1 to BA_j iff $j \in \mathcal{U}_i$.
4. Set $C_i = \{j |$the output of BA_j is $1\}$. Wait until $C_i \subseteq \mathcal{U}_i$.
5. Output C_i.

Figure 4-3: ACS - A protocol for Agreement on a Core Set

uncorrupted party P_j, party P_i will eventually have $S_j \subseteq \mathcal{U}_i$ (since the collection $\mathcal{U}_1 \ldots \mathcal{U}_n$ is uniform).

Step 3 will be completed with probability 1, since all the Byzantine Agreement protocols are completed with probability 1. To see that Step 4 will be completed, we note that if a Byzantine Agreement protocol BA_j has output 1, then there exists a uncorrupted party P_k that started BA_j with $j \in \mathcal{U}_k$; thus, party P_i will eventually have $j \in \mathcal{U}_i$.

We assert the Correctness property. The unanimity of the outputs of the parties follows directly from the definition of Byzantine Agreement. Step 4 of the protocol assures us that $C \subseteq \mathcal{U}_i^*$, within each uncorrupted party P_i. It is left to show that $|C| \geq m$.

Let U_i^r denote the value of accumulative set $\mathcal{U}_i$ upon completion of iteration r of Step 2 by party P_i. We show by induction on r that every set D of at most 2^r uncorrupted parties, all of which have completed iteration r, satisfies $|\cap_{j \in D} U_j^r| \geq m$.

The base of induction ($r = 0$) is immediate from Step 1 of the protocol. For the induction step, consider a set D of up to 2^r uncorrupted parties. We first observe that for every two parties $P_j, P_k \in D$, we have $|F_j^r \cap F_k^r| \geq t + 1$, since $n \geq 3t + 1$ and $|F_j^r| \geq n - t$ for each j. Therefore, there exists at least one *uncorrupted* party $P_l \in F_j^r \cap F_k^r$; we call P_l an *arbiter* of P_j and P_k. Party P_j (resp. P_k) has received the (r, S_l) message; furthermore, $U_l^{r-1} \subseteq S_l$. Thus, $U_l^{r-1} \subseteq U_j^r$ (resp. $U_l^{r-1} \subseteq U_k^r$).

Consider an arbitrary partition of the set D to pairs of parties, choose an arbiter for each pair, and let D' be the set of these arbiters; thus, $|D'| \leq 2^{r-1}$. By the induction hypothesis, we have $m \leq |\cap_{j \in D'} U_j^{r-1}|$. Every party in D has an arbiter in D', thus $\cap_{j \in D'} U_j^{r-1} \subseteq \cap_{j \in D} U_j^r$. Therefore, we have $m \leq |\cap_{j \in D} U_j^r|$.

Let D be the set of uncorrupted parties that start Step 3 of the protocol, and let $\hat{C} = \cap_{j \in D} U_j^{\log n}$; by the above induction, we have $|\hat{C}| \geq m$. For every $j \in \hat{C}$, the inputs of all the uncorrupted parties to the BA_j protocol is 1. Therefore, by the definition of Byzantine Agreement, the output of every BA_j protocol is 1. Thus, $\hat{C} \subseteq C$ and $|C| \geq m$.

□

4.3 Fail-Stop faults

We show how to securely t-compute any function whose input is partitioned among n parties, when $n \geq 3t+1$ and the faults are Fail-Stop. Our costruction follows the outline of the first [BGW] construction (namely, the construction resilient against corrupted parties which only try to gather information but otherwise follow the protocol).

Let F be a finite field known to the parties with $|F| > n$, and let $f : F^n \rightarrow F$ be the computed function. (Namely, we restrict our discussion to functions where the input of each party, as well as the function value, are field elements. We note that no generality is lost in this restriction: if the range of the computed function does not 'fit into' F, then the function can be partitioned into several functions whose range is F. Furthermore, a party can have more than one field element for input; in this case, it executes a different copy of the protocol for each field element.)

We assume that the parties have an arithmetic circuit computing f; the circuit consists of addition and multiplication gates of in-degree 2 (we regard adding a constant as a special case of addition). All the computations in the sequel are done in F.

An outline of the protocol follows. Let x_i be the input of P_i. As a first step, each party shares its input among the parties, using a technique similar to Shamir's secret sharing scheme [Sh]. (Namely, for each party P_i that successfully shared its input, a random polynomial $p_i(\cdot)$ of degree t is generated, so that each party P_j has $p_i(j)$ and $p_i(0)$ is P_i's input value. We say that $p_i(j)$ is P_j's **share** of $p_i(0)$.) Next, the parties agree, using protocol ACS, on a core set C of parties that have successfully shared their input. Once C is computed, the parties proceed to compute $f_C(\vec{x})$, in the following way. First, the input values of the parties not in C are set to a default value, say 0; then, the parties evaluate the given circuit, gate by gate, in a way described below. Note that the output of the protocol is fixed once the set C is fixed.

For each gate, the parties use their shares of the input lines to jointly and securely 'generate' a random polynomial $p(\cdot)$ of degree t such that every party P_i computes $p(i)$, and $p(0)$ is the output value of this gate. (Namely, $p(i)$ is P_i's share of the output line of this gate.) Once enough parties have computed their shares of the output line of the entire circuit, the parties reveal their shares of the output line, and interpolate the output value.

In the rest of this section, we describe our construction in more detail. First, we describe the Global-Share and Reconstruct protocols. Next, we describe the evaluation of a linear gate and of a multiplication gate. Finally, we put all the pieces together to describe the main protocol, and prove its correctness.

When presenting each protocol, we describe the uncorrupted parties' outputs of the protocol. We postpone the formal proof of security to Section 4.3.5.

4.3.1 Global-Share and Reconstruct

The Global-Share protocol, denoted GShare, consists of two phases. First, each party shares a secret among the parties; next, the parties use an ACS protocol in order to agree on a set, of size at least $n-t$, of parties that have successfully shared their secret. Party P_i's output of the GShare protocol is the set of parties that have successfully shared their inputs, along with the ith share of the input of each of the parties in this set.

This protocol is applied once at the beginning of the protocol (where each party shares its input of the main protocol), and twice in each invocation of the protocol for computing a multiplication gate (see Section 4.3.3). Protocol GShare has a 'security parameter', d. This parameter will be set to either t or $2t$, according to the context in which the protocol is activated. Protocol GShare is presented in Figure 4-4.

Protocol GShare$[d](x_i)$

Party P_i acts as follows, on inputs x_i, d.

1. "Share x_i":
   (a) Choose a random polynomial $h_i(\cdot)$ of degree d, such that $h_i(0) = x_i$.
   For each $1 \leq j \leq n$, send $h_i(j)$ to party P_j.
   (b) Broadcast `'Party` P_i `completed sharing'`.
2. Upon receiving the share $s_{i,j}$ of P_j's secret and the `'Party` P_j `completed sharing'` broadcast, add j to a set C_i of parties that have successfully shared their secret.
   Set $C =$ACS$[n-t, n](C_i)$.
3. Output C, and $\{s_{i,j} | j \in C\}$.

Figure 4-4: GShare - The global sharing protocol

We note that the accumulative sets $C_1 \ldots C_n$ of Step 2 define an $(n-t, t)$-uniform collection. Thus, all the uncorrupted parties complete protocol ACS (and, hence, the entire GShare protocol) with the desired output. Furthermore, the corrupted parties have no information about the values shared by the uncorrupted parties.

In the Reconstruct protocol, described in Figure 4-5, the parties reconstruct a secret from its shares. The parameters of this protocol are the 'security parameter' d, and a set R of the parties to which the secret is to be revealed. Party P_i's input is the i-th share of the secret, denoted s_i. The Reconstruct protocol is invoked once for every multiplication gate, and once at the end of the protocol. (Parameter R is necessary in the multiplication gate invocation. In the other invocation it is set to $R = [n]$.)

We note that if $d + 1 \leq n - t$ then a Reconstruct protocol that is run by all the parties will be completed. Furthermore, if there exists a polynomial, $p(\cdot)$, of degree d such that the share of each active party P_j is $s_j = p(j)$, then all the uncorrupted parties in R will output $p(0)$.

4.3.2 Evaluating a linear gate

We describe the evaluation of a linear gate, instead of a simple addition gate; this more general formulation will be convenient in presenting the protocol for computing a multiplication gate.

Evaluating a linear gate is easy and requires no communication. Let $c = \sum_{j=1}^{k} \alpha_j \cdot a_j$ be a linear gate, where $a_1 \ldots a_k$ are the input lines of the gate, $\alpha_1 \ldots \alpha_k$ are fixed coefficients,

Protocol Reconstruct$[d, R](s_i)$

Party P_i acts as follows, on input s_i, and parameters $d \in \{t, 2t\}$ and $R \subseteq \{P_1, \ldots, P_n\}$.

1. Send s_i to all the parties in R.
2. If $P_i \notin R$, output 0.
   Otherwise, upon receiving $d+1$ values (value v_j from party P_j), interpolate a polynomial $p_i(\cdot)$ of degree d such that $p(j) = v_j$ for each received value v_j.
   Output $p_i(0)$.

Figure 4-5: Reconstruct - The reconstruction protocol

and c is the output line. Let $A_j(\cdot)$ be the polynomial associated with the jth input line; namely, party P_i's share of this line is $a_{i,j} = A_j(i)$.

Each party P_i locally sets its share of the output line to $c_i = \sum_{j=1}^{k} \alpha_j \cdot a_{i,j}$. It can be easily seen that the shares $c_1 \ldots c_n$ define a random polynomial $C(\cdot)$ of degree t with the correct free coefficient, and with $C(i) = c_i$ for all i.

4.3.3 Evaluating a multiplication gate

Let $c = a \cdot b$ be a multiplication gate, and let $A(\cdot), B(\cdot)$ be the polynomials associated with the input lines; namely, each party P_i's shares of these lines are $A(i)$ and $B(i)$, respectively. The parties will jointly compute their shares of a random polynomial $C(\cdot)$ of degree t, satisfying $C(0) = A(0) \cdot B(0)$; namely, each uncorrupted party P_i's share of the output line will be $C(i)$.

Initially, each party locally computes its share of the polynomial $E(\cdot) = A(\cdot) \cdot B(\cdot)$, by setting $E(i) = A(i) \cdot B(i)$. Clearly, $E(\cdot)$ has the required free coefficient. However, $E(\cdot)$ is of degree $2t$; moreover, it is not uniformly distributed. The parties use their shares of $E(\cdot)$ to compute their shares of the desired polynomial $C(\cdot)$, in a secure manner. The computation proceeds in two steps: first, the parties jointly generate a *random* polynomial, $D(\cdot)$, of degree $2t$, so that $D(0) = E(0)$ (namely, each party P_i will have $D(i)$). Next, the parties will use their shares of $D(\cdot)$ in order to jointly compute their shares of polynomial $C(\cdot)$. These steps are described below.

Randomization. We describe how polynomial $D(\cdot)$ is generated. First, the parties generate a random polynomial $H(\cdot)$ of degree $2t$ and with $H(0) = 0$; namely, each party P_i will have $H(i)$.

We first describe how the polynomial $H(\cdot)$ is shared in a synchronous setting [BGW]. Each party P_i selects a random polynomial $H_i(\cdot)$ of degree $2t$ with $H_i(0) = 0$, and shares it among the parties; namely, each party P_j receives $H_i(j)$. Polynomial $H(\cdot)$ is set to $H(\cdot) = \sum_{j=1}^{n} H_j(\cdot)$; namely, each party P_i computes $H(i) = \sum_{j=1}^{n} H_j(i)$.

In our asynchronous setting, a party cannot wait to receive its share from all the other parties. Instead, the parties agree, using protocol ACS, on a set C of parties that have successfully shared their H_i polynomials; next, polynomial $H(\cdot)$ will be set to $H(\cdot) =$

$\sum_{j\in C} H_j(\cdot)$. In other words, the parties run protocol GShare[2t](0); on output $(C, \{H_j(i)|j \in C\})$ of the GShare protocol, party P_i computes $H(i) = \sum_{j\in C} H_j(i)$.

Polynomial $D(\cdot)$ is now defined as $D(\cdot) = E(\cdot) + H(\cdot)$; namely each party P_i computes $D(i) = E(i) + H(i)$. Clearly, $D(0) = A(0) \cdot B(0)$, and all the other coefficients of $D(\cdot)$ are uniformly and independently distributed over F.

Degree-reduction. In the degree-reduction step, the parties use their shares of $D(\cdot)$ in order to jointly and securely compute their shares of a random polynomial $C(\cdot)$ of degree t, with $C(0) = D(0)$. Polynomial $C(\cdot)$ will be set to the 'truncation' of polynomial $D(\cdot)$ to degree t; namely, the $t+1$ coefficients of $C(\cdot)$ are the coefficients of the $t+1$ lower degrees of $D(\cdot)$. An important observation is that the information gathered by the corrupted parties (namely, t shares of polynomial $D(\cdot)$, along with t shares of the truncation polynomial $C(\cdot)$), is independent of $C(0)$. This statement is formalized in Lemma 4.16 on page 70, and used in proving the security of the entire protocol.

Let $\vec{d} = D(1)\ldots D(n)$, and let $\vec{c} = C(1)\ldots C(n)$. [BGW] noted that there exists a fixed $n \times n$ matrix M such that $\vec{c} = \vec{d}M$. It follows that the desired output of each party P_i is a linear combination of the inputs of the parties: $C(i) = [\vec{d}M]_i = \sum_{j=1}^{n} D(i) \cdot M_{j,i}$. We colloquially call such an operation 'multiplying the inputs by a fixed matrix'. (In this case, the parties' inputs are their shares of the polynomial $D(\cdot)$.)

In a synchronous setting, 'multiplying the inputs by a fixed matrix' can be done by securely computing the appropriate n fixed linear combinations of the inputs, so that the value of the ith linear combination is revealed to party P_i only. Linear combinations are securely computed as follows. First, each party shares its input; next, each party computes the linear combination of its shares (as in Section 4.3.2), and reveals this linear combination to the specified party; finally, the specified party computes the output value by interpolating a degree t polynomial from the received combinations. [7]

Linear combinations of *all* the inputs cannot be computed in an asynchronous setting. Thus, the synchronous method described above cannot be used. We outline our solution for the asynchronous setting. First, we describe a technique for multiplying inputs by a fixed matrix in an asynchronous setting, for the case where the matrix and the set of inputs are related in a special way described below. Next, we note that the matrix M and the set of possible inputs of the degree-reduction step are related in this special way.

Definition 4.11 *Let A be an $n \times n$ matrix, and let $S \subseteq F^n$ be a set of input vectors. we say that S is t-*multipliable* by A, if for every set $G \subseteq [n]$ with $|G| \geq n-t$, there exists an (easily computable) matrix A_G, of size $|G| \times n$, such that for every input $\vec{x} \in S$, we have $\vec{x}_G \cdot A_G = \vec{x} \cdot A$.*

Let S be t-multipliable by A. Then, the protocol described in Figure 4-6 'multiplies inputs in S by the matrix A'. Let $\vec{x} \in S$. The parties first execute a GShare protocol (applied on the input vector $\vec{x}$); once the common set G is computed, each party locally computes A_G. Next, the parties run n Reconstruct protocols; in the ith Reconstruct, the

[7] Note that computing a linear combination of input values is harder than computing a linear combination of already-shared values (i.e., the problem addressed in section 4.3.2): the former computation involves *sharing* of all the inputs.

parties let P_i compute $[\vec{x}_G \cdot A_G]_i$, by sending him the appropriate linear combination of their shares.

Protocol MAT(x_i, A)

Party P_i acts as follows, on input x_i, and matrix A.

1. Set $(G, \{s_{i,j} | j \in G\})$ =GShare$[t](x_i)$.
   Enumerate $G = g_1, \ldots, g_{|G|}$, and let $\vec{s_i} = s_{i,g_1} \ldots s_{i,g_{|G|}}$.
2. Compute A_G.
3. For $1 \le k \le n$, set y_k =Reconstruct$([\vec{s_i} \cdot A_G]_k, t, \{k\})$.
   Output y_i.

Figure 4-6: MAT - A protocol for multiplying the input by a fixed matrix

We say that an input vector $\vec{x}$ is d-**generated** if there exists a polynomial $P(\cdot)$ of degree d satisfying $x_i = P(i)$ for every i; the set of possible inputs of the degree-reduction step is the set of $2t$-generated vectors.

Proposition 4.12 *Let M be the $n \times n$ matrix introduced in* [BGW]. *Then, the set of $2t$-generated n-vectors is t-multipliable by M, when $n \ge 3t + 1$.*

Proof: Recall that matrix M used in [BGW] is constructed as $M = V^{-1}TV$, where V is a (Vandermonde) $n \times n$ matrix defined by $V_{i,j} = i^j$, and T is constructed by setting all but the first $t + 1$ rows of a Unit matrix to 0. (Let $\vec{d}, \vec{c}$ be as defined above. To see that $\vec{d} \cdot M = \vec{c}$, note that $\vec{d} \cdot V^{-1}$ is the coefficients vector of the polynomial $D(\cdot)$; thus $\vec{d} \cdot V^{-1}T$ is the coefficients vector of $C(\cdot)$, and $\vec{d} \cdot V^{-1}TV = \vec{c}$.)

Let $G \subseteq [n]$ with $|G| \ge n - t$, and let $G = g_1 \ldots g_{|G|}$. Matrix M_G is constructed as follows. Let the $|G| \times |G|$ matrix V^G be the matrix V projected on the indices in G; namely, $V^G_{i,j} = (g_i)^j$. Next, construct the $|G| \times n$ matrix $\hat{V}$ by appending $n - |G|$ zero columns to $(V^G)^{-1}$. Finally, set $M_G = \hat{V}TV$, where T is defined above.

Since $n \ge 3t + 1$, we have $|G| \ge 2t + 1$; it can be verified that in this case, $\vec{x}_G \cdot \hat{V}$ is, once again, the coefficients vector of $D(\cdot)$ (namely, $\vec{x}_G \cdot \hat{V} = \vec{x} \cdot V^{-1}$). Thus, $\vec{x}_G\hat{V}TV = \vec{x} \cdot V^{-1}TV = \vec{x} \cdot M$ as required. □

Combining the Randomization and Degree-reduction steps, we derive a protocol for computing a multiplication gate. This protocol, denoted MUL, is presented in Figure 4-7.

4.3.4 The main protocol

Let $f : F^n \to F$ be given by an arithmetic circuit A. Our protocol for securely t-computing f is described in Figure 4-8.

Theorem 4.13 *Let $f : F^n \to F$, for some field F with $|F| > n$, and let A be a circuit computing f. Then, protocol* FScompute$[A]$ *asynchronously* $(\lceil \frac{n}{3} \rceil - 1)$*-securely computes f in the bounded secure channels setting with n non-erasing parties, provided that only Fail-Stop adversaries are considered.*

Protocol MUL(a_i, b_i)

Party P_i acts as follows, on inputs a_i, b_i.

1. Set $(C', \{h_{i,j} | j \in C'\})$ =GShare$[2t](0)$.
   Let $d_i = a_i \cdot b_i + \sum_{j \in C'} h_{i,j}$.
2. Let $M_{n \times n}$ be the matrix introduced in [BGW].
   Set c_i =MAT(d_i, M).
   Output c_i.

Figure 4-7: MUL - A protocol for computing a multiplication gate

Protocol FScompute$[A](x_i)$

Party P_i acts as follows, on local input x_i, and given circuit A.

1. Set $(C, \{s_{i,j} | j \in C\})$ =GShare$[t](x_i)$.
   For a line l in the circuit, let $l^{(i)}$ denote the share of party P_i in the value of this line. If l is the jth input line of the circuit, then set $l^{(i)} = s_{i,j}$ if $j \in C$, and $l^{(i)} = 0$ otherwise.
2. For each gate g in the circuit, wait until the ith shares of all the input lines of g are computed. Then, do the following.
   (a) If g is an addition gate with output line l and input lines l_1 and l_2, then set $l^{(i)} = l_1^{(i)} + l_2^{(i)}$.
   (b) If g is a multiplication gate $l = l_1 \cdot l_2$, then set $l^{(i)}$ =MUL$(l_1^{(i)}, l_2^{(i)})$.
3. Let l_{out} be the output line of the circuit.
   Once $l_{out}^{(i)}$ is computed, send 'Ready' to all the parties.
   (The only role of the 'Ready' messages is simplifying the proof of correctness, below.)
4. Wait to receive $n - t$ 'Ready' messages.
   Set y =Reconstruct$[t, [n]](l_{out}^{(i)})$.
5. Output (C, y).

Figure 4-8: FScompute - The protocol for Fail-Stop faults

We present the proof of Theorem 4.13 in two steps. First, we show security of the protocol in the presence of non-adaptive adversaries. (When the adversary is non-adaptive there is no difference between the cases where the uncorrupted parties are semi-honest or honest. We thus assume that the uncorrupted parties are honest.) We do not present a definition of non-adaptively, secure computation in asynchronous networks. However, such a definition can be easily assembled from the definition of adaptive security in asynchronous networks (Definition 4.4), and definition of non-adaptive security in synchronous networks (Definition 2.4).

Next we show adaptive security of protocol FScompute in the presence of non-erasing parties. We remark that protocol FScompute can be shown adaptively secure also in the presence of honest-looking parties, using similar constructs to those described in Section 3.4.

4.3.5 Proof of correctness — non-adaptive case

let $\mathcal{A}$ be a non-adaptive Fail-stop adversary, and let B be the set of corrupted parties (we incorporate the scheduler in $\mathcal{A}$).

Termination

We use FS to shorthand protocol FScompute$[A]$. We show that for every input $\vec{x}$, with probability 1 all the uncorrupted parties terminate protocol FS.

Protocol FS consists of several invocations of protocols GShare and Reconstruct; if a uncorrupted party terminates all these invocations, then this party terminates the entire protocol as well. We have seen that a Reconstruct protocol always terminates, provided that $n \geq 3t+1$. If all the Consensus protocols invoked in a GShare protocol terminate, then the GShare protocol terminates as well.

Each one of these Consensus protocols terminates with probability 1, by the definition of Consensus. Thus, with probability 1, all the Consensus protocols invoked in the entire protocol terminate; therefore, with probability 1, all the GShare protocols terminate as well.

Security

For any real-life non-adaptive $(\lceil \frac{n}{3} \rceil - 1)$-limited real-life adversary $\mathcal{A}$, we construct a ideal-model adversary $\mathcal{S}$, so that for every input $\vec{x}$, we have $\text{IDEAL}_{f,\mathcal{S}}(\vec{x}) = \text{EXEC}_{\text{FS},\mathcal{A}}(\vec{x})$. (The notations $\mathcal{S}$ and $\text{IDEAL}_{f,\mathcal{S}}(\cdot)$, as well as $\text{EXEC}_{\pi,\mathcal{A}}(\cdot)$ for a protocol π (here π =FS), are defined in Section 4.1.2, on page 23.) Here the set of corrupted parties, denoted B, is fixed. We represent the ideal-model adversary $\mathcal{S}$ as a quadruple $A = (B, h, c, O)$.

Construction of the ideal-model adversary. We remark that adversary $\mathcal{S}$ operates via black-box simulation of the real-life adversary. (Black-box simulation was described, for the synchronous setting, in Section 2.3. The asynchronous version is analogous.) Here the set B of parties corrupted by the real-life adversary $\mathcal{A}$ is fixed. Adversary $\mathcal{S}$ corrupts the parties in B. The input substitution function h, the core set selection function c and the output function O are computed via the following simulated interaction between the real-life adversary $\mathcal{A}$ and parties executing protocol FS.

On input $\vec{x}_B$ and random input r, $\mathcal{S}$ extends $\vec{x}_B$ to an n-vector $\vec{x}'$ by filling the extra entries with, say, zeros. Partition r into n sections $r_1 \ldots r_n$, and let $\vec{r} = r_1 \ldots r_n$. Simulate an execution of protocol FS with adversary $\mathcal{A}$, and with x'_i as input and r_i as random input of each party P_i. (In the sequel we incorporate D in $\mathcal{A}$). Let C be the core set decided upon in the GShare invocation in Step 1 of the protocol. For each $P_i \in C$, let y_i be the value shared by P_i (namely, $y_i = x_i$); for $P_i \notin C$, let $y_i = 0$. Let $\vec{y} = y_1 \dots y_n$. Set $h(\vec{x}_B, r) = \vec{y}_B$, and $c(\vec{x}_B, r) = C$. [8]

Our next step is computing the output function O of the corrupted parties in the ideal model. The inputs to this function are $\vec{x}_B$, r, and $f_C(\vec{y})$. (Recall that $f_C(\vec{y})$ is the value received from the trusted party). We continue the above simulation of protocol FS, with the following modification. Step 3 of the protocol assures us that by the time the first uncorrupted party starts executing the Reconstruct protocol of Step 4, at least $t+1$ uncorrupted parties have computed their shares of the output line. In the proof of Lemma 4.15 below, we show that these shares determine a unique polynomial of degree t; let $p(\cdot)$ denote this polynomial.[9] Once the first uncorrupted party reaches the final Reconstruct protocol, the simulator computes the above polynomial $p(\cdot)$, and chooses a random polynomial, $p'(\cdot)$, of degree t, such that $p'(0) = f_C(\vec{y})$, and $p'(i) = p(i)$ for every *corrupted* party P_i. In the simulated run, every *uncorrupted* party P_j executes the last Reconstruct protocol with input $p'(j)$, instead of $p(j)$. Let w denote the output of the corrupted parties after this simulated run of protocol FS. Set $O(\vec{x}_B, r, f_C(\vec{y})) = w$. [10]

Validity of the ideal-model adversary. It is left to show that for every input vector $\vec{x}$, the output vector $\text{EXEC}_{\text{FS},\mathcal{A}}(\vec{x})$ of the parties in the 'real' computation has the same distribution as the output vector $\text{IDEAL}_{f,\mathcal{S}}(\vec{x})$ of the parties in the ideal model. An outline of our proof follows. Recall the definition of the **adversary view** of the computation (we re-define this notion in more detail below). This notion captures the information gathered by the adversary during the computation, combined with the information seen by the scheduler. In Lemma 4.14 below, we show that the adversary view of the real computation is distributed equally to the adversary view in the interaction simulated by the ideal-model adversary. We complete our proof by showing that whenever the adversary view of the real execution equals the adversary view of the simulated execution, the output vector of *all* the parties in the real-life model equals the output vector of *all* the parties in the ideal model.

We proceed to re-define the adversary view. First, define the **transcript** of a computation. The transcript U consists of the inputs $\vec{x}$ of all the parties, their random inputs $\vec{r}$, and the entire communication among the parties. The communication is organized in **events**; each event, e, consists of a message received by a party, and its corresponding responses. The order of events is induced by the order of delivery of the messages. Namely, we have

$$U = \vec{x}, \vec{r}, e_1, \dots, e_k.$$

We partition each message in protocol FS to its **contents** and its **frame**. The contents of

[8] Since the faults are Fail-Stop, we could let the input substitution function h be the identity function. However, this formulation is valid for the proof of Theorem 4.30 (Security of the Byzantine protocol) as well. Note that $f(\vec{y}) = f_C(\vec{y})$; in the sequel, we leave the subscript C for clarity.

[9] Step 3 of the algorithm is a technical step whose purpose is making the fact that $p(\cdot)$ is fixed at this stage more obvious. It can be seen that, even without Step 3, by the time the first uncorrupted party starts executing the last Reconstruct protocol, at least $n-2t$ uncorrupted parties have computed their shares of the output line.

[10] The output of the corrupted parties is defined as the contents of their output tapes after all the messages sent by the uncorrupted parties have been received.

a message consists of the elements of the field F, appearing in this message. The frame of a message is the message itself where each field element is replaced by a 'blank'. In particular, the frame contains the type of the message, the name of the protocol and invocation that the message belongs to, and the identity of the sender and the receiver. Messages that do not contain any field elements have empty contents. (Informally, the scheduler sees only the frame of each message sent, while the receiver sees the entire message.) [11]

Let S be a set of parties, and let $U = \vec{x}, \vec{r}, e_1, \ldots, e_k$ be a transcript of some computation. The **view** of the computation, as seen by the parties in S, is the sequence $U_S = \vec{x}_S, \vec{r}_S, \hat{e}_1, \ldots, \hat{e}_k$, where $\vec{x}_S$ is the inputs of the parties in S and $\vec{r}_S$ is their random inputs. Each $\hat{e}_i$ is a **semi-event**; namely, if the recipient party of event e_i is in S, then $\hat{e}_i$ consists of the full received message and the subsequent responses. Otherwise, $\hat{e}_i$ consists of only the frame of the received message and responses. The **adversary view** is the view of the set B of corrupted parties. We note that the adversary view contains all the information gathered by the adversary during the relevant computation. Without loss of generality, we assume that the corrupted parties' output of a computation is the adversary view. Let $\hat{O}$ denote this function.

Fix an input vector $\vec{x}$. Define the random variable μ: for each random input $\vec{r}$, set μ to be the corrupted parties' view of the execution of protocol FS on input $\vec{x}$, random input $\vec{r}$, and adversary $\mathcal{A}$. Let the random variable μ' be similarly defined, with respect to the simulated protocol described above. In Lemma 4.14 below we show that μ and μ' are identically distributed.

Lemma 4.14 *The random variables μ and μ' are identically distributed.*

(The proof of Lemma 4.14 is given at the end of this section.)

It is left to show that the following two quantities are equal, with respect to every possible view, V, of the corrupted parties (i.e., every view V in the support set of μ and μ'):[12]

(a) The output of *all* the parties in the ideal model with the adversary $\mathcal{S}$ described above, when the corrupted parties' view of the simulated execution is V.

(b) The output of *all* the parties after an execution of protocol FS, when the view of the corrupted parties is V.

Equality is proven by showing that in both cases, the output of the corrupted parties is $\hat{O}(V)$, and the output of the uncorrupted parties is $(C, f_C(\vec{y}))$, where C is the core set appearing in V and $\vec{y}$ is the substituted input vector defined above. (Note that both C and $\vec{y}$ are uniquely determined by $\vec{x}$ and V).

Case **(a)** (the ideal model): By the definition of the output of the parties in the ideal model (Definition 2.10 on page 23), the uncorrupted parties output $(C, f_C(\vec{y}))$, and the corrupted parties output $O(\vec{x}_B, r, f_C(\vec{y}))$; in the above construction of $\mathcal{S}$, we have set $O(\vec{x}_B, r, f_C(\vec{y})) = \hat{O}(V)$.

---

[11] Note that this partitioning of a message to its contents and frame is specific to protocol FS. A more general definition may let the frame of a message be its length only. However, the present definition of a frame is more natural for our protocol; furthermore, it simplifies our proof of Lemma 4.14 below.

[12] We use V to shorthand U_B.

Case **(b)** (an execution of protocol FS): The corrupted parties output $\hat{O}(V)$, by the definition of function $\hat{O}$. In Lemma 4.15 below, we show that the uncorrupted parties output $f_C(\vec{y})$. □

Lemma 4.15 *Consider an execution of protocol FS on input $\vec{x}$ with adversary B, and let C and $\vec{y}$ be the quantities defined above, with respect to this execution. Then, the uncorrupted parties' output of this execution is $f_C(\vec{y})$.*

Proof: For each line l in the computed circuit, let v_l be the value of this line when the input of the circuit is $\vec{y}$. Let l_{out} denote the output line of the computed circuit; since the circuit computes the function f, we have $v_{l_{out}} = f(\vec{y}) = f_C(\vec{y})$.

Consider an execution of protocol FS as in the Lemma. For each line, l, let $p_l(\cdot)$ be the lowest degree polynomial such that the share of each uncorrupted party P_i in this line, in this execution, is $p_l(i)$. It can be seen, by induction on the structure of the circuit, that every $p_l(\cdot)$ is of degree at most t and that $p_l(0) = v_l$: each line is either an input line, an output of a linear gate, or an output of a multiplication gate. These cases were dealt with in the sections describing the initial commit, the linear gate, and the multiplication gate protocol, respectively. Thus, in the final Reconstruct protocol, the uncorrupted parties retrieve (and output) $p_{l_{out}}(0) = v_{l_{out}} = f_C(\vec{y})$. □

Proof of Lemma 4.14. We show, by induction on the number of communication events, that every prefix of μ has the same distribution as the corresponding prefix of μ' with the same number of events.

We use the following notations. For every $i \geq 0$, let μ_i denote the prefix of μ that ends after the ith semi-event. Let μ'_i be similarly defined with respect to μ'. Fix an instance, V_i, of μ_i, and let ϵ_{i+1} be the random variable describing the distribution of the $(i+1)$th semi-event in μ, given that the corresponding prefix of μ is V_i. Namely, for each semi-event $\hat{e}$,

$$\text{Prob}(\epsilon_{i+1} = \hat{e}) = \text{Prob}(\mu_{i+1} = (V_i, \hat{e}) \mid \mu_i = V_i).$$

Recall that each semi-event consists of a received message and the subsequent response messages. Let σ_{i+1} and τ_{i+1} be the random variables having the distributions of the received message and the response messages in ϵ_{i+1}, respectively. Let ϵ'_{i+1}, σ'_{i+1}, and τ'_{i+1} be similarly defined with respect to μ'.

For the base of induction, we note that $\vec{x}_B = \vec{x}'_B$, and both random inputs $\vec{r}_B$ and $\vec{r}'_B$ are uniformly distributed; thus, $(\vec{x}_B, \vec{r}_B)$ and $(\vec{x}'_B, \vec{r}\,')$ are identically distributed.

For the induction step, we show that for every $i \geq 0$ and for every instance V_i, the random variables ϵ_{i+1} and ϵ'_{i+1} are identically distributed.

First, note that V_i contains all the information seen by the scheduler when delivering the $(i+1)$st message. Therefore, the frame of the $(i+1)$st message is uniquely determined by V_i. In particular, the sender (resp. the recipient) of the $(i+1)$st message is the same party in both computations. We distinguish two cases:

(a) The recipient party is uncorrupted. In this case, it is left to show that the frames of the response messages sent in this event are identically distributed. However, it can be seen, by observing the protocol, that the frames of the messages sent by a uncorrupted party are uniquely determined by the frames of the messages received by this party so far; these frames are part of the common instance V_i of the prefixes μ_i and μ'_i.

(b) The recipient party is corrupted. Clearly, if the received messages σ_{i+1} and σ'_{i+1} are identically distributed, then the response messages τ_{i+1} and τ'_{i+1} are identically distributed as well. If the sender is corrupted, then entire received message is explicitly written in the instance V_i, which is the same in both computations. It remains to consider the case in which the sender is uncorrupted. We have that the *frames* of σ_i and σ'_i are equal. We show that the *contents* of σ_i and σ'_i are identically distributed.

Every message in protocol FS belongs either to some invocation of protocol GShare, or to some invocation of protocol Reconstruct. We consider these two cases separately. In the sequel, we say that a random variable is a **semi-random polynomial** of degree d, if its instances are polynomials of degree at most d, such that all the coefficients other than the free coefficient are uniformly and independently distributed. (The free coefficient may have any distribution.)

GShare messages. The only messages of a GShare protocol having non-empty contents are shares of some secret. However, the corrupted parties receive only up to t shares of each secret shared by a uncorrupted party, namely up to t shares of a semi-random polynomial of degree t. Consequently, both in μ and in μ' the contents is distributed uniformly and independently over the field F, regardless of the shared values.

In an invocation of Reconstruct, the corrupted parties receive n shares of a semi-random polynomial $p(\cdot)$ of degree t. Thus, the contents of the first t messages are uniformly distributed, both in μ and in μ'. The contents of the other messages are uniquely determined by the t shares known to the corrupted parties (namely, the contents of the first t messages of the same invocation), and the free coefficient of $p(\cdot)$. Thus, it suffices to show that the free coefficient of $p(\cdot)$ (namely, the output of this invocation of Reconstruct) is identically distributed in μ and μ'. We distinguish two types of invocations of Reconstruct: a Reconstruct within a multiplication step (namely, a Reconstruct invoked in Step 3 of protocol MAT), and the final invocation of Reconstruct, in Step 4 of protocol FS.

Reconstruct as a part of a multiplication step. Consider an invocation of protocol MUL (see Figure 4-7 on page 64). We show that the corrupted parties' outputs of the t invocations of Reconstruct, along with the contents of the other messages of this invocation of MUL, received by the corrupted parties, are uniformly and independently distributed, regardless of the values of the input lines of the corresponding multiplication gate.

The polynomial $H(\cdot)$ generated by the parties in the Randomization phase (Step 1 of protocol MUL) is a semi-random polynomial of degree $2t$, with zero free coefficient. The contents of the GShare messages of Step 1 received by the corrupted parties add up to t shares of the polynomial $H(\cdot)$. Fix some arbitrary polynomials $A(\cdot)$ and $B(\cdot)$ associated with the input lines of the corresponding multiplication gate, and let $D(\cdot) = H(\cdot) + A(\cdot) \cdot B(\cdot)$. Then, both in μ and in μ', $D(\cdot)$ is a semi-random polynomial of degree $2t$ with some fixed free coefficient.

The data gathered by the corrupted parties during the entire invocation of MUL consists of t shares of the semi-random polynomial $D(\cdot)$, along with t shares of the truncation polynomial $C(\cdot)$ (namely, the outputs of the t invocations of Reconstruct). Let $s = A(0) \cdot B(0)$. Lemma 4.16 below shows that there exists a *unique* instance of $D(\cdot)$ (namely, a unique polynomial of degree $2t$ whose free coefficient is s), that corresponds to each sequence of $2t$ field elements gathered by the corrupted parties. Consequently, the sequence of $2t$ field-elements gathered by the corrupted parties is uniformly distributed, both in μ and in μ'. [13]

Final Reconstruct. Let C be the core set determined by V_i, and let $\vec{y}$ be the substituted input vector described above. By Lemma 4.15, the parties' output of the final Reconstruct in a computation of protocol FS is $f_C(\vec{y})$. By the construction of the ideal-model adversary $\mathcal{S}$, the parties' output of the final Reconstruct in the simulated execution is also $f_C(\vec{y})$.

This concludes our proof that μ and μ' have the same distribution. □

Lemma 4.16 (technical lemma). *Let F be a finite field with $|F| > d$, and let $s \in F$. Then for every sequence $v_1, \ldots, v_d, u_1, \ldots, u_d$ of field-elements, there exists a unique polynomial $p(\cdot)$ of degree $2d$ with $p(0) = s$, such that:*

1. *For $1 \leq i \leq d$, we have $p(i) = v_i$.*
2. *For $1 \leq i \leq d$, we have $q(i) = u_i$, where $q(\cdot)$ is the truncation of $p(\cdot)$ to degree d. (Namely, the coefficients of $q(\cdot)$ are the coefficients of the $d+1$ lower degrees of $p(\cdot)$.)*

Proof: See Section 4-B. □

4.3.6 Proof of correctness — adaptive case

Let $\mathcal{A}$ be a t-limited adaptive Fail-stop adversary, where $n \geq 3t+1$ (we incorporate the scheduler within the adversary). Termination is shown as in the non-adaptive case, provided that the Consensus protocol in use is resilient against adaptive adversaries. To show Security, we construct an ideal-model adversary $\mathcal{S} = (t, b, h, c, O)$, as in Definition 4.1. We remark that our construction of $\mathcal{S}$ uses only with black-box access to $\mathcal{A}$. Here we assume non-erasing parties. That is, each party keeps all the information gathered during the computation. This information consists of the party's input and random input, and the messages received from other parties.

Construction of the ideal-model adversary. The functions b, h, c, O are determined via a simulated execution of protocol FS with adversary $\mathcal{A}$. That is, $\mathcal{S}$ hands $\mathcal{A}$ information on the computation, in an interactive process that simulates a real-life computation. (We use the description of a real-life computation presented immediately above Definition 4.4 on page 52.)

Recall the definition of the contents and frame of a message (presented on page 66). The information handed to $\mathcal{A}$ in the simulated interaction is as follows.

1. $\mathcal{S}$ hands $\mathcal{A}$ the *frame* of each message sent by each party according to protocol FS. (This is the information seen by the scheduler. We note that protocol FS has the following property. At any point during the computation, the frames of the messages

[13] In Lemma 4.16 we assume, without loss of generality, that the corrupted parties are $P_1, \ldots, P_t$.

received by a party so far determine the frame of the next message sent by this party. Thus, the frames of the messages sent by the uncorrupted parties are predictable by the adversary.)

2. Up to the **decision point**, defined below, The *contents* of each message sent from an uncorrupted party to a corrupted party is set to uniformly chosen random elements in the field F.

3. Up to the decision point, whenever the adversary decides to corrupt a party P, the simulator $\mathcal{S}$ proceeds as follows.

   First, $\mathcal{S}$ hands $\mathcal{A}$ simulated contents of all the messages previously received by P_i. The simulated contents of the messages sent by corrupted parties, including parties that were uncorrupted at the time of sending but became corrupted since, are consistent with the contents already known to the adversary. The simulated contents of messages sent by still uncorrupted parties are chosen uniformly from F.

   Next $\mathcal{S}$ decides to corrupt P_i in the ideal-model (in the first corruption stage). Upon learning x_i, the input of P_i, the simulator chooses, in a way described below, a **simulated random input** r_i for P_i that is consistent with input x_i, with the contents of the messages previously received by P_i (as chosen above), and with the messages previously sent by P to corrupted parties. Note that r_i determines also the messages that were supposedly sent by P_i to *un*corrupted parties. Finally $\mathcal{S}$ hands $\mathcal{A}$ the input x_i and the simulated random input r_i.

Functions b, c, h are determined as follows. Let the **decision point** be the time where the first uncorrupted party invokes the **final Reconstruct** (i.e., the Reconstruct of Step 4 of protocol FS, see Figure 4-8). The selection function $b()$ of corrupted parties is determined as described in Step 3 above. Let B be the set of corrupted parties at the decision point. Let C be the core set decided upon in the initial GShare (i.e., in the GShare of Step 1 of the protocol). For each $P_i \in C$, let y_i be the value shared by P_i (namely, $y_i = x_i$); for $P_i \notin C$, let $y_i = 0$. Let $\vec{y} = y_1 \ldots y_n$. Set $h(\vec{x}_B, r) = \vec{y}_B$, and $c(\vec{x}_B, r) = C$.

The output function O of $\mathcal{S}$ is determined as follows. At the decidion point $\mathcal{S}$ asks the trusted party for the computed function value. $\mathcal{S}$ continues the above simulation of protocol FS, with the following modifications. At the decision point, $\mathcal{S}$ computes each corrupted party's share of the output line of the circuit. (This can be efficiently done based on $\mathcal{S}$'s information on the computation.) Next, $\mathcal{S}$ chooses a random polynomial $p'(\cdot)$ such that $p'(0) = f_C(\vec{y})$ and for each corrupted party P_j, the value $p'(j)$ equals P_j's share of the output line. (The polynomial $p'(\cdot)$ plays a similar role as in the non-adaptive case.) In the simulated run, each uncorrupted party P_i executes the final Reconstruct protocol with input $p'(i)$. If the adversary decides to corrupt P_i after the decision point, then $\mathcal{S}$ proceeds as in Step 3 above, with the exception that the chosen random input r_i should be consistent also with P_i having $p'(i)$ as input to the final Reconstruct. (In the ideal model these corruptions are part of the second corruption stage.) Let w denote the output of the corrupted parties after this simulated run of protocol FS, and let B' be the set of faulty parties in this run. Set $O(\vec{x}_{B'}, r, f_C(\vec{y})) = w$.

It remains to describe how the random input r_i is chosen, upon the corruption of a party P_i. Assume first that P_i is corrupted before the decision point. We distinguish the following cases.

Random input for the initial invocation of GShare: Here the relevant information known to the adversary consists of up to $t-1$ uniformly chosen elements in F, sent by P_i to corrupted parties. $\mathcal{S}$ chooses a random polynomial $q()$ of degree t, such that $q(0) = x_i$ (where x_i is P_i's input), and for each corrupted party P_j, the value of $q()$ at point j equals the value sent by P_i to P_j. We note that $q()$ can be chosen efficiently. Furthermore, $q()$ has the distribution of a semi-random polynomial of degree t with $q(0) = x_i$. (This is so since the values sent by the simulated P_i to corrupted parties were uniformly chosen from F.) $\mathcal{S}$ sets the random input of P_i for the initial invocation of GShare so that the polynomial chosen in Step 1(a) of GShare (See Figure 4-4) is $q()$.

We remark that the messages received by P_i from other parties during this invocation of GShare are irrelevant to P_i's random choices.

Random input for an invocation of MULT: The relevant information gathered by $\mathcal{A}$ on P_i in an invocation of MULT is as follows.

- P_i's shares of the input lines of the corresponding gate. These shares are determined by the simulated invocations of the procedures for evaluating the input lines of this gate.
- Up to $t-1$ shares of the degree-$2t$ polynomial chosen by P_i in the invocation of GShare in the Randomization Step (Step 1 of MULT, see Figure 4-7). These are the shares sent by P_i to corrupted parties.
- P_i's shares of the polynomials shared by other parties in this invocation of GShare. ($\mathcal{S}$ hands $\mathcal{A}$ these uniformly chosen shares upon the corruption of P_i, as described in Step 2 above.)
- Up to $t-1$ shares of the degree-t polynomial chosen by P_i in the Degree-Reduction Step (Step 1 of MAT, see Figure 4-6). These are the shares sent by P_i to corrupted parties.

Let the **fingerprint** of an invocation of MULT within P_i consist of this information seen by $\mathcal{A}$. Each fingerprint is consistent with an equal number of random inputs of this invocation of MULT (i.e., with an equal number of polynomials chosen by P_i in Steps 1 of MULT and MAT). This can be seen in a similar way to Lemma 4.16: Each fingerprint can be completed, in the same number of ways, to a sequence that uniquely determines the polynomials chosen by P_i. Furthermore, as remarked in the proof of Lemma 4.16, sampling these polynomials with the appropriate probability, given a fingerprint, can be done efficiently.

If P_i is corrupted after the decision point, then it may be the case that P_i has already started the final Reconstruct, and has sent $p'(i)$ to corrupted parties. In this case, P_i's simulated random input should be consistent with P_i having $p'(i)$ for its share of the output line of the circuit. Such random input can be efficiently sampled, in a similar way to the previous case (i.e., the case where P_i is corrupted before the decision point.)

We note that randomness is used also in the various invocations of Consensus. $\mathcal{S}$ uses, and hands $\mathcal{A}$ upon corruption of P_i, uniformly chosen random inputs for the use of P_i in the simulated invocations of Consensus.

Validity of the ideal-model adversary. The validity of the simulation is shown in a similar way to the non-adaptive case, with the exception that an additional type of event is added to the adversary view of the computation: corruption of some party. Consequently, the case of corruption of a party has to be considered in the inductive proof of Lemma 4.14. In Lemma 4.17 we state the inductive claim of Lemma 4.14, for the case of a corruption event. A full proof of this lemma is omitted.

Lemma 4.17 *Consider n parties running protocol* FS *on some input $\vec{x}$. Let μ_i consist of the first i events in some instance μ of the adversary view of this computation, such that the $(i+1)$th event in μ is the corruption of party P_l. Let $c^{(l)}$ (resp., $c'^{(l)}$) denote the random variable having the distribution of the information gathered by the adversary in the $(i+1)$th event in an authentic (resp., simulated) computation, given that the adversary view up to the ith event is μ_i. Then, $c^{(l)}$ and $c'^{(l)}$ are equally distributed.* □

4.4 Asynchronous verifiable secret sharing

We define Verifiable Secret Sharing in an asynchronous setting (AVSS), and describe an AVSS scheme. Our AVSS scheme is resilient against t-limited (Byzantine) adversaries in a network of n parties, as long as $n \geq 4t + 1$.

Our construction uses ideas appearing in [BGW, FM, Fe]. In particular, [Fe] and [CR] describe different AVSS schemes, for $n \geq 4t + 1$ and $n \geq 3t + 1$, respectively (the [CR] scheme is presented in Chapter 5). However, in those schemes the parties have a small probability of error in reconstructing the secret (and in [CR] also a small probability of not terminating), whereas our scheme has no probability of error.

We describe our AVSS scheme as a preamble for our construction for Byzantine adversaries. In our construction we do not use the AVSS scheme as such: in an AVSS scheme, the dealer shares a secret among the parties; later, the parties reconstruct the secret from its shares. In our protocol, the parties' shares of the secret are further processed, and the reconstructed secret is different than the shared secret (much as in the Fail-Stop case). Nevertheless, for clarity and completeness, we first define AVSS and describe a stand-alone AVSS scheme; the components of this scheme will be used in the Byzantine protocol.

In Section 4.4.1 we define AVSS. In Sections 4.4.2 through 4.4.4 we describe our AVSS scheme. In Section 4.4.5 we prove the correctness of our scheme.

4.4.1 A definition

A Verifiable Secret Sharing scheme (either synchronous or asynchronous) consists of two protocols: a sharing protocol, in which a dealer shares a secret among the other parties, and a reconstruction protocol, in which the parties reconstruct the secret from its shares. An AVSS scheme should have the following properties: first, a uncorrupted dealer should be able to share a secret in a reconstructible way. Furthermore, if *one* uncorrupted party accepts a sharing of a secret, then *all* the uncorrupted parties accept this sharing, and a unique secret will be reconstructed (even if the dealer is corrupted). Finally, if the dealer is uncorrupted, then the shared secret should remain unknown to the corrupted parties, as long as no uncorrupted party has started the reconstruction protocol. The following definition formalizes these requirements in our asynchronous setting.

Definition 4.18 *Let (S, R) be a pair of protocols such that each uncorrupted party that completes protocol S subsequently invokes protocol R with its local output of protocol S as local input. We say that (S, R) is a* t-resilient AVSS scheme *for n parties if the following holds, for every t-limited adversary.*

- **Termination.** *1. If the dealer is uncorrupted, then every uncorrupted player will eventually complete protocol S.*

  2. If some uncorrupted party has completed protocol S, then all the uncorrupted parties will eventually complete protocol S.

  3. If a uncorrupted party has completed protocol S, then it will complete protocol R.

- **Correctness.** *Once the first uncorrupted party has completed protocol S, a* unique *value, r, is fixed, such that:*

  1. r is each uncorrupted party's output of prtocol R.

  2. If the dealer is uncorrupted, sharing a secret s, then $r = s$.

- **Secrecy.** *If the dealer is uncorrupted and as long as no uncorrupted party has invoked protocol R, then the adversary view of the computation is distributed independently of the shared secret.*

Remark. We stress that a uncorrupted party is not required to complete protocol S in case that the dealer is corrupted. (We do not distinguish between the case where a uncorrupted party did not complete protocol S, and the case where a uncorrupted party has completed S unsuccessfully.)

4.4.2 An AVSS scheme

We present an outline of our scheme. Roughly speaking, the sharing protocol (denoted V-Share) consists of three stages: first, each party waits to receive its share of the secret from the dealer. Next, the parties jointly try to verify that their shares define a unique secret. Once a party is convinced that a unique secret is defined, it locally computes and outputs a 'corrected share' of the secret (using the information gathered in the verification stage).

Our AVSS scheme has the following additional property. There exists a polynomial $p(\cdot)$ of degree t such that each uncorrupted party P_i's output of protocol V-Share is $p(i)$, and $p(0)$ is the shared secret. This property is at the heart of our construction for Byzantine adversaries.

In the reconstruction protocol (denoted V-Recon), each party sends its share to the parties in some predefined set R (the set R is an external parameter with the same role as in the Fail-Stop case). Next, each party in R waits to receive enough shares to uniquely determine a secret, and outputs the reconstructed secret. In order to deal with possibly erroneous shares, we use a procedure for error correcting of Generalized Reed-Solomon (GRS) codes in an 'on-line' fashion. This procedure is presented in Section 4.4.4 below.

We turn to describing the V-Share protocol in more detail. To share a secret, s, the dealer chooses at random a polynomial $h(\cdot,\cdot)$ of degree t in two variables such that $h(0,0) = s$,[14] and sends each party P_i the t-degree polynomials $f_i(\cdot) \triangleq h(i,\cdot)$ and $g_i(\cdot) \triangleq h(\cdot,i)$. Each party P_i now sends a verification message $v_{i,j} = f_i(j)$ to each party P_j. If the verification message sent from P_i to P_j is correct, then party P_j has $v_{i,j} = f_i(j) = h(i,j) = g_j(i)$; in this case P_j 'confirms party P_i' by Broadcasting (OK, j, i). Party P_i accepts the shared secret when it finds a large enough set of parties that have confirmed each other in a 'dense' enough manner, described below.

We describe each party's view of the 'OK' Broadcasts in terms of a graph. Namely, let the OK_i graph be the (undirected) graph over the nodes $[n]$, where an edge (j,k) exists if party P_i completed both the (OK, j, k) Broadcast (initiated by P_j), and the (OK, k, j) Broadcast (initiated by P_j). Define an (n,t)-star in a graph:[15]

Definition 4.19 *Let G be a graph over the nodes $[n]$. We say that a pair (C,D) of sets such that $C \subseteq D \subseteq [n]$ is an (n,t)-star in G, if the following hold:*

- $|C| \geq n - 2t$
- $|D| \geq n - t$
- *for every $j \in C$ and every $k \in D$ the edge (j,k) exists in G.*

In the sequel, we use star to shorthand (n,t)-star. Party P_i accepts a shared secret when it finds a star in its OK_i graph. Lemma 4.26 on page 82 below implies that, provided that $n \geq 4t+1$, the shares of the uncorrupted parties in a star in the OK_i graph define a unique polynomial of degree t in two variables; Lemma 4.27 implies that the polynomials defined by the stars of every two parties are equal. Thus, once a uncorrupted party finds a star, a unique secret is defined.

Remark: conceptually, we could have let a party wait to have a clique of size $n-t$ (instead of a star) in its OK graph, before accepting a shared secret: if the dealer is uncorrupted, then the OK graph will eventually contain such a clique. However, finding a maximum size clique in a graph is an NP-complete problem. Instead, the party will try to find a star. In Section 4.4.3 we describe an efficient procedure for finding a (n,t)-star in a graph, provided that the graph contains a clique of size $n-t$.

We want to make sure that if a uncorrupted party finds a star in its OK graph, then all the uncorrupted parties will find a star, even when the dealer is corrupted and the OK graph does not contain a clique. For this purpose, upon finding a star, the party sends it to all the other parties. Upon receiving an $(\mathrm{OK},\cdot,\cdot)$ Broadcast, a party that has not found a star checks whether any of the suggested stars is indeed a star in its OK graph. Note that an edge in the OK graph of a uncorrupted party will eventually be an edge in the OK graph of every uncorrupted party (since all the $(\mathrm{OK},\cdot,\cdot)$ messages are Broadcasted); thus, a star in the OK graph of some uncorrupted party will eventually be a star in the OK graph of every other uncorrupted party.

[14] That is, $h(x,y) = \sum_{i=0}^{t}\sum_{j=0}^{t} h_{i,j}x^iy^j$, where $h_{0,0} = s$, and all the other coefficients $h_{0,1},\ldots,h_{t,t}$ are chosen uniformly and independently over F.

[15] Our definition of an (n,t)-star is different than the "standard" definition.

Party P_i's output of protocol V-Share will be $h_i(0,i)$, where $h_i(\cdot,\cdot)$ is the (unique) polynomial defined by the star found by P_i. If P_i is a member of its own star, then the polynomial $g_i(\cdot)$ that P_i received from the dealer satisfies $g_i(\cdot) = h_i(\cdot, i)$, and P_i outputs $g_i(0)$. If P_i is not a member of its own star, then $g_i(\cdot)$ may be erroneous (possibly, P_i didn't even receive $g_i(\cdot)$.) Therefore, upon finding a star (C_i, D_i), and if $P_i \notin D_i$, then P_i computes $h_i(0,i)$ using the verification messages $v_{j,i}$ it has received (or still expects to receive) from the parties $P_j \in D_i$. (For the $2t+1$ *uncorrupted* parties $P_j \in D_i$, we have $v_{j,i} = h_i(j,i)$. Therefore, the values received from the parties in D_i uniquely determine the polynomial $h_i(\cdot, i)$.) In Section 4.4.4 we describe how this computation is carried out.

The code of the V-Share and V-Recon protocols is described in Figures 4-9 and 4-10, respectively.

protocol V-Share

<u>Code for the dealer (on input s):</u>

1. choose a random polynomial $h(\cdot,\cdot)$ of degree t in two variables, such that $h(0,0) = s$. For every $1 \le i \le n$, send the polynomials $f_i(\cdot) = h(\cdot, i)$ and $g_i(\cdot) = h(i, \cdot)$ to party P_i.

<u>Code for party P_i:</u>

2. Upon receiving $f_i(\cdot)$ and $g_i(\cdot)$ from the dealer, and for each $1 \le j \le n$, send $f_i(j)$ to party P_j.
3. Upon receiving $v_{i,j}$ from party P_j: if $v_{i,j} = g_i(j)$, then broadcast (OK, i, j).
4. Upon receiving a broadcast (OK, j, k), check for the existence of a star in OK_i, using procedure STAR described in Section 4.4.3 below. If a star (C_i, D_i) is found, go to Step 6 and send (C_i, D_i) to all the parties.
5. Upon receiving a message (C_j, D_j) add (C_j, D_j) to the set of 'suggested stars'. As long as a star is not yet found, then whenever an (OK, k, l) Broadcast is received, check whether (C_j, D_j) form a star in the OK_i graph.
6. Upon finding a star (C_i, D_i), and if $i \notin D_i$, correct polynomial $g_i(\cdot)$, based on the verification messages received from the parties in D_i, and using the error correcting procedure OEC described in Section 4.4.4.

   Namely: let $\mathcal{V}_i = \{(j, v_{i,j}) | j \in D_i\}$; set $g_i(\cdot) = \mathrm{OEC}[t,t](\mathcal{V}_i)$.
7. Once $g_i(\cdot)$ is corrected, (locally) output $g_i(0)$.

Figure 4-9: V-Share - The verifiable sharing protocol

4.4.3 Efficiently finding a star

We describe an efficient procedure for finding a (n,t)-star in a graph of n nodes (see Definition 4.19 on page 75), provided that the graph contains a clique of size $n-t$. Namely, our procedure outputs either a star in the graph, or a message 'star not found'; whenever the input graph contains a clique of size $n-t$, then the procedure outputs a star in the graph.

We follow an idea of Gabril, appearing in [GJ] p. 134. There, the following approxi-

Protocol V-Recon$[R](a_i)$

Code for party P_i (on input a_i, and with parameter $R \subseteq \{P_1, \ldots, P_n\}$):

7. Send a_i to the parties in R.
8. Let $\mathcal{S}_i = \{(j, a_j) |\ a_j$ has been received from $P_j\}$.
   If $P_i \in R$, terminate without output.
   Otherwise, set $z_i(\cdot) = \mathrm{OEC}[t,t](\mathcal{S}_i)$, and output $z_i(0)$.

Figure 4-10: V-Recon - The verifiable reconstruction protocol

mation algorithm is described: if the input graph contains a clique of size $n-k$, then a clique of size $n-2k$ is found. The algorithm is simple: find a **maximal matching**[16] in the **complementary graph**[17] and output the set of unmatched nodes. Clearly, the output is an independent set in the complementary graph; thus, it forms a clique in the original graph. Furthermore, if a graph contains a clique of size $n-k$, then any maximal matching in the complementary graph involves at most k edges and $2k$ nodes.

We first restate our problem in terms of the complementary graph: if the input graph has an independent set of size $n-t$, find sets $C \subseteq D$ of nodes, such that $|C| \geq n-2t$, $|D| \geq n-t$, and no edges exist between nodes in C and nodes in $C \cup D$. We call such a pair of sets an (n,t)-$\overline{\text{star}}$. In the rest of this section, we refer to the complementary graph only.

In our procedure, we find a *maximum* matching in the graph (say, using [Ed] or [MV]). [18] Based on this matching, we compute sets C, D of nodes, and check whether (C, D) form an (n,t)-$\overline{\text{star}}$ in the graph. If the input graph contains an independent set of size $n-t$, then the computed sets C, D form an (n,t)-$\overline{\text{star}}$ in the input graph.

We describe how sets C and D are computed. Consider a matching; we say that a node is a **triangle-head** if it is unmatched, and two of its neighbours are a matched pair (namely, the edge between these two neighbours is in the matching). Let C denote the set of unmatched nodes that are not triangle-heads. Let B be the set of matched nodes that have neighbours in C, and let $D \triangleq [n] - B$.

Our procedure is described in Figure 4-11. Figure 4-12 illustrates the relations among the different subsets of nodes.

Proposition 4.20 *Assume procedure* STAR *outputs* (C, D)*, on input graph* G*. Then,* (C, D) *form a* t-$\overline{\text{star}}$ *in* G*.*

Proof: Clearly, if algorithm STAR outputs (C, D) then $|C| \geq n-2t$ and $|D| \geq n-t$, and $C \subseteq D$. We show that for every $i \in C$ and every $j \in D$, the nodes i and j are not neighbours in G.

[16] A matching in a graph is a set M of edges such that no two edges in M have a common endpoint. A matching is maximal if every edge added to it has a common endpoint with an edge in the matching.

[17] The complementary graph is the graph in which an edge exists iff it does not exist in the original graph.

[18] In fact, we only need that the matching cannot be improved by augmenting paths of length at most 3.

Procedure STAR$[t](G)$

input: an undirected graph G (over the nodes $[n]$), a parameter t.
output: a t-$\overline{\text{star}}$ in the graph G, or a message: 'star not found'.

1. Find a maximum matching M in G.
   Let N be the set of matched nodes (namely, the endpoints of the edges in M), and let $\bar{N} \triangleq [n] - N$.
2. Verify that the matching, M, has property $\mathcal{P}$:
   (a) Let T be the set of triangle-heads; namely, set
   $T \triangleq \{i \in \bar{N} | \ \exists j, k \text{ s.t } (j, k) \in M \text{ and } (i, j), (i, k) \in G\}$.
   Let $C \triangleq \bar{N} - T$.
   (b) Let B be the set of matched nodes that have neighbours in C;
   namely, let $B \triangleq \{j \in N | \ \exists i \in C \text{ s.t. } (i, j) \in G\}$.
   Let $D \triangleq [n] - B$.
   (c) If $|C| \geq n - 2t$ and $|D| \geq n - t$, output (C, D). Otherwise, output 'star not found'.

Figure 4-11: STAR - A procedure for finding a $\overline{\text{star}}$ in a graph

Figure 4-12: Partition of the graph G

Assume that $i \in C$ and $j \in D$, and that (i,j) is an edge in G. As $j \in D$, we must have $j \notin B$. By The definition of B, we have $j \notin N$ (if $i \in C$ and $j \in N$, then $j \in B$). Furthermore, $i \in C \subseteq \bar{N}$. Thus, both i and j are unmatched. Consequently, the edge (i,j) can be added to the matching to create a larger matching, and the matching is not maximum (in this case, it is not even maximal). □

Proposition 4.21 *Let G be a graph over n nodes containing an independent set of size $n-t$. Then procedure* STAR *outputs a t-$\overline{\text{star}}$ in G.*

Proof: We show that if the input graph G contains an independent set of size $n-t$, then the sets C and D determined in Steps 2(a) and 2(b) of procedure STAR are large enough (i.e., $|C| \geq 2t+1$ and $|D| \geq n-t$). Consequently, the procedure outputs (C,D) in Step 4; Proposition 4.20 assures us that (C,D) form a star in G.

First, we show that $|C| \geq n-2t$. Let $I \subseteq [n]$ be an independent set of size $n-t$ in G, and let $\bar{I} = [n] - I$. We adopt the definitions of N, T, and C in the procedure (see Figure 4-12).

Let $F \triangleq I - C$. We show below that $|F| \leq |\bar{I}|$. However, $|\bar{I}| \leq t$. Consequently, we have $|C| \geq |I| - |F| \geq n-2t$.

To see that $|F| \leq |\bar{I}|$, we show a one-to-one correspondence $\phi : F \rightarrow \bar{I}$. Let $i \in F$; since $i \notin C$, we have either $i \in N$ or $i \in T$.

Case 1: $i \in N$. Then, let $\phi(i)$ be the node matched to i in M. Clearly, $\phi(i) \in \bar{I}$: otherwise, we had an edge $(i, \phi(i))$ where both i and $\phi(i)$ are in an independent set.

Case 2: $i \in T$. By the definition of T, node i has two neighbours j, k such that $(j,k) \in M$. Arbitrarily set $\phi(i) = j$. Clearly, both j and k are in $\bar{I}$.

Wo show that ϕ is one-to-one. Consider two distinct nodes $l, m \in F$; we distinguish three case:

Case 1: $l, m \in N$. In this case, $\phi(l) \neq \phi(m)$ since M is a matching.

Case 2: $l \in N$ and $m \in T$. Since $m \in T$, there exists an edge between m and the node matched to $\phi(m)$. Since $l \in N$, the node matched to $\phi(l)$ is l. Now, assume that $\phi(l) = \phi(m)$. Thus, (l,m) is an edge in G. However, both l and m are in the independent set I: a contradiction.

Case 3: $l, m \in T$. Assume $\phi(l) = \phi(m)$. Let a be the node matched to $\phi(m)$ in M; then, both l and m are neighbours of both $\phi(m)$ and a. However, in this case the matching M is not maximum since, for instance, $M - \{(\phi(m), a)\} \cup \{(\phi(m), l), (a, m)\}$ is a larger matching.

It remains to show that $|D| \geq n-t$. Recall that $D = [n] - B$. We show below that $|B| \leq |M|$; since G contains an independent set of size $n-t$, we have $|M| \leq t$. Thus, $|D| = n - |B| \geq n - |M| \geq n-t$.

To see that $|B| \leq |M|$, we show that at most one of the endpoints of every edge $(a,b) \in M$ is in B. Suppose on the contrary that both a and b have neighbours in C, and let $c, d \in C$ be the neighbours of a and b, respectively. Surely, $c \neq d$ (otherwise, c was a triangle-head, and we had $c \notin C$). However, in this case the matching M is not maximum, since, for instance, $M - \{(a,b)\} \cup \{(a,c),(b,d)\}$ is a larger matching. □

4.4.4 On-line error correcting

Consider the following scenario. A party expects to receive messages from m parties (message a_j from party P_j), so that there exists a polynomial $p(\cdot)$ of degree d, satisfying $p(j) = a_j$ for every message a_j. The messages arrive one by one, in some arbitrary order; furthermore, up to t of the messages may be wrong or missing. (In our context, we have $m = n - t$ and $d = t$.) We describe an efficient procedure that enables the party to compute this polynomial 'on-line'; namely, the party will recognize when the received messages define a unique 'interpolated polynomial' of degree d, and compute this polynomial. The following definitions provide tools for precisely stating this problem.

Definition 4.22 *Let δ and ρ be integers, and let $S \subseteq [n] \times F$ such that for every two elements (i, a) and (i', a') in S we have $i \neq i'$. We say that S is (δ, ρ)-*interpolated *if there exists a polynomial $p(\cdot)$ of degree δ, so that at most ρ elements $(i, a) \in S$ do* not *satisfy $p(a) = e$. We say that $p(\cdot)$ is a (δ, ρ)-*interpolated polynomial *of S.*

(Remark: A d-interpolated vector, defined in Section 4.3.3 on page 61, is a different formulation of a $(d, 0)$-interpolated set.) Note that the (δ, ρ)-interpolated polynomial of a (δ, ρ)-interpolated set S is unique, provided that $|S| \geq \delta + 2\rho + 1$. (Proof: assume S has two (δ, ρ)-interpolated polynomials $p(\cdot), q(\cdot)$. Then, for at least $|S| - 2\rho$ elements $(a, e) \in S$ we have $p(a) = e = q(a)$. However, $|S| - 2\rho \geq \delta + 1$; thus, $p(\cdot) = q(\cdot)$.)

Definition 4.23 *Let $\mathcal{I}$ be an accumulative set* [19]*. We say that $\mathcal{I}$ is* eventually (δ, τ)-interpolated*, if for every τ-limited adversary, and every run, there exists an integer $0 \leq \rho \leq \tau$, such that $\mathcal{I}$ will eventually hold a (δ, ρ)-interpolated set of size at least $\delta + \tau + \rho + 1$.*

Let $\mathcal{I}$ be an eventually (δ, τ)-interpolated accumulative set. Using a similar argument to the one used above, it can be seen that all the (δ, ρ)-interpolated sets of size at least $\delta + \tau + \rho + 1$ held in $\mathcal{I}$ have the same interpolated polynomial. We call this polynomial the (δ, τ)-interpolated polynomial of $\mathcal{I}$.

Using these notations, the (dynamic) input of the procedure described in this section is an eventually (d, t)-interpolated accumulative set $\mathcal{I}$. The required output of this procedure is the (d, t)-interpolated polynomial of $\mathcal{I}$. (Definition 4.23 assures us that at least $d + t + 1$ values in $\mathcal{I}$ will 'sit on a polynomial' of degree t. At least $t + 1$ of thus values originate with uncorrupted parties. Consequently, the (d, t)-interpolated polynomial of $\mathcal{I}$ is bound to be the 'correct' polynomial, namely the polynomial defined by the values of the uncorrupted parties.)

Our procedure, denoted OEC (for On-line Error Correcting), consists of up to t iterations. In iteration r, the party waits until the accumulative set $\mathcal{I}$ is of size at least $d + t + r + 1$; then, the party uses a procedure, described below, that determines whether this set is (d, r)-interpolated, and computes the corresponding interpolated polynomial. If a (d, r)-interpolated polynomial is found, then we output this polynomial and terminate. Otherwise, we proceed to iteration $r + 1$ (since $\mathcal{I}$ is (d, t)-eventually interpolated, it is bound to have at least one more element; thus, iteration $r + 1$ will be completed).

[19] Accumulative sets are defined in Section 4.1.3 on page 53.

It is left to describe how to determine if a given set is (d,r)-interpolated, and how to compute the interpolated polynomial. We use a result from Coding theory. Consider the following code: a word $W = (i_1, a_1)\dots(i_l, a_l)$ is a code-word iff there exists a polynomial $p(\cdot)$ of degree d such that $p(i_j) = a_j$ for every $1 \le j \le l$. This is a Generalized Reed-Solomon (GRS) code; GRS codes have an efficient error correcting procedure that detects and corrects up to r errors in an input word, W, provided that $|W| \ge d + 2r + 1$ (see, for instance, [MS pp. 294-315]). Let EC denote such a procedure. Namely, let S be a (d,r)-interpolated set of size at least $d + 2r + 1$; then, procedure EC, on input (d, r, S), outputs the (d,r)-interpolated polynomial of S.

Procedure OEC is described in Figure 4-13.

Procedure OEC$[d,t](\mathcal{I})$

For $0 \le r \le t$ do:

1. Let I_r denote the contents of accumulative set $\mathcal{I}$, when $\mathcal{I}$ contains $d + t + r + 1$ elements.

   Wait until $|\mathcal{I}| \ge d + t + r + 1$. Then, run EC$(d, r, I_r)$, and let $p(\cdot)$ be the output polynomial.

   If $p(\cdot)$ is a (d,r)-interpolated polynomial of I_r (namely, if for $d + t + 1$ elements $(i, a) \in I_r$ we have $p(i) = a$), output $p(\cdot)$. Otherwise, proceed to the next iteration.

Figure 4-13: OEC - A procedure for on-line error correcting

Proposition 4.24 *Let $\mathcal{I}$ be an eventually (d,t)-interpolated accumulative set. Then, procedure OEC$[d,t](\mathcal{I})$ halts and outputs the (d,t)-interpolated polynomial of $\mathcal{I}$.*

Proof: Let $\hat{r}$ be the smallest r such that $I_{\hat{r}}$ is $(d,\hat{r})$-interpolated; since $\mathcal{I}$ is eventually (d,t)-interpolated, we have $\hat{r} \le t$. All the iterations up to iteration $\hat{r}$ will be completed (unsuccessfully). The (d,t)-interpolated polynomial of $\mathcal{I}$ will be found in iteration $\hat{r}$. □

4.4.5 Correctness of the AVSS scheme

Theorem 4.25 *The pair (V-Share, V-Recon) is a t-resilient AVSS scheme in a network of n parties, provided that $n \ge 4t + 1$.*

Proof: We assert the Correctness, Termination and Secrecy requirements of Definition 4.18 on page 74.

Correctness. We associate with every uncorrupted party P_i that completed protocol V-Share a unique polynomial $h_i(\cdot,\cdot)$ of degree t in two variables, and show that every two uncorrupted parties P_i and P_j have $h_i(\cdot,\cdot) = h_j(\cdot,\cdot)$. Next, we show that Conditions 1 and 2 of the Correctness requirement are met, with respect to $r = h_i(0,0)$ (for some uncorrupted party P_i). Moreover, we show that party P_i's output of protocol V-Share is $h_i(i,0)$.

We use two technical lemmas.

Lemma 4.26 *Let $m \geq d+1$, and let $f_1(\cdot)\ldots f_m(\cdot)$ and $g_1(\cdot)\ldots g_m(\cdot)$ be polynomials of degree d over a field F with $|F| \geq m$, such that for every $1 \leq i \leq d+1$ and every $1 \leq j \leq m$ we have $f_i(j) = g_j(i)$ and $g_i(j) = f_j(i)$. Then, there exists a unique polynomial $h(\cdot,\cdot)$ of degree d in two variables so that for every $1 \leq i \leq m$ we have $h(\cdot,i) = f_i(\cdot)$ and $h(i,\cdot) = g_i(\cdot)$.*

Proof: See Appendix B. □

Lemma 4.27 *Let $h(\cdot,\cdot)$, $h'(\cdot,\cdot)$ be two polynomials of degree d in two variables over a field F with $|F| > d$, and let $v_1 \ldots v_{d+1}$ be distinct elements in F. Assume that for every $1 \leq i,j \leq d+1$ we have $h(v_i,v_j) = h'(v_i,v_j)$. Then, $h(\cdot,\cdot) = h'(\cdot,\cdot)$.*

Proof: See Appendix B. □

Let P_i be a uncorrupted party that completed protocol V-Share, and let (C_i, D_i) be the star found by P_i. Let D_i' be the set of uncorrupted parties in D_i, and let C_i' be the set of uncorrupted parties in C_i; thus, $|D_i'| \geq |D| - t \geq n - 2t$ and $|C_i'| \geq |C| - t \geq n - 3t \geq t+1$ (since $n \geq 4t+1$). Applying Lemma 4.26, we get that the polynomials $f_j(\cdot), g_j(\cdot)$ of the parties $j \in D_i'$ determine a unique polynomial of degree t in two variables. Let $h_i(\cdot,\cdot)$ denote this polynomial. (Namely, $h_i(\cdot,\cdot)$ is the polynomial associated with P_i.) Note that $h_i(\cdot,\cdot)$ is fixed once P_i has completed protocol V-Share.

For every other uncorrupted party P_j, let $I_{i,j}$ be the set of uncorrupted parties in $D_i \cap D_j$. Since $n \geq 4t+1$, we have $|D_i \cap D_j| \geq n - 2t \geq 2t+1$, thus $|I_{i,j}| \geq t+1$. For every two parties $k,l \in I_{i,j}$ we have $h_i(k,l) = v_{k,l} = h_j(k,l)$, where $v_{k,l}$ is the verification piece sent by P_k to P_l in Step 2 of protocol V-Share. Applying Lemma 4.27, we have $h_i(\cdot,\cdot) = h_j(\cdot,\cdot)$. The value r required in the Correctness condition is $r = h_i(0,0)$.

We assert Condition 1 (namely, that if the dealer is uncorrupted and has shared a value s, then $r = s$). If the dealer is uncorrupted and has chosen a polynomial $h(\cdot,\cdot)$ in Step 1, then for every two parties $P_k, P_l \in D_i'$ we have $h_i(k,l) = h(k,l)$. Applying Lemma 4.27 again, we get $h_i(\cdot,\cdot) = h(\cdot,\cdot)$. In the sequel, we omit the subscript from the polynomial $h(\cdot,\cdot)$.

Next, we show that each uncorrupted party P_i's output of protocol V-Share is $h(i,0)$. Polynomial $h(i,\cdot)$ is the (only) interpolated polynomial of P_i's accumulative set $\mathcal{V}_i$ in Step 6. Therefore, the output of the error correcting procedure OEC in Step 6 will be $h(i,\cdot)$, and the output of protocol V-Share will be $h(i,0)$.

It remains to assert Condition 2 (namely, that P_i's output of V-Recon is r). Every uncorrupted party P_j will Broadcast $h(j,0)$ in Step 7; thus, $h(\cdot,0)$ is the (only) interpolated polynomial of P_i's accumulative set $\mathcal{S}_i$ in Step 8. Therefore, the output of the error correcting procedure OEC in Step 8 will be $h(\cdot,0)$, and the output of protocol V-Recon will be $h(0,0) = r$.

Termination. Condition 1. If the dealer is uncorrupted, then for every two uncorrupted parties P_j and P_k, both (OK, j, k) and (OK, k, j) will be broadcasted, since $f_j(k) = h(k,j) = g_k(j)$ and $g_j(k) = h(j,k) = f_k(j)$. Thus, every uncorrupted party P_i will eventually have a clique of size $n - t$ in its OK_i graph. Therefore, procedure STAR will find a star in OK_i and Step 4 will be completed. Step 6 will be completed since the input of procedure OEC (namely, the accumulative set $\mathcal{V}_i$ which is based on the star found in Steps 4 or 5) is eventually (t,t)-interpolated.

Condition 2. Let P_i be a uncorrupted party that completed protocol V-Share, and let (C_i, D_i) be the star found by P_i. Then, (C_i, D_i) will eventually be a star in the OK_j

graph of every uncorrupted party P_j, unless P_j has already completed protocol V-Share. Furthermore, party P_j will receive the (C_i, D_i) message (sent by P_i in Step 4), and will verify, in Step 5, that the sets (C_i, D_i) form a star in OK_j. Upon finding a star, P_j will execute Step 6 and complete protocol V-Share.

Condition 3. If all the uncorrupted parties have started protocol V-Recon, then the accumulative set $\mathcal{S}_i$ of Step 8 of each uncorrupted party P_i is eventually (t,t)-interpolated. Thus, all the uncorrupted parties will complete procedures OEC and V-Recon.

Secrecy. We use the following notations.

- For a value v, let $\mathcal{H}_v$ denote the set of polynomials of degree t in two variables, with free coefficient v.
- We say that a sequence $f_1(\cdot), \ldots, f_t(\cdot), g_1(\cdot), \ldots, g_t(\cdot)$ of polynomials is interleaved if for every $1 \leq i, j \leq t$ we have $f_i(j) = g_j(i)$. Let I denote the set of interleaved sequences of $2t$ polynomials of degree t.

Lemma 4.28 *Let F be a field with $|F| > d$, and let $s \in F$. Then, for every interleaved sequence $f_1(\cdot), \ldots, f_d(\cdot), g_1(\cdot), \ldots, g_d(\cdot)$ in I, there exists a unique polynomial $h(\cdot,\cdot) \in \mathcal{H}_s$, so that for every $1 \leq i \leq d$ we have $h(\cdot, i) = f_i(\cdot)$ and $h(i, \cdot) = g_i(\cdot)$.*

Proof: See Appendix B. □

Assume a uncorrupted dealer, and let s be the shared value. Then, the dealer has chosen, in Step 1 of protocol V-Share, a polynomial $h(\cdot,\cdot)$ with uniform distribution over $\mathcal{H}_s$. Furthermore, all the relevant information a set of t parties received during an execution of protocol V-Share, is an interleaved sequence $f_1(\cdot), \ldots, f_t(\cdot), g_1(\cdot), \ldots, g_t(\cdot)$ in I, so that for every $1 \leq i \leq t$ we have $h(\cdot, i) = f_i(\cdot)$ and $h(i, \cdot) = g_i(\cdot)$.

Lemma 4.28 implies that for every shared value $s \in F$, this correspondence between polynomials in $\mathcal{H}_s$ and interleaved sequences in I is one to one and onto. Therefore, a uniform distribution over the polynomials in $\mathcal{H}_s$ induces a uniform distribution over the interleaved sequences in I.

Thus, all the corrupted parties have after executing protocol V-Share is a sequence of interleaved polynomials of degree t, chosen with uniform distribution over I, regardless of the shared value. □

4.5 Byzantine adversaries

As in the Fail-Stop case, let the computed function be $f : F^n \to F$, and assume that the parties have an arithmetic circuit computing f. We describe an n party protocol for securely t-computing f in an asynchronous network with arbitrary (i.e., Byzantine) adversaries, provided that $n \geq 4t + 1$.

We follow the outline of the Fail-Stop protocol, modifying its components to the Byzantine setting. First, we extend Shamir's Secret Sharing scheme (used for Fail-Stop adversaries) to an Asynchronous Verifiable Secret Sharing (AVSS) scheme. Next, The multiplication step is adapted to a Byzantine setting.

In our protocol we do not use the AVSS scheme as such: in an AVSS scheme, the dealer shares a secret among the parties; later, the parties reconstruct the secret from its shares. In our protocol, the parties' shares of the secret are further processed, and the reconstructed secret is different than the shared secret (much as in the Fail-Stop case). Nevertheless, for clarity and completeness, we first define AVSS and describe a stand-alone AVSS scheme; the components of this scheme will be used in the Byzantine protocol.

Our AVSS scheme, as well as the multiplication step protocol, makes use of error correcting techniques for Generalized Reed-Solomon (GRS) codes. In the AVSS scheme we describe a procedure, run locally by each party, that enables the party to locate and correct, in an 'on-line' fashion, erroneous or missing messages in a sequence of received messages. In the multiplication step, a GRS code-word is generated, so that each party holds a piece of this code-word. Each party shares its piece, using AVSS; then, the parties agree, without learning further information, on a set of parties whose shared pieces are indeed the pieces of the original code-word.

4.5.1 Global Verifiable Share

The Global Verifiable Share (GV-Share) protocol, described in Figure 4-14, is the Byzantine counterpart of the Fail-Stop GShare protocol (described in Figure 4-4 on page 60). It consists of two phases: first, each party shares its input, using the V-Share protocol of the AVSS scheme (Figure 4-9 on page 76); next, the parties use protocol ACS (Section 4.2.3 on page 56) to agree on a set C of at least $n-t$ parties who properly shared their inputs. Party P_i's output of protocol GV-Share is this set C, and the ith share of each secret shared by a party in C. (Recall that protocol GShare had an additional security parameter, d; here, the security parameter is fixed to $d = t$.)

Protocol GV-Share(x_i)

Code for Party P_i, on input x_i:

1. Initiate V-Share$_i(x_i)$ (with P_i as the dealer).
   For $1 \leq j \leq n$, participate in V-Share$_j$.
   Let v_j be the output of V-Share$_j$.
2. Let $\mathcal{U}_i = \{j \mid$ V-Share$_j$ has been completed$\}$. Set $C =$ACS$[n-t, n](\mathcal{U}_i)$.
3. Once the set C is computed, output $(C, \{v_j \mid j \in C\})$.

Figure 4-14: GV-Share - The global verifiable sharing protocol

4.5.2 Computing a multiplication gate

Let $c = a \cdot b$ be a multiplication gate, and let $A(\cdot), B(\cdot)$ be the polynomials associated with the input lines. Namely, each uncorrupted party P_i's shares of these lines are $A(i)$ and $B(i)$ respectively, and $A(0) = a$ and $B(0) = b$. As in the Fail-Stop case, the parties will jointly compute their shares of a random polynomial $C(\cdot)$ of degree t with $C(0) = A(0) \cdot B(0)$, so that each uncorrupted party P_i's share of the output line will be $C(i)$.

The Byzantine multiplication procedure follows the outline of its Fail-Stop counterpart. Namely, the parties first generate a random polynomial $D(\cdot)$ of degree $2t$ with free coefficient

$D(0) = A(0) \cdot B(0)$. Then, the parties compute their shares of the truncation polynomial of $D(\cdot)$ to degree t; this truncation polynomial is the output polynomial $C(\cdot)$.

We proceed to describe the Byzantine implementations of these two steps.

Randomization

The Byzantine randomization step follows the outline of its Fail-Stop counterpart. Namely, each party P_i first shares (in a way described below) a random polynomial $H_i(\cdot)$ of degree $2t$ with $H_i(0) = 0$. Next, the parties use protocol ACS to agree on a set C of parties that have successfully shared their polynomial. Finally, each party P_i locally computes $H(i) = \sum_{j \in C} H_j(i)$, and $D(i) = A(i) \cdot B(i) + H(i)$.

It remains to describe how each party P_i shares its polynomial $H_i(\cdot)$. We use the [BGW] method. This method 'has the effect' of sharing polynomial $H_i(\cdot)$ in the 'straightforward' way. Namely, on one hand, each party P_j will have $H_i(j)$; on the other hand, the information gathered by the corrupted parties will be 'equivalent' to each corrupted party knowing only its share of $H_i(\cdot)$. Consequently, the information gathered by the corrupted parties *in the entire multiplication step* will be independent of the computed value (i.e., $A(0) \cdot B(0)$).

We describe this sharing method. Each party P_i shares t uniformly chosen values, using t invocations of V-Share. Let $z_{i,j,k}$ be party P_k's output of the jth invocation of V-Share where P_i is the dealer. Upon completing all the t invocations of V-Share, each party P_k locally computes $H_i(k) = \sum_{j=1}^{t} k^j \cdot z_{i,j,k}$.

Let us reason this slightly unintuitive sharing method (for a formal proof, see the proof of Theorem 4.30 on page 90). Let $S_{i,j}(\cdot)$ be the polynomial of degree t defined by the jth V-Share initiated by P_i (namely, $S_{i,j}(k) = z_{i,j,k}$ for every uncorrupted party P_k). Polynomial $H_i(\cdot)$ is now defined as $H_i(x) = \sum_{j=1}^{t} x^j \cdot S_{i,j}(x)$: each party P_k locally computes $H_i(k) = \sum_{j=1}^{t} k^j \cdot S_{i,j}(k) = \sum_{j=1}^{t} k^j \cdot z_{i,j,k}$. Let $s_{i,j,l}$ be the coefficient of x^l in $S_{i,j}(x)$; it might be helpful to visualize

$$
\begin{array}{l}
H_i(x) = \\
\begin{array}{ccccccccccccccc}
s_{i,1,0}x & + & s_{i,1,1}x^2 & + & \ldots & + & s_{i,1,t-1}x^t & + & s_{i,1,t}x^{t+1} & + & & & & & \\
 & & s_{i,2,0}x^2 & + & \ldots & + & s_{1,2,t-2}x^t & + & s_{i,2,t-1}x^{t+1} & + & s_{i,2,t}x^{t+2} & + & & & \\
 & & & & \ldots & & & & & & & & & & \\
 & & & & & & s_{i,t,0}x^t & + & s_{i,t,1}x^{t+1} & + & & & \ldots & + & s_{i,t,t}x^{2t}
\end{array}
\end{array}
$$

The free coefficient of $H_i(\cdot)$ is 0; thus, $H_i(0) = 0$. Each polynomial $S_{i,j}(\cdot)$ is of degree t; thus, $H_i(x)$ is of degree $2t$. Furthermore, it can be seen that the coefficients of the monomials $x, \ldots, x^t$ in $H_i(x)$ (and, thus, the same coefficients of the sum polynomial $H(\cdot)$ and in polynomial $D(\cdot)$) are uniformly distributed over F. Consequently, the coefficients of all the non-zero powers of the truncation polynomial $C(\cdot)$) are uniformly distributed over F.

Clearly, the corrupted parties gather some extra information on top of the t shares of $H_i(\cdot)$. However, it is plausible that this information is independent of $A(0) \cdot B(0)$: party P_i chooses $t^2 + t$ random coefficients, and the corrupted parties receive only t^2 values. (In the proof of Lemma 4.31 on page 91 we show that the information gathered by the corrupted parties during the whole multiplication protocol is independent of $A(0) \cdot B(0)$.)

Degree-reduction

Next, the parties use their shares of polynomial $D(\cdot)$ in order to jointly and securely compute their shares of the 'truncation' of $D(\cdot)$ to degree t; namely, the $t+1$ coefficients of the output polynomial $C(\cdot)$ are the coefficients of the $t+1$ lower degrees of $D(\cdot)$.

In the Fail-Stop protocol, the parties computed their shares of $C(\cdot)$ by invoking a protocol for 'multiplying the inputs vector by a fixed matrix'; we shortly review this protocol. Let $\vec{d} = D(1), \ldots, D(n)$, let $\vec{c} = C(1), \ldots, C(n)$, and let M be the $n \times n$ matrix such that $\vec{d} \cdot M = \vec{c}$. First, each party P_i shared its 'input' $D(i)$; next, the parties agreed on a set of parties that have successfully shared their inputs. Once this set, G, was agreed upon, each party locally computed the appropriate matrix M^G, and the products of his shares by M^G. Finally the parties invoked n Reconstruct protocols, so that in the ith invocation of Reconstruct, party P_i computed $C(i) = \sum_{j \in G} d_j \cdot M^G_{j,i}$.

In the Byzantine setting, the parties will use V-Share and V-Recon instead of the simple sharing scheme of the Fail-Stop case. Still, a major problem remains: the agreed set G may contain (corrupted) parties P_i that have shared some value different than the expected value $D(i)$; in this case, the parties will not have the expected outputs. In the rest of this section, we describe how the parties make sure that the value associated with each party P_i in the agreed set (namely, the free coefficient of the t-degree polynomial defined by the uncorrupted parties' shares of P_i's input) is indeed $D(i)$.

For a party P_i, let s_i be the value associated with P_i (recall that s_i is fixed once the first uncorrupted party has completed P_i's V-Share); for a set A of parties, let $S_A = \{(i, s_i) | P_i \in A\}$. We first note that it is enough to agree on a set G of at least $3t + 1$ parties, such that S_G is $(2t, 0)$-interpolated [20] (namely, all the values shared by the parties in G 'sit on a polynomial of degree $2t$'). This is so, since the set G, being of size $3t + 1$, contains at least $2t + 1$ *uncorrupted* parties; thus, the interpolated polynomial of S_G is bound to be $D(\cdot)$.

We describe a protocol for agreement on a set A of parties, such that S_A is $(2t, 0)$-interpolated. This protocol, denoted AIS (for Agreement on an Interpolated Set), is a 'distributed implementation' of procedure OEC (described in Section 4.4.4 on page 80). Protocol AIS consists of up to t iterations. In iteration r $(0 \le r \le t)$, the parties first use protocol ACS (Figure 4-3 on page 58) to agree on a set, G_r, of at least $3t+1+r$ parties that have successfully shares their inputs. Next, the parties perform a computation, described below, to check whether S_{G_r} is $(2t, r)$-interpolated. If S_{G_r} is $(2t, r)$-interpolated, then there exists a set $G'_r \subseteq G_r$, of size at least $3t + 1$, such that $S_{G'_r}$ is $(2t, 0)$-interpolated; the parties will compute and output G'_r. Otherwise (i.e., S_{G_r} is not $(2t, r)$-interpolated), the parties will proceed to the next iteration. We stress that the parties will not know the interpolated polynomial of each S_{G_r}. They will only know whether such a polynomial exists.

It remains to describe how to check, given a set G of size $3t+1+r$, whether S_G is $(2t, r)$-interpolated, and how to compute the corresponding set G' (i.e., $G' \subseteq G$, $|G'| \ge 3t+1$, and $S_{G'}$ is $(2t, 0)$-interpolated). As in procedure OEC, we use error correcting for Generalized Reed-Solomon codes. However, in procedure OEC, the 'word', S_G, was a (dynamic) input of one party. Thus, each party could locally run an error correcting procedure. In our setting, each party has only one share of each element of S_G; the parties will invoke a joint computation implementing a specific error correcting procedure, and use it to check whether

[20] Interpolated sets and interpolated polynomials were defined in Section 4.4.4 on page 80.

G is $(2t, r)$-interpolated and to compute G'.

Let us first outline the particular error correcting procedure we will be implementing. The inputs to this procedure are (d, r, W). If $|W| \geq d + 2r + 1$ and W is (d, r)-interpolated, then the output is the interpolated polynomial of W. (Otherwise, an appropriate message is output). The procedure consists of three steps:

Computing the syndrome. For an input word $W = (i_1, s_1) \ldots (i_l, s_l)$, let $V_{l\times l}$ be the (Vandermonde) matrix defined by $V_{j,k} = (i_k)^j$. Let $\vec{a} = s_1 \ldots s_l$. The **syndrome** of W is the $l - (d+1)$ right most elements of the l-vector product $\vec{a} \cdot V^{-1}$.
First, compute the syndrome of W.

Remark. Let $Q(\cdot)$ be the polynomial so that for every $(i, a) \in W$ we have $Q(i) = a$. Then, the vector $\vec{a} \cdot V^{-1}$ is the vector of the coefficients of $Q(\cdot)$; the syndrome consists of the coefficients of the monomials $x^{d+1}, \ldots, x^l$ of $Q(x)$. In particular, if $Q(\cdot)$ is of degree d (namely, W is a code-word), then the syndrome is the zero vector.

Computing the error-vector. The **error-vector** is the l-vector $\vec{e} = e_1 \ldots e_l$, where e_j is the 'displacement of s_j from the correct value'. Namely, assume that W is (d, r)-interpolated, and let $P(\cdot)$ be the (d, r)-interpolated polynomial of W; then $e_j = P(i_j) - s_j$, for every element $(i_j, s_j) \in W$. The error-vector is unique, since the interpolated polynomial $P(\cdot)$ is unique.

Compute the error-vector, using the syndrome. A widely used implementation of this step is the Berlkamp-Massey algorithm (see [MS pp. 365-368]).

Remark. We stress that the error-vector can be computed based on the syndrome *only*. If the input word, W, is not (d, r)-interpolated, then the computed error-vector may be erroneous.

Computing the output polynomial. Choose $2t + 1$ correct elements in W (namely, elements (i_j, a_j) such that $e_j = 0$), and use them to interpolate $P(\cdot)$. (This step will not be implemented.)

An important observation is that the syndrome can be represented as a function of the error vector only; thus, it holds no information on the (d, r)-interpolated polynomial, $P(\cdot)$, of W. Namely, let $\vec{P}$ (resp. $\vec{Q}$) be the coefficients vector of the polynomial $P(\cdot)$ (resp. $Q(\cdot)$), completed to length l. (Polynomial $Q(\cdot)$ is the polynomial satisfying $Q(i) = a$ for every pair $(i, a) \in W$.) Then,

$$\vec{Q} = \vec{a} \cdot V^{-1} = (\vec{P} \cdot V + \vec{e}) \cdot V^{-1} = \vec{P} + \vec{e} \cdot V^{-1}.$$

The last $l - (d+1)$ elements in $\vec{P}$ are zero. Therefore, for $l - (d+1) < i \leq l$, we have $Q_i = [\vec{e} \cdot V]_i$. Consequently, the last $l - (d+1)$ elements in $\vec{Q}$ (namely, the syndrome) are a linear combination of the elements of $\vec{e}$ only.

Let us now describe how we implement and use this error correcting procedure in our setting. Each element of the syndrome is a linear combination of the inputs; thus, the parties can jointly compute the syndrome. That is, given an agreed set, G, each element of the syndrome of S_G is computed as follows. Each party computes the appropriate linear combination of its shares, and invokes a V-Recon protocol with the result of this linear combination as input. Once all these V-Recon protocols are completed, each party has the full syndrome of S_G.

Once the syndrome is computed, each party uses the Berlkamp-Massey algorithm in order to *locally* compute the error vector. If, in iteration r, S_{G_r} is $(2t,r)$-interpolated, then the computed error vector is the 'true' error vector of S_{G_r}; however, if S_{G_r} is not $(2t,r)$-interpolated, then the computed error vector may be incorrect. Consequently, the parties draw the following conclusions. (Each party draws these conclusions locally; still, all the uncorrupted parties draw the same conclusions.) If the computed error vector, denoted $\vec{e'}$, contains more than r non-zero elements, then surely S_{G_r} is not $(2t,r)$-interpolated, and the parties proceed to the next iteration. If $\vec{e'}$ contains up to r non-zero elements, then the parties still have to verify that $\vec{e'}$ is the correct error vector: let G'_r be the set of parties P_i such that $e'_i = 0$; the parties will recompute the syndrome, based on G'_r alone. If the recomputed syndrome is all zeros, then $S_{G'_r}$ is $(2t,0)$-interpolated, and the parties output G'_r and terminate. If the recomputed syndrome is non-zero, the parties conclude that S_{G_r} is not $(2t,r)$-interpolated, and proceed to the next iteration. [21]

Protocol AIS is described in Figure 4-15. Let $z_{i,j}$ be P_i's share of the value shared by P_j. The (dynamic) input of each party P_i is the following accumulative set, denoted $\mathcal{Z}_i$: once P_i has successfully completed P_j's V-Share, the pair $(j, z_{i,j})$ is added to $\mathcal{Z}_i$. The parties' common output is a set G of at least $3t+1$ parties, such that each uncorrupted party P_i has completed the V-Share of every party in G (namely, $G \subseteq \{P_j | (j, z_{i,j}) \in \mathcal{Z}_i\}$), and S_G is $(2t,0)$-interpolated.

Claim 4.29 *Assume that protocol AIS is run with dynamic inputs $\mathcal{Z}_1, \ldots, \mathcal{Z}_n$ as described above. Then all the uncorrupted parties terminate protocol AIS with a common set G of at least $3t+1$ parties, such that S_G is $(2t,0)$-interpolated.*

Proof: First, assume that the ACS protocol of Step 1 of some iteration is completed by the uncorrupted parties. Then, all the uncorrupted parties compute the same syndrome in Step 2 of this iteration; thus, all the uncorrupted parties make the same decisions in Step 3. Consequently, all the uncorrupted parties complete this iteration; furthermore, either all the parties output a $(2t,0)$-interpolated set of size at least $3t+1$, or all the parties proceed to the next iteration.

Next, we show, by induction on the number of iterations, that as long as the required set is not found, all the uncorrupted parties will complete each invocation of ACS. For the base of induction, we note that the sequence $\mathcal{U}_i, \ldots, \mathcal{U}_n$ of the accumulative sets defined in Step 1 of the first iteration is $(3t+1, n)$-uniform (see Definition 3.6 on page 37). Thus, all the uncorrupted parties will complete the ACS invocation of the first iteration. For the induction step, assume that the uncorrupted parties have completed the ACS protocol of iteration r. Consequently, the sequence $\mathcal{U}_1, \ldots, \mathcal{U}_n$ is $(3t+r+1, t)$-uniform. Assume further that the set agreed upon in iteration r is not $(2t,r)$-interpolated; let G_r denote this set. Then, G_r contain at most $3t$ *uncorrupted* parties. Thus, there exists at least one uncorrupted party in $[n] - G_r$. This uncorrupted party will eventually be in the accumulative sets of all the uncorrupted parties; thus the collection $\mathcal{U}_1 \ldots \mathcal{U}_n$ is $(3t+r+2, t)$-uniform. Consequently, all the uncorrupted parties will complete the ACS invocation of iteration $r+1$.

[21] A more "time-efficient" version of this protocol lets the parties execute all the t iterations 'in parallel'; consequently, the running time of the protocol is the running time of the slowest iteration. (In this 'parallel' version, different parties may complete the 'iterations' in different order. Thus, the parties need to execute an additional simple protocol to agree on the iteration whose output is adopted.)

Protocol AIS($\mathcal{Z}_i$)

Code for party P_i, on dynamic input $\mathcal{Z}_i$.
for $0 \le r \le t$ do

1. Let $\mathcal{U}_i = \{P_j | (j, z_{i,j}) \in \mathcal{Z}_i\}$.
   Set G =ACS$[3t+1+r, n](\mathcal{U}_i)$.
2. Once G is computed, compute the syndrome of S_G:
   Let V be the Vandermonde matrix of the indices in G. Namely, let $G = k_1 \dots k_{|G|}$; then, $V_{i,j} = (k_j)^i$.
   Let $\vec{z_i} = z_{i,k_1}, \dots, z_{i,k_{|G|}}$.
   For $2t+1 < j \le |G|$ set σ_j =V-Recon$([\vec{z_i} \cdot V^{-1}]_j, [n])$.
   Let $\vec{\sigma} = \sigma_1 \dots \sigma_{t+r}$. ($\vec{\sigma}$ is the syndrome of S_G).
3. Run the Berlkamp-Massey algorithm on $\vec{\sigma}$, and let $\vec{e'}$ be the output.
   (a) If $\vec{e'}$ has more than r non-zero elements, continue to the next iteration (S_G is not $(2t, r)$-interpolated).
   (b) If $\vec{e'}$ has up to r non-zero elements, verify that $\vec{e'}$ is correct:
   Let G' be the set of parties in G whose corresponding entry in $\vec{e'}$ is zero. Repeat step 2 with respect to G'.
   If the syndrome of $S_{G'}$ is the zero vector, output G' and halt.
   Otherwise, proceed to the next iteration (S_G is not $(2t, r)$-interpolated).

Figure 4-15: AIS - The agreement on an interpolated set protocol

Finally, we note that G_t (namely, the set $[n]$) is $(2t, t)$-interpolated; thus, the required set will be found in iteration t, unless it was found beforehand. □

The multiplication protocol

Combining the Randomization and the Degree-Reduction steps, we derive a protocol for computing a multiplication gate. The code is presented in Figure 4-16.

Protocol BMUL

Code for party P_i, on inputs a_i and b_i.
Randomization

1. For $1 \le k \le t$, execute V-Share$_{i,k}(r_k)$, where $r_k \in_R F$ and P_i is the dealer.
   For $1 \le j \le n$, for $1 \le k \le t$, participate in V-Share$_{j,k}$.
   Let $h_{i,j,k}$ be P_i's output of V-Share$_{j,k}$.
2. Let $\mathcal{U}_i = \{P_j |$ V-Share$_{j,k}$ has completed, for all $1 \le k \le n\}$.
   Set G =ACS$[n-t, n](\mathcal{U}_i)$.
   Set $h_i = \sum_{j \in G} \sum_{k=1}^{t} i^k \cdot h_{i,j,k}$, and $d_i = a_i \cdot b_i + h_i$.

Degree-Reduction

3. Once d_i is computed, execute V-Share$_i(d_i)$, where P_i is the dealer.
   For $1 \le j \le n$, participate in V-Share$_j$.
4. Let $z_{i,j}$ be P_i's share of P_j's shared secret,
   and let $\mathcal{Z}'_i = \{(j, z_{i,j}) |$ V-Share$_j$ has been completed$\}$.
   Set G' =AIS$(\mathcal{Z}'_i)$.
5. Let $V^{\hat{G}'}$ be the matrix used in the Fail-Stop multiplication step, as described in section 4.3.3 on page 61, and let $\vec{z}_i = z_{i,j_1} \ldots z_{i,j_{|G'|}}$, where $j_1 \ldots j_{|G'|}$ are the indices of the parties in G'.
   For $1 \le j \le n$, Set c_j =V-Recon$([\vec{z}^{\,i}_{\hat{G}} \cdot (V^{\hat{G}'}]_j, \{j\})$.
6. Once c_i is computed, output c_i.

Figure 4-16: BMUL - The Byzantine multiplication gate protocol

4.5.3 The Byzantine protocol

The overall structure of the Byzantine protocol is the same as that of the Fail-Stop protocol. Namely, in the Byzantine protocol, denoted Bcompute, the parties execute the code of protocol FStop, with the exception that protocols GShare, MUL, and Reconstruct are replaced by protocols GV-Share, BMUL, and V-Recon, respectively.

Theorem 4.30 *Let $f : F^n \to F$ for some field F with $|F| > n$, and let A be a circuit computing f. Then, protocol Bcompute$[A]$ asynchronously $(\lceil \frac{n}{4} \rceil - 1)$-securely computes f in the bounded secure channels setting in the presence of adaptive adversaries and non-erasing parties.*

Proof: Here we consider only the case of non-adaptive adversaries. The case of adaptive adversaries is proven in the same way as in Section 4.3.6.

The proof of the Termination property, as well as the construction of the ideal-model adversary is the same as in the proof of Theorem 4.13 on page 63 (security of protocol FS-Compute). We stress that our construction of the ideal-model adversary for Fail-Stop adversaries is valid even for corrupted parties that change their inputs, although in the Fail-Stop case the inputs of the corrupted parties are not changed.

The validity of this construction in the Byzantine setting is shown in the same way as in the Fail-Stop case, with the exception that lemmas analogous to Lemmas 4.15 and 4.14 (on pages 68 and 67, respectively) need to be proven. The proof of the Byzantine analogue of Lemma 4.15 is similar to the proof of Lemma 4.15, with the exception that now, the correctness of the uncorrupted parties' outputs of the *Byzantine* implementations of the different protocols (namely, GV-Share, V-Recon, and BMUL) need to be proven. Correctness of the GV-Share and V-Recon protocols is implied by the correctness of the AVSS scheme; correctness of protocol BMUL is discussed in Section 4.5.2.

In order to state the Byzantine analogue of Lemma 4.14, let us redefine some notations. The view of a set of parties is defined as in the proof of Theorem 4.13. Fix an adversary and an input vector $\vec{x}$. The random variables μ (resp. μ') take the distribution of the corrupted parties' view of protocol Bcompute, run on input $\vec{x}$ (resp. the corrupted parties' view of a simulated computation). Lemma 4.31 below is the Byzantine analogue of Lemma 4.14. Our proof of Theorem 4.30 is thus completed. □

Lemma 4.31 *Fix an adversay and an input vector. Then, the corresponding random variables μ and μ' are identically distributed.*

Proof: We use the notations of Lemma 4.14. Namely, fix a prefix, V_i, of length i of a view (namely, V_i is a prefix of an instance of either μ or μ'). The random variable $\hat{s}_{i+1}$ (resp. $\hat{s}'_{i+1}$) describes the distribution of the $(i+1)$th delivered message in μ (resp. μ'), given that V_i is the corresponding prefix of the view.

As in the proof of Lemma 4.14, it is enough to show that the *contents* of $\hat{s}_{i+1}$ and $\hat{s}'_{i+1}$ are identically distributed, for the case that the sender of $\hat{s}_{i+1}$ is uncorrupted and the recipient is corrupted. Each message of protocol Bcompute falls into one of the following cases.

Messages of some invocation of V-Share. We distinguish two cases: if the dealer of the relevant invocation of V-Share is uncorrupted, then equality of the distributions of $\hat{s}_{i+1}$ and $\hat{s}'_{i+1}$ is implied by the proof of the privacy property of the AVSS scheme (Theorem 4.25 on page 81). If the dealer is corrupted, then the contents of $\hat{s}_{i+1}$ and $\hat{s}'_{i+1}$ can be inferred from the common prefix V_i (and are, thus, equal).

Messages of some invocation of ACS. This case is trivial, since messages of protocol ACS have empty contents.

The last three cases are different types of invocations of protocol V-Recon. We note that, as in the Fail-Stop case, in order to show that the contents of *all* the messages of some invocation of V-Recon are identically distributed in the two computations, it suffices to show that the *output* of this invocation of V-Recon is identically distributed in the two computations.

Messages of the final invocation of V-Recon. (Namely, Step 4 of protocol Bcompute. See figure 4-8 on page 64.) This case is shown using similar considerations to those of the Fail-Stop case.

Messages of an invocation of V-Recons within protocol AIS. (Namely, Step 2 of the AIS.) See Figure 4-15 on page 89.) We have seen in Section 4.5.2 that the outputs of the V-Recons of the AIS protocol are the different syndromes computed by the parties; furthermore, we have seen that the syndromes are uniquely determined by the errors introduced by the corrupted parties. However, these errors can be inferred from the common prefix V_i: the values associated with each corrupted party, P_i, in each set G_r can be inferred from the messages sent by P_i in the corresponding invocation of V-Share (these messages appear in the common prefix V_i). Thus, the errors introduced by the corrupted parties, as well as the different syndromes, are identical in the two computations.

Messages of an invocation of V-Recon within protocol BMUL. (Namely, Step 5 of BMUL. See Figure 4-16 on page 90.) As in the Fail-Stop case, we show that the corrupted parties' outputs of the t invocations of V-Recon, along with the contents of the other messages of this invocation of BMUL, received by the corrupted parties, are uniformly and independently distributed, regardless of the values of the input lines of the corresponding multiplication gate.

The polynomial $H(x)$ interpolated by the parties in Step 2 (Randomization) of protocol BMUL is $H(x) = \sum_{i \in G} \sum_{j=1}^{t} x^j \cdot S_{i,j}(x)$. (The set G is the output of the corresponding invocation of ACS, and each polynomial $S_{i,j}(\cdot)$ is the polynomial defined by the uncorrupted parties' outputs of P_i's jth invocation of V-Share. See Section 4.5.2 on page 85 for more details.) Let us first regard this polynomial in a more convenient way: reversing the order of summation, we have $H(x) = \sum_{j=1}^{t} x^j \cdot \hat{S}_j(x)$, where $\hat{S}_j(\cdot) = \sum_{i \in G} S_{i,j}(\cdot)$. All the coefficients of each polynomial $\hat{S}_j(\cdot)$ are uniformly distributed, since the set G contains uncorrupted parties. The contents of the randomization-phase messages received by the corrupted parties constitute t shares of each polynomial $\hat{S}_j(\cdot)$. Consequently, the data gathered by the corrupted parties during the entire BMUL protocol adds up to t shares of each one of the t polynomials $\hat{S}_1(\cdot), \ldots, \hat{S}_t(\cdot)$, along with the outputs of the t invocations of V-Recon (namely, t shares of the truncation polynomial $C(\cdot)$).

Fix some arbitrary input polynomials $A(\cdot)$ and $B(\cdot)$ of the multiplication step. Lemma 4.32 below shows that for each sequence of $t^2 + t$ field elements gathered by the corrupted parties there exists a unique choice of the polynomials $\hat{S}_1(\cdot) \ldots \hat{S}_t(\cdot)$ that yield this sequence. However, the polynomials $\hat{S}_1(\cdot) \ldots \hat{S}_t(\cdot)$ are uniformly distributed; thus, the sequence of field-elements gathered by the corrupted parties is uniformly distributed, both in μ and in μ'. [22]

□

[22] In Lemma 4.32 we assume, without loss of generality, that the corrupted parties are $P_1, \ldots, P_t$.

Lemma 4.32 *Let F be a finite field with $|F| > d$, and let $A(\cdot)$ and $B(\cdot)$ be polynomials of degree d over F. Then, for every sequence $a_{1,1}, \ldots, a_{d,d}, c_1, \ldots, c_d$ of elements in F there exists a unique sequence $\hat{S}_1(\cdot), \ldots, \hat{S}_t(\cdot)$ of polynomials of degree d such that:*

1. *Fore each $1 \leq i, j \leq d$, we have $\hat{S}_i(j) = a_{i,j}$.*
2. *Let $C(\cdot)$ be the truncation to degree d of the polynomial*

$$D(x) = A(x) \cdot B(x) + \sum_{j=1}^{d} x^j \cdot \hat{S}_j(x)$$

Then, for $1 \leq i \leq d$ we have $C(i) = c_i$.

Proof: See Section 4-B. □

4.6 Lower bounds

We show that our protocols have optimal resilience, both in the Fail-Stop and in the Byzantine cases. Firsr we show, in Theorem 4.34, that there exist functions that cannot be $\lceil \frac{n}{3} \rceil$-securely computed in an asynchronous network of n parties if Fail-Stop adversaries are allowed. We prove this result in two steps. First we reduce the problem of secure computation in a *synchronous* network with $n = 2t$ to the problem of secure computation in an *asynchronous* network with $n = 3t$. Next we use the results of [BGW] (and also Chor and Kushilevitz [CK]) that there exist functions that cannot be securely computed (or even approximated) when $n \leq 2t$. Simple examples of such functions are the OR and AND functions of n boolean inputs. Even though a direct (and conceivably shorter) impossiblity proof for the asynchronous case is possible, we believe that our proof by reduction is clearer as well as more general.

Next we show, in Theorem 4.35, that there exist functions that cannot be $\lceil \frac{n}{4} \rceil$-securely computed with no probability of error in an asynchronous network if general (Byzantine) adversaries are allowed. This proof is a generalization of a technique used in [BGW]. Both Theorems 4.34 and 4.35 apply even to non-adaptive adversaries.

We remark that Ben-Or Kelmer and T. Rabin [BKR] show, using techniques from [BGW, CCD, CR], how any function can be asynchronously $(\lceil \frac{n}{3} \rceil - 1)$-securely computed, in the presence of Byzantine adversaries, with exponentially small (but positive) probability of either not terminating or having wrong outputs.

For the proof, we use the following notations. For an asynchronous protocol ρ, a set G of parties, an adversary $\mathcal{A}$ and input $\vec{x}$, let $\mu_{\rho,G,\mathcal{A}}(\vec{x})$ be the random variable having the distribution of the view of the parties in G when running protocol ρ with adversary $\mathcal{A}$ and input $\vec{x}$. (Unlike the notion of adversary view defined in previous sections, here the view of a set of parties does not include the information seen by the scheduler.)

A party's output of a computation is a function of its view only. Consequently, if $\mu_{\rho,G,\mathcal{A}}(\vec{x})$ and $\mu_{\rho,G,\mathcal{A}'}(\vec{x})$ are identically distributed (for some protocol ρ, input $\vec{x}$, some set G of parties, two adversaries $\mathcal{A}$ and $\mathcal{A}'$ and some input $\vec{x}$), then the *output* of the parties in G is identically distributed with adversaries $\mathcal{A}$ and $\mathcal{A}'$.

4.6.1 Fail-Stop adversaries

We use the following result of [CK]. This result refers to *synchronous* networks where the adversaries are **eavesdropping** (namely, the corrupted parties only try to gather information, but otherwise follow the protocol). In this model, we say that a protocol **approximates** a boolean function, if for every input, with probability greater than $\frac{1}{2}$ the uncorrupted parties output the function value. Informally, a protocol is t-**private** if every adversary gathers no information from executing the protocol, other than the output of the uncorrupted parties (namely the computed function value).

Theorem 4.33 [BGW, CK]: *For every $n \geq 2$, there exist boolean functions f such that there is no synchronous $\lceil \frac{n}{2} \rceil$-private protocol for n parties that approximates f.*

Some intuition for the proof of Theorem 4.33 follows. Consider the case where $n = 2$, and f is the OR function of two boolean inputs. The first party, A, should make sure that if the input of the other party, B, is 0 then the communication between the parties will be independent of A's input (otherwise B will learn about A's input). Similarly, B should make sure that if the input of A is 0 then the communication between the parties will be independent of B's input. Now, consider the case where both A and B have input 0. Since the communication must be independent of both inputs the parties will never be able to decide on an output.

This argument can be generalized in a simple way to deal with three parties in an asynchronous network with one possible Fail-stop fault. Assume that A is good, and the third party, C, does not respond to A's messages. Then, A must complete the protocol based only on its communication with B, since C may be faulty. Furthermore, the communication with B should be independent of A's input (for the case where C is good but slow and B is faulty with input 0). The argument now continues as in the synchropbous case.

To be more precise, we prove the following lower bound based on Theorem 4.33.

Theorem 4.34 *For every $n \geq 3$, there exist boolean functions f, such that no asynchronous protocol for n parties securely $\lceil \frac{n}{3} \rceil$-computes f when Fail-Stop adversaries are allowed.*

Proof: Let $n \geq 3$, and let $m = n - \lceil \frac{n}{3} \rceil$. (Hence, $m = \lfloor \frac{2n}{3} \rfloor$ and $\lceil \frac{m}{2} \rceil \leq \lceil \frac{n}{3} \rceil$.) Let $f' : \{0,1\}^m \rightarrow \{0,1\}$ be a boolean function as in Theorem 4.33 (namely, f' cannot be approximated by any $\lceil \frac{m}{2} \rceil$-private protocol). Construct the function $f : \{0,1\}^n \rightarrow \{0,1\}$ so that for every n-vector $\vec{x}$, we have $f(\vec{x}) = f'(\vec{x}_{[m]})$. We show that *if* there exists a protocol, π, that securely $\lceil \frac{n}{3} \rceil$-computes f in an asynchronous network of n parties with Fail-Stop adversaries, *then* there exists an $\lceil \frac{m}{2} \rceil$-private *synchronous* protocol, π', for m parties, that approximates f'. The theorem follows.

Suppose we have an n-party asynchronous protocol, π, that securely $\lceil \frac{n}{3} \rceil$-computes f. We construct an m-party synchronous protocol, π', as follows. In protocol π' each party will simulate a party executing protocol π *in an asynchronous network of n parties.* The simulation proceeds as follows. Let $P'_1, \ldots, P'_m$ be the parties of the actual synchronous network. Let $P_1, \ldots P_n$ be the 'virtual' parties addressed by protocol π. Actual party P'_i will simulate virtual party P_i; parties $P_{m+1} \ldots P_n$ are not simulated. Each (actual) party keeps a queue of incoming messages. In each (synchronous) communication round, the party invokes a cycle of protocol π for the first message in the queue. Whenever protocol

π instructs to send a message to 'virtual' party P_i for $i \in [m]$, the party sends this message to P'_i. If $i > m$, the instruction is ignored. Once virtual party P_i terminates, actual party P'_i terminates with the output of protocol P_i.

We note that if the asynchronous protocol, π, doesn't terminate, then the synchronous protocol π' doesn't terminate as well. Thus, there may exist executions in which protocol π' doesn't terminate. We describe how we avoid this phenomenon at the end of the proof.

We assert the validity of protocol π' (namely, we show that it is $\lceil \frac{m}{2} \rceil$-private and approximates f). An outline of the proof follows. For every *synchronous* adversary interacting with protocol π', we describe three *synchronous* adversaries, denoted $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, for protocol π:

- Adversary $\mathcal{A}_1$ is the adversary that corresponds to the virtual execution of protocol π, when run by protocol π'. With this adversary, the messages sent by t parties re delayed (these are the parties $P_{m+1} \ldots P_n$), and additional t parties are corrupted (these are the parties that correspond to the actual corrupted parties).

- Adversary $\mathcal{A}_2$ delivers the messages in the same order as adversary $\mathcal{A}_1$. However, the parties $P_1 \ldots P_m$ are uncorrupted, and the parties $P_{m+1} \ldots P_n$ are corrupted. The security of protocol π asserts that with this adversary the parties $P_1 \ldots P_m$ output the correct function value.

- Adversary $\mathcal{A}_3$ delivers the messages sent by the parties $P_1 \ldots P_m$ in the same order as adversary $\mathcal{A}_1$. The messages of the parties $P_{m+1} \ldots P_n$ are postponed until there are no undelivered messages from the parties $P_1 \ldots P_m$. The corrupted parties are the parties that correspond to the actual corrupted parties. The security of protocol π asserts that with this adversary the set of corrupted parties gathers no information, other than the output of the uncorrupted parties.

We show below that, for any synchronous adversary that interacts with π', the outputs of the parties $P_1, \ldots, P_m$ running protocol π are identically distributed in the presence of all three (asynchronous) adversaries. It follows that, for every (synchronous) adversary the uncorrupted parties output the correct function value, and the corrupted parties gather no information other than the computed function value. Consequently, the synchronous protocol, π', is t-private and approximates f.

We now describe adversaries $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ in more detail. We partition the set of parties of protocol π (namely, $P_1, \ldots, P_n$) as follows. Let B be the set of parties that correspond to the corrupted parties the synchronous network (namely, $P_i \in B$ iff P'_i is corrupted). Let G be the set of parties that correspond to *uncorrupted* parties in the synchronous network (i.e.,, $G = \{P_1 \ldots P_m\} - B$). Let L be the set of "silent" parties (namely, $L = \{P_{m+1}, \ldots, P_n\}$).

Adversary $\mathcal{A}_1$. Corrupted parties: the corrupted parties are the parties in B; they follows the protocol without fail-stopping.
Scheduler: messages sent from the parties in $P_1, \ldots, P_m$ are delivered in a 'round robin'; messages sent from the parties in L are not delivered.

Adversary $\mathcal{A}_2$. Corrupted parties: the corrupted parties are the parties in L; they do not send any messages.
Scheduler: messages sent from the parties $P_1, \ldots, P_m$ are delivered in a 'round robin'.

By the definition of asynchronous secure computation (Definition 4.4 on page 52), the following property holds with adversary $\mathcal{A}_2$: on every input, $\vec{x}$, the parties in $G \cup B$ must complete protocol π with output $f_C(\vec{x})$, for some core set, C, of size at least $m = n - \lceil \frac{n}{3} \rceil$. The only inputs available are the inputs of the parties $P_1, \ldots, P_m$. Thus, with adversary $\mathcal{A}_2$, the parties in $G \cup B$ output $f_{[m]}(\vec{x})$. (In the sequel, we use this observation only for the parties in G.)

Furthermore, it can be seen, by induction on the number of communication events, that the following relation holds for every asynchronous protocol, ρ, and for every input vector, $\vec{x}$: Every prefix of a view of the parties $P_1, \ldots, P_m$ when running protocol ρ on input $\vec{x}$ is identically distributed with adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$. Thus, $\mu_{\pi,[m],\mathcal{A}_1}(\vec{x})$ and $\mu_{\pi,[m],\mathcal{A}_2}(\vec{x})$ are identically distributed. Consequently, the outputs of the parties $P_1, \ldots, P_m$ are identically distributed with adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$.

Adversary $\mathcal{A}_3$. Corrupted parties: as with adversary $\mathcal{A}_1$.
Scheduler: messages sent from the parties $P_1, \ldots, P_m$ are delivered in a 'round robin'; messages sent from the parties in L are delayed until all the messages from parties in $P_1, \ldots, P_m$ are delivered.

For every input, $\vec{x}$, let $\mu'_{\pi,[m],\mathcal{A}_3}(\vec{x})$ denote the longest prefix of $\mu_{\pi,[m],\mathcal{A}_3}(\vec{x})$ that does not contain messages sent by parties in L. It can be seen (again, by induction on the number of events in a prefix of a view) that the random variables $\mu'_{\pi,[m],\mathcal{A}_3}(\vec{x})$ and $\mu_{\pi,[m],\mathcal{A}_1}(\vec{x})$ are identically distributed; in particular, they have the same support set. We have seen above that in every execution where the view of the parties in G is in the support set of $\mu_{\pi,[m],\mathcal{A}_1}(\vec{x})$, these parties terminate. Consequently, with adversary $\mathcal{A}_3$ the parties in G terminate, deciding on an output, once their view is in the support set of $\mu'_{\pi,[m],\mathcal{A}_3}(\vec{x})$ (i.e., before any message from a party in L is delivered). Furthermore, the parties in G have the same output as with adversary $\mathcal{A}_1$, namely $f_{[m]}(\vec{x})$. In addition, since protocol π is secure, adversary $\mathcal{A}_3$ gathers no information other than the output of the uncorrupted parties (namely, $f_{[m]}(\vec{x})$). Consequently, the corrupted parties gather no information other than the computed function value with adversary $\mathcal{A}_1$.

It remains to fix the termination condition of protocol π'. We use the following provision. If, in the 'simulated execution', protocol π doesn't terminate after some predefined large enough number of rounds, then the party terminates with some default output. We compute this limit on the number of rounds: With adversary $\mathcal{A}_1$ protocol π terminates with probability 1. Consequently, there exist an integer, k, such that on every input, with probability at least $\frac{3}{4}$ all the uncorrupted parties complete protocol π after k communication events. (The limit $\frac{3}{4}$ is arbitrary. Any number in the interval $(\frac{1}{2}, 1)$ would do.) We fix the limit on the number of rounds to k. Thus, protocol π' always terminates, and with probability at least $\frac{3}{4}$ the uncorrupted parties output the output of protocol π, namely $f_{[m]}(\vec{x}) = f'(\vec{x}_{[m]})$. (We note that this "truncated" synchronous protocol is $\lceil \frac{n}{2} \rceil$-private, since the limit, k, is independent of the inputs.) □

4.6.2 Byzantine adversaries

For the proof of the lower bound for the Byzantine case we use the following additional notation. Fix some adversary and input vector. Let A be a set of parties. Let the **conversation**, $\mathcal{C}_A$, among the parties in A, be the random variable having the distribution of the sequence of all messages sent among the parties in A. (The order of the messages in each instance of the conversation is induced by their order in the corresponding view.)

Theorem 4.35 *For every $n \geq 4$, there exist functions f such that no asynchronous protocol for n parties securely $\lceil \frac{n}{4} \rceil$-computes f, if Byzantine adversaries are allowed.*

Proof (sketch): First, we note that the straightforward reduction to the synchronous case (used in the previous proof) does not hold in the Byzantine case, for the following reason. In the Fail-Stop case, a uncorrupted party has no way of telling whether a party is uncorrupted or corrupted; therefore, each uncorrupted party must terminate whenever *any* set of $n-t$ parties are ready to terminate. in the presence of Byzantine adversaries, there may exist strategies for the corrupted parties that cause the uncorrupted parties to recognize some party, P, as corrupted. In this case, the uncorrupted parties can wait for $n-t$ parties *other than P*. Consequently, in the simulation of the previous proof, if some corrupted party is recognized as corrupted then the uncorrupted parties may never terminate.

We prove the Theorem for the case $n = 4$. The proof can be easily generalized to all n. Let π be a four party protocol that securely 1-computes the following function:

$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } x_2 = x_3 = 1 \\ 0 & \text{otherwise} \end{cases}$$

Let P_1, P_2, P_3 and P_4 be the parties (and let x_i be the input of P_i).

Consider the following setting. Party P_1 is corrupted (with a strategy described below), and the adversary delivers the messages of parties P_1, P_2 and P_3 in a 'round robin'; messages sent by P_4 are delivered only when there are no undelivered messages of the other parties. We show below that, with small but non-zero probability, both P_2 and P_3 do not recognize P_1 as being corrupted. In this case, both P_2 and P_3 will terminate *before any message of P_4 is delivered;* this can be shown using similar considerations to those of the previous proof. We show that in this case, the output of P_2 is different than the output of P_3.

We first observe that the conversation $\mathcal{C}_{\{P_2,P_3\}}$ is independent of both x_2 and x_3: consider, for contradiction, the first message in $\mathcal{C}_{\{P_2,P_3\}}$ that depends on the input of its sender (say, P_2). Then, P_3 can learn x_2, regardless of the value of x_3. Consequently, it is the combination of the conversations $\mathcal{C}_{\{P_2,P_3\}}$ and $\mathcal{C}_{\{P_2,P_1\}}$ that determines the output of P_2. Similarly, it is the combination of the conversations $\mathcal{C}_{\{P_3,P_2\}}$ and $\mathcal{C}_{\{P_3,P_1\}}$ that determines the output of P_3.

Now, assume the following strategy of P_1: send some random string instead of each message expected of P_1. Let $x_2 = x_3 = 1$. With small but non-zero probability, the combination of the conversations $\mathcal{C}_{\{P_1,P_2\}}$ and $\mathcal{C}_{\{P_2,P_3\}}$ is consistent with input 0 of P_3 (and some input of P_1), and at the same time the combination of the conversations $\mathcal{C}_{\{P_1,P_3\}}$ and $\mathcal{C}_{\{P_3,P_2\}}$ is consistent with input 1 of P_2 (and some input of P_1). In this case, both P_2 and P_3 terminate *before any message of P_4 is delivered.* Furthermore, P_2 outputs 0, and P_3 outputs 1. That is, P_2 and P_3 have different outputs and the protocol is not 1-secure. □

Remark: Using a technique similar to the technique of the above proof, it can be shown that there exist functions that cannot be securely computed in a *synchronous* network where a third of the parties are corrupted. (This result is stated in [BGW].) Moreover, the result for the synchronous case is much stronger: consider a secure, synchronous protocol for computing the AND function of three variables (i.e., $\text{AND}(x_1, x_2, x_3) = 1$ iff $x_1 = x_2 = x_3 = 1$) in a network where the parties are $\{P_1, P_2, P_3\}$, and one corrupted party, P_1, does not send any messages. Using the same argument as in the above proof, it can be shown that the view of party P_2 is independent of the input of P_3, and the view of party P_3 is independent of the input of P_2. Consequently, there must exist inputs for which the output of the uncorrupted parties is incorrect with probability at least one half (instead of negligible probability in the above proof).

We note that this 'synchronous' strategy for the corrupted party is useless in our asynchronous model: if a corrupted party, P, does not send messages (or, alternatively, if this party is 'caught sending nonsense messages'), then the uncorrupted parties may interact with $n-t$ parties *other than* P.

4-A Expected running times

We analyze the running times of protocols FScompute and Bcompute. First, let us informally present the standard definition of the running time of an asynchronous protocol.

Consider a virtual 'external clock' measuring time in the network (of course, the players cannot read this clock). Let the **delay** of a message be the time elapsed from its sending to its receipt. Let the **period** of a finite execution of a protocol be the longest delay of a message in this execution. The **duration** of a finite execution is the total time measured by the global clock divided by the period of this execution. (Infinite executions have infinite duration.)

The **expected running time** of a protocol, is the maximum over all inputs and applicable adversaries, of the average over the random inputs of the players, of the duration of an execution of the protocol.

Let n be the number of parties, and let d be the depth of the computed circuit. Consider protocol FScompute. The expected running time of protocol ACS is $O(\log n)$; thus, the expected running time of protocol GShare is also $O(\log n)$. Protocol Reconstruct runs in constant time. Consequently, each invocation of protocol MUL has expected running time of $O(\log n)$, and Protocol FScompute has expected running time of $O(d \cdot \log n)$.

Consider protocol Bcompute. Protocols V-Share and V-Recon run in constant time. Protocol AIS, as presented in Figure 4-15 on page 89, consists of $O(n)$ iterations, where each iteration has expected running time of $O(\log n)$. However, as remarked at the end of Section 4.5.2, we can let all iterations run 'in parallel'; in this version, protocol AIS has expected running time of $O(\log n)$. Consequently, protocol Bcompute runs in $O(d \cdot \log n)$ expected time.

4-B Proofs of technical lemmas

In the sequel, we let $V^{(k)}$ denote the $k \times k$ (Vendermonde) matrix, where $(V^{(k)})_{i,j} = i^j$. Note that $V^{(k)}$ is non-singular. (When the dimension k is obvious, we write V instead of $V^{(k)}$.)

Lemma 4.3 *Let F be a finite field with $|F| > d$, and let $s \in F$. Then for every sequence $v_1, \ldots, v_d, u_1, \ldots, u_d$ of field-elements, there exists a unique polynomial $p(\cdot)$ of degree $2d$ with $p(0) = s$, such that:*

1. *For $1 \le i \le d$, we have $p(i) = v_i$.*

2. *For $1 \le i \le d$, we have $q(i) = u_i$, where $q(\cdot)$ is the truncation of $p(\cdot)$ to degree d (i.e., the coefficients of $q(\cdot)$ are the coefficients of the $d+1$ lower degrees of $p(\cdot)$).*

Proof: Let $p_0, \ldots, p_{2d}$ be the coefficients of polynomial $p(\cdot)$ claimed in the Lemma. Then, the following $2d+1$ equations hold:

$$\begin{array}{lcccccccccccl} p_0 1^0 & + & \ldots & + & p_d 1^d & + & p_{d+1} 1^{d+1} & + & \ldots & + & p_{2d} 1^{2d} & = & v_1 \\ \ldots & & & & & & & & & & & & \\ p_0 d^0 & + & \ldots & + & p_d d^d & + & p_{d+1} d^{d+1} & + & \ldots & + & p_{2d} d^{2d} & = & v_d \\ p_0 & + & \ldots & + & 0 & + & 0 & + & \ldots & + & 0 & = & s \\ p_0 1^0 & + & \ldots & + & p_d 1^d & + & 0 & + & \ldots & + & 0 & = & u_1 \\ \ldots & & & & & & & & & & & & \\ p_0 d^0 & + & \ldots & + & p_d d^d & + & 0 & + & \ldots & + & 0 & = & u_d \end{array} \tag{4.1}$$

We show that the equations (4.1) (in the variables $p_0, \ldots, p_{2d}$) are linearly independent. The coefficients matrix M of the left hand side of (4.1) can be partitioned into:

$$M_{(2d+1)\times(2d+1)} = \begin{bmatrix} A_{d\times(d+1)} & B_{d\times d} \\ C_{(d+1)\times(d+1)} & 0 \end{bmatrix}$$

Matrix C equals $V^{(d+1)}$; thus the last $d+1$ rows of M are linearly independent.

We show that matrix B is non-singular: let D be the $d \times d$ diagonal matrix, where $D_{i,i} = i^d$; then, $B = D \cdot V^{(d)}$. Matrices $V^{(d)}$ and D are non-singular; thus, B is non-singular as well. Therefore, the first d rows of M are linearly independent.

Assume that some linear combination c of the first t rows of M equals a linear combination of the last $d+1$ rows. Then, the same linear combination c, restricted to the columns of B, yields a row of d zeros, in contradiction with the non-singularity of matrix B.

We remark that the polynomial $p(\cdot)$ can be efficiently computed. Furthermore, given a degree $t \ge d$, it is possible to efficiently sample a random polynomial of degree t that satisfies conditions (1) and (2) of the lemma. $\square$

Lemma 4.4 *Let $m \ge d+1$, and let $f_1(\cdot) \ldots f_m(\cdot)$ and $g_1(\cdot) \ldots g_m(\cdot)$ be polynomials of degree d over a field F with $|F| \ge m$, such that for every $1 \le i \le d+1$ and every $1 \le j \le m$ we have $f_i(j) = g_j(i)$ and $g_i(j) = f_j(i)$. Then, there exists a unique polynomial $h(\cdot,\cdot)$ of degree d in two variables so that for every $1 \le i \le m$ we have $h(\cdot, i) = f_i(\cdot)$ and $h(i, \cdot) = g_i(\cdot)$.*

Proof: Let E be the $(d+1) \times (d+1)$ matrix where $E_{i,j}$ is the coefficient of x^j in $f_i(x)$. Then, the (i,j)th entry in $E \cdot V$ is $f_i(j)$.

Let $H \triangleq (V^T)^{-1} \cdot E$, and let $h(x,y)$ be the polynomial of degree d in two variables[23] where the coefficient of $x^i y^j$ is $H_{i,j}$. Then, for every $1 \le i,j \le d+1$ we have

$$h(i,j) = V^T \cdot H \cdot V = E \cdot V = f_i(j) = g_j(i).$$

Consequently, for every $1 \le i \le d+1$, the polynomials $f_i(\cdot)$ and $h(i,\cdot)$ are two polynomials of degree d that are equal in $d+1$ places. Thus, $f_i(\cdot) = h(i,\cdot)$. Similarly, $g_i(\cdot) = h(\cdot, i)$.

[23] Namely, $h(x,y) = \sum_{i=0}^{d} \sum_{j=0}^{d} a_{i,j} \cdot x^i y^j$, where $a_{0,0}, \ldots, a_{d,d}$ are fixed coefficients.

Now, consider an index $d+1 < i \leq m$. For every $1 \leq j \leq d+1$, we have $f_i(j) = g_j(i)$; by the construction of polynomial $h(\cdot,\cdot)$, we also have $h(\cdot,j) = g_j(i)$. Namely, $f_i(\cdot)$ and $h(i,\cdot)$ are polynomials of degree d that are equal in $d+1$ places. Consequently, $f_i(\cdot) = h(i,\cdot)$. Similarly, $g_i(\cdot) = h(\cdot, i)$. $\square$

Lemma 4.5 *Let $h(\cdot,\cdot)$, $h'(\cdot,\cdot)$ be two polynomials of degree d in two variables over a field F with $|F| > d$, and let $v_1, \ldots, v_{d+1}$ be distinct elements in F. Assume that for every $1 \leq i,j \leq d+1$ we have $h(v_i, v_j) = h'(v_i, v_j)$. Then, $h(\cdot,\cdot) = h'(\cdot,\cdot)$.*

Proof: Let W denote the $(d+1)\times(d+1)$ (Vendermonde) matrix defined by $W_{i,j} = (v_j)^i$. Let H be the $(d+1)\times(d+1)$ matrix where $H_{i,j}$ is the coefficient of $x^i y^j$ in $h(x,y)$. Let H' be similarly defined with respect to $h'(x,y)$. Using these notations, we have

$$W^T \cdot H \cdot W = W^T \cdot H' \cdot W.$$

Note that W is non-singular, since $v_1, \ldots, v_{d+1}$ are distinct. Consequently, $H = H'$. $\square$

Lemma 4.6 *Let F be a field with $|F| > d$, and let $s \in F$. Then, for every sequence $f_1(\cdot), \ldots, f_t(\cdot), g_1(\cdot), \ldots, g_t(\cdot)$ of polynomials of degree d, such that $f_i(j) = g_j(i)$ for every $1 \leq i,j \leq d$, there exists a unique polynomial $h(\cdot,\cdot)$ of degree d in two variables with $h(0,0) = s$, so that for every $1 \leq i \leq d$ we have $h(\cdot,i) = f_i(\cdot)$ and $h(i,\cdot) = g_i(\cdot)$.*

Proof: Let E be the following $(d+1)\times(d+1)$ matrix. For $1 \leq i,j \leq d$ let $E_{i,j} = f_i(j) = g_j(i)$; let $E_{j,0} = g_j(0)$, and $E_{0,i} = f_i(0)$. Finally, let $E_{0,0} = s$. Let $H \triangleq (V^T)^{-1} \cdot E \cdot V^{-1}$, and let $h(x,y)$ be the polynomial of degree d in two variables, such that the coefficient of $x^i y^j$ is $H_{i,j}$. Similar arguments to those of the proof of Lemma 4.26 show that $h(x,y)$ is the required, uniquely defined, polynomial. $\square$

Lemma 4.8 *Let F be a finite field with $|F| > d$, and let $A(\cdot)$ and $B(\cdot)$ be polynomials of degree d over F. Then, for every sequence $a_{1,1}, \ldots, a_{d,d}, c_1, \ldots, c_d$ of elements in F there exists a unique sequence $\hat{S}_1(\cdot), \ldots, \hat{S}_d(\cdot)$ of polynomials of degree d such that:*

1. *Fore each $1 \leq i,j \leq d$, we have $\hat{S}_i(j) = a_{i,j}$.*
2. *Let $C(\cdot)$ be the truncation to degree d of the polynomial $D(x) = A(x)\cdot B(x) + \sum_{j=1}^{t} x^j \cdot \hat{S}_j(x)$. Then, for $1 \leq i \leq d$ we have $C(i) = c_i$.*

Proof: Let $a_{1,1}, \ldots, a_{d,d}, c_1, \ldots, c_d$ be a sequence of field elements, and let $s_{i,j}$ be the coefficient of x^j in the ith polynomial in a sequence $\hat{S}_1(\cdot), \ldots, \hat{S}_d(\cdot)$ of polynomials satisfying requirements 1 and 2 of the Lemma. Then, requirements 1 and 2 translate to d^2+d equations in the d^2+d variables $s_{1,0} \ldots s_{d,d}$. It remains to show that the $(d^2+d)\times(d^2+d)$ matrix M of the coefficients of these equations is non-singular. Matrix M has the following form:

$$M = \begin{bmatrix} W_1 & 0 & 0 & \ldots & 0 \\ 0 & W_2 & 0 & \ldots & 0 \\ \ldots & & & & \\ \ldots & & & & \\ 0 & 0 & \ldots & 0 & W_d \\ U_1 & U_1 & \ldots & & U_d \end{bmatrix}$$

where each sub-matrix W_i (of dimensions $d \times (d+1)$) is constructed by deleting the last row from $V^{(d+1)}$, and every matrix U_i (of dimensions $d \times (d+1)$) is constructed by deleting the $i-1$ leftmost columns from a $V^{(d)}$ matrix and appending i zero columns on the right.

Clearly, the first d^2 rows in M are linearly independent, and the last d rows in M are linearly independent. It remains to show that no non-trivial linear combination of the first d^2 rows equals a linear combination of the last d rows.

Let the (d^2+d)-vector L_1 be a linear combination of the first d^2 rows, let the (d^2+d)-vector L_2 be a linear combination of the last d rows, and assume that $L_1 = L_2$. We show that $L_1 = L_2 = 0$. Partition L_2 to d blocks, where each block contains $d+1$ elements (the ith block is a combination of the elements in U_i).

The d rightmost columns of U_d are zero, and the d rightmost columns of W_d are linearly independent. Thus, $L_1 = L_2$ involves no rows of W_d. Consequently, *all* the last $d+1$ entries (namely, the dth block) in L_2 are zero.

However, U_{d-1} is a shift-right by one of U_d. Thus, the d rightmost entries in the $(d-1)$th block of L_2 are zero as well. Using a similar argument to that of the last paragraph, we get that *all* the entries in the $(d-1)$th block in L_2 are zero.

Applying this argument $d-2$ more times, we get that $L_1 = L_2 = 0$. □

Chapter 5

Asynchronous Byzantine agreement

We describe the first $(\lceil \frac{n}{3} \rceil - 1)$-resilient asynchronous Byzantine agreement protocol with polynomial complexity. We use ideas and techniques that emerge from secure multiparty computation. See Section 1.5 for an introductory presentation and discussion.

In Section 5.1 we recall the asynchronous model and definitions of Byzantine Agreement and Asynchronous Verifiable Secret Sharing (AVSS). In Section 5.2 we state our main theorems, and present an overview of our protocols. In Section 5.3 we describe tools used in our construction. In Sections 5.4 through 5.6 we describe our $(\lceil \frac{n}{3} \rceil - 1)$-resilient AVSS scheme. In Sections 5.7 and 5.8 we describe our BA protocol given an AVSS scheme.

5.1 Definitions

We assume the same model as in Chapter 4 (see Section 4.1 on page 49). We also use the same conventions for writing asynchronous protocols (see Section 4.1.3), and the same measure of running times of asynchronous protocols (see Section 4-A). We re-define Byzantine agreeement and AVSS. The definitions here are identical to the definitions in Chapter 4 (Definitions 4.5 and 4.18, respectively), with the exception that here we allow the parties to not terminate (and in AVSS also to have wrong output) with small probability.

Definition 5.1 *Let π be an asynchronous protocol for which each party has binary input. We say that π is a $(1-\epsilon)$*-terminating, t-resilient Byzantine Agreement *protocol if the following requirements hold, for every t-adversary and every input.*

- **Termination.** *With probability $1 - \epsilon$ all the uncorrupted parties complete the protocol (i.e., terminate locally).*
- **Correctness.** *All the uncorrupted parties who have terminated have identical outputs. Furthermore, if all the uncorrupted parties have the same input, σ, then all the uncorrupted parties output σ.*

Definition 5.2 *Let S be a finite set. Let (S, R) be a pair of protocols in which a dealer, D, shares a secret $s \in S$. All parties invoke protocol* S*, and later invoke protocol* R *with the* *local output of protocol* S *as local input. We say that* (S, R) *is a* $(1-\epsilon)$*-correct, t-resilient AVSS scheme for n parties if the following hold, for every t-adversary.*

- **Termination.** *With probability* $1-\epsilon$ *the following requirements hold.*

  1. If the dealer is uncorrupted, then each uncorrupted party will eventually complete protocol S*.*

  2. If some uncorrupted party has completed protocol S*, then each uncorrupted party will eventually complete protocol* S*.*

  3. If all the uncorrupted parties have completed protocol S*, then all the uncorrupted parties will complete protocol* R*.*

- **Correctness.** *Once the first uncorrupted party has completed protocol* S*, then a value, r, is fixed, such that the following requirements hold with probability* $1-\epsilon$*:*

  1. Each uncorrupted party outputs r. (Namely, r is the reconstructed secret.)

  2. If the dealer is uncorrupted, then r is the shared secret, i.e. $r = s$*.*

- **Secrecy.** *If the dealer is uncorrupted and no uncorrupted party has begun executing protocol* R*, then the information gathered by the adversary during the computation is independent of the shared secret.*

Remark: We stress that an uncorrupted party is not required to complete protocol S in case that the dealer is corrupted. We do not distinguish between the case where an uncorrupted party did not complete protocol S, and the case where an uncorrupted party has completed S unsuccessfully.

5.2 Overview of the protocols

First, let us state our main results.

Theorem 5.3 (AVSS). *Let* $n \geq 3t+1$*. For every* $\epsilon > 0$ *there exists a* $(1-\epsilon)$*-correct, t-resilient AVSS scheme for n parties. Conditioned on the event that the honest parties terminate, they do so in constant time. Furthermore, the computational resources required of each party are polynomial in n and* $\log \frac{1}{\epsilon}$*.*

Theorem 5.4 (BA). *Let* $n \geq 3t+1$*. For every* $\epsilon > 0$ *there exists a* $(1-\epsilon)$*-terminating, t-resilient, asynchronous Byzantine Agreement protocol for n parties. Conditioned on the event that the honest parties terminate, they do so in constant expected time. Furthermore, the computational resources required of each party are polynomial in n and* $\log \frac{1}{\epsilon}$*.*

Our Byzantine Agreement protocol is complex and involves many layers. To facilitate the reading we first present an overview of our protocol. Let $\mathcal{F}$ be a field of size greater than n. All the computations in the sequel are done in $\mathcal{F}$. The BA protocol employs the idea of using 'common coins' to reach agreement, as follows.

BA using Common Coin. This part of our protocol follows the constructions of Rabin, Bracha and Feldman [MRa2, Br, Fe]. The protocol proceeds in rounds. In each round, each party has a 'modified input' value. In the first round, the modified input of each party is his local input. In each round the parties invoke two protocols, called Vote and Common Coin. Protocol Common Coin has the following property. Each party has a random input, and binary output. For every value $\sigma \in \{0,1\}$, with probability at least $\frac{1}{4}$ *all* the honest parties output σ. In protocol Vote each party tries to establish whether there exists a 'distinct majority' for some value amongst the parties' modified inputs of this round (we define distinct majority in the sequel). If a party recognizes a 'distinct majority' for some value he takes this value to be his modified input for the next round. Otherwise, he sets his modified input for the next round to be the output of protocol Common Coin. We show that no two honest parties ever recognize a distinct majority for two different values. This is used to show that in each round, with probability at least $\frac{1}{4}$ all parties have the same modified input.

If all the honest parties have the same modified input in some round, then all the honest parties will recognize this and terminate outputting this value. It follows that the BA protocol terminates within a constant expected number of rounds. Given that all the honest parties complete all Common Coin protocols they invoked, then each round of the Byzantine Agreement protocol terminates in constant time.

Common Coin using AVSS. Our construction follows Feldman, and Feldman and Micali [Fe, FM]. The protocol proceeds roughly as follows. First, each party shares n random secrets using our AVSS scheme. Once a party is assured that enough secrets have been properly shared, he starts reconstructing the relevant secrets. Once all these secrets are reconstructed, each party locally computes his output based on the reconstructed secrets.

AVSS from scratch. Our AVSS scheme is constructed in three 'layers'. Each layer consists of a different secret sharing scheme (with an allowed-error parameter, ϵ). The scheme of the lowest layer is called Asynchronous Recoverable Sharing A-RS). The next scheme is called Asynchronous Weak Secret Sharing (AWSS). The last ('top') layer is an AVSS scheme (as in Definition 5.2). Each scheme is used as a building block for the next. All three sharing schemes satisfy the *termination* and *secrecy* requirements of Definition 5.2. In all three schemes, if the dealer is honest then the honest parties always reconstruct the secret shared by the dealer. The correctness property for the case that the dealer is faulty is upgraded from A-RS to AWSS and finally to AVSS:

A-RS- Once the first honest party completes the reconstruction phase, a subset $S \subseteq \mathcal{F}$, of size $2t+1$, is fixed. With probability $1-\epsilon$, each honest party will reconstruct a value $r \in S \cup \{null\}$. (We stress that it is not required that all honest parties end up with the same reconstructed value.)

AWSS- Once the first honest party completes the sharing phase, a value s is fixed. With probability $1-\epsilon$, each honest party will reconstruct either s or *null*.

AVSS- Once the first honest party completes the sharing phase, a value s is fixed. With probability $1-\epsilon$, each honest party will reconstruct s.

5.3 Tools

5.3.1 Information Checking Protocol- ICP

The Information Checking Protocol (ICP) is a tool for authenticating messages in the presence of (computationally unbounded) faulty parties. The ICP was introduced by T.

ICP

Generation phase (Gen):

1. The dealer D, having a secret $s \in \mathcal{F}$, chooses random values $y_1, ..., y_{2k}$ and $b_1, ..., b_{2k}$ uniformly distributed in F. He computes $c_i \triangleq b_i s + y_i$ for $1 \leq i \leq 2k$.
2. D sends s and $\vec{y}_s \triangleq y_1, ..., y_{2k}$ to I.
3. D sends the *check vectors* $(b_1, c_1), ..., (b_{2k}, c_{2k})$ to R.

Verification (Ver):

1. I chooses k distinct random indices $d_1, ..., d_k$, $1 \leq d_i \leq 2k$, and asks R to reveal $(b_{d_1}, c_{d_1}), ..., (b_{d_k}, c_{d_k})$.
2. R reveals these values. The remaining indices will be the *unrevealed indices.*
3. For each one of the revealed indices d_i, I tests whether $s \cdot b_{d_i} + y_{d_i} = c_{d_i}$. If *all* k indices satisfy this requirement then I sets **Ver**$= 1$. Otherwise, he sets **Ver**$= 0$.

Authentication (Auth):

1. I sends s and the rest of the y's to R.
2. If $k/2$ out of the unrevealed indices d_i satisfy $s \cdot b_{d_i} + y_{d_i} = c_{d_i}$ then R sets **Auth**$= s$, otherwise **Auth**$=$*null*.

Figure 5-1: The Information Checking Protocol of [TRa, RB]

Rabin and Ben-Or [TRa, RB] We first state the properties of this protocol. Next we sketch the [TRa, RB] construction.

The protocol is executed by three parties: a dealer D, an intermediary I, and a receiver R. The dealer hands a secret value s over to I. At a later stage, I is required to hand this value over to R, and to convince R that s is indeed the value which I received from D. More precisely, the protocol is carried out in three phases:

GENERATION(s) is initiated by D. In this phase D hands the secret s to I and some auxiliary data to both I and R.

VERIFICATION is carried out by I and R. In this phase I decides whether to continue or abort the protocol. I bases his decision on the prediction whether, in the Authentication phase, R will output s, the secret held by I. We denote continuation (resp., abortion) by **Ver**$= 1$ (resp., 0).

AUTHENTICATION is carried out by I and R. In this phase R receives a value s' from I, along with some auxiliary data, and either accepts or rejects it. We denote acceptance of a secret s', (resp., rejection) by **Auth**$= s'$ (resp., null).

The ICP has the following properties, given an 'allowed-error parameter' ϵ:

Correctness:

1. If D, holding a secret s, I and R are all honest, then **Ver**$= 1$ and **Auth**$= s$.
2. If I and R are honest, and I has decided, in the Verification phase, to continue the protocol, with local input s', then with probability $(1 - \epsilon)$, R will output s' in the Authentication phase.
3. If D and R are honest, and R accepted a secret s' in the Authentication phase, then

$s' = s$, with probability $(1-\epsilon)$.

Secrecy:

4. If D and I are honest, then as long as I has not joined in the Authentication phase, R has no information about the secret s.

For self containment we sketch the [TRa, RB] construction in Figure 5-1.[1]

5.3.2 Broadcast

We use the same definition of Broeadcast as in Chapter 4 (see Section 4.2.2). We also use the elegant implementation of Bracha [Br], described there.

5.4 Asynchronous Recoverable Sharing — A-RS

Unlike synchronous systems, in an asynchronous system it is not possible to decide whether a party from which messages do not arrive is faulty or just slow. As argued in the Introduction, this difficulty causes known techniques for AVSS to fail when $n \leq 4t$. We overcome this difficulty using the Information Checking Protocol (ICP) described in the previous section. We first show how the ICP is used to construct A-RS.

An A-RS scheme satisfies the *termination* and *secrecy* requirements of AVSS (Definition 5.2). Also, if the dealer is honest then the honest parties always reconstruct the secret shared by the dealer. The difference from AVSS lies in the case where the dealer is faulty. For A-RS we only require that:

Correctness of A-RS for the case that the dealer is faulty: *once the first honest party completes the reconstruction phase, a subset $S \subseteq \mathcal{F}$, of size $n-t$, is defined. With probability $1-\epsilon$, each honest party will reconstruct a value $r \in S \cup \{null\}$. (We stress that it is not required that all honest parties end up with the same reconstructed value.)*

A-RS has a "synchronizing effect" on the network, in the sense that it guarantees a bounded wait for receiving values, *even from faulty parties.* That is, if we have completed the sharing of a value by some party, then we are guaranteed to eventually reconstruct *some* (not necessarily valid) value. We use this property in the construction of AWSS in the next section.

We describe our construction. Basically, we use Shamir's classic secret sharing scheme [Sh] while using ICP to verify the shares. This way, we can make sure that the following two properties hold simultaneously (with overwhelming probability): (a) the shares of at least $t+1$ honest parties will always be available at reconstruction, and (b) if the dealer is honest then all the shares available at reconstruction are the originally dealt shares. A more detailed description of our construction follows.

Remark: Here, as well as in all subsequent secret sharing schemes, we use the convention that the dealer, in addition to executing his specific code, also participates as one of the parties (and will eventually have a share of the secret).

[1] In fact, the version presented here differs from the one in [TRa, RB] in the decision rule for R in the Authentication stage (in the original construction R accepted the secret if there *existed* an index d_i that satisfied the requirement. This modification allows us to tolerate fields of size only slightly larger than the number of parties.

Sharing protocol. The dealer, having a secret s, chooses values $a_1, ..., a_t \in_{\mathrm{R}} \mathcal{F}^t$, and defines the polynomial $f(x) = a_t x^t + ... + a_1 x + s$.[2] (We call this process **choosing a random polynomial** $f(x)$ **for** s.) Next, for each i, the dealer sends $\phi_i \triangleq f(i)$ to P_i. We say that ϕ_i is P_i's share of s. In addition, for each share ϕ_i and for each party P_j, the dealer executes the Generation phase of ICP (described in Section 5.3.1), with party P_i acting as the intermediary, and party P_j acting as the receiver. The ICP protocol will have the appropriate allowed error parameter ϵ' (we set $\epsilon' = \frac{\epsilon}{n^2}$). We denote this execution of the Generation phase by $\mathbf{Gen}_{D,i,j}[\epsilon'](\phi_i)$. (In the sequel, we omit the parameter ϵ' from the notation.) Next, every two parties P_i and P_j execute the Verification phase of ICP, denoted $\mathbf{Ver}_{i,j}[\epsilon']$. Success of $\mathbf{Ver}_{i,j}[\epsilon']$ assures P_i that, with probability $1 - \epsilon'$, P_j will later authenticate his share ϕ_i. Once P_i successfully (i.e. with output 1) terminates $2t + 1$ invocations of $\mathbf{Ver}_{*}[\epsilon']$, he broadcasts `(OK,` P_i`)` . A party completes the sharing protocol upon completing $2t + 1$ broadcasts of the form `(OK,` P_k`)` . Our protocol, denoted A-RS Share, is presented in Figure 5-2.

Definition 5.5 *Let $\mathcal{F}$ be a field, and let $r \geq t + 1$. A set $\{(i_1, \phi_{i_1}), \ldots, (i_r, \phi_{i_r})\}$ where $\phi_{i_j} \in \mathcal{F}$ is said to* **define a secret** *s if there exists a (unique) polynomial $f(x)$ of degree at most t, such that $f(0) = s$, and $f(i_j) = \phi_{i_j}$, for $1 \leq j \leq r$. (We shall interchangeably say that the* **shares** *$\phi_{i_1}, \ldots, \phi_{i_r}$ define the polynomial $f(x)$.)*

Using interpolation, one can efficiently check whether a given set of shares define a secret.

Reconstruction protocol. First, each party P_i initiates an broadcast of his share, ϕ_i. He then executes the Authentication phase of ICP with every other party, P_j. (We denote this execution by $\mathbf{Auth}_{i,j}$.) For each party P_j, whose share is accepted (namely, $\mathbf{Auth}_{j,i}[\epsilon'] = \phi'_j$), party P_i broadcasts `(`P_i `authenticates` P_j`)` .

Party P_i considers a share, ϕ_j, **legal**, if P_j has been authenticated by at least $t+1$ parties. Once P_i has $t + 1$ legal shares, he computes the secret which they define, and broadcasts it. If there exists a value which is broadcasted by at least $n - 2t$ parties, then P_i output this value. Otherwise, P_i outputs *null*. (This extra broadcast is required to limit the size of the set of possible values reconstructed by the different parties.)

The reconstruction protocol, denoted A-RS-Rec, is presented in Figure 5-3.

Theorem 5.6 *Let $n \geq 3t + 1$. Then for every $\epsilon > 0$, the pair* (A-RS-Share[ϵ],A-RS-Rec[ϵ]) *is a $(1 - \epsilon)$-correct, t-resilient A-RS scheme for n parties.*

Proof: Fix a t-adversary. We show that the Termination and Secrecy requirements of Definition 5.2 (AVSS) are met, as well as the Correctness property for the case that the dealer is honest. The Correctness for the case that the dealer is honest was stated above.

Termination (1): When the dealer is honest, the Verification phase of ICP between every two honest parties will terminate with **Ver**=1 (see the definition of ICP on page 104). Therefore, each honest party will have $2t + 1$ successful verifications for his share ϕ_i. Thus, each honest party P_i will initiate an broadcast of `(OK,` P_i`)` . Consequently, each honest party will complete participation in $2t+1$ broadcasts of the form `(OK,` P_j`)` , and terminate protocol A-RS-Share.

[2] We let $e \in_{\mathrm{R}} D$ denote an element e chosen uniformly at random from domain D.

Protocol A-RS-Share[ϵ]

Code for the dealer, with secret s: [a]

1. Choose a random polynomial, $f(x)$, for s.
2. Set $\phi_i \stackrel{\triangle}{=} f(i)$ for $1 \leq i \leq n$.
   Let $\epsilon' \stackrel{\triangle}{=} \frac{\epsilon}{n^2}$. Execute $\mathbf{Gen}_{D,i,j}[\epsilon'](\phi_i)$ for $1 \leq i, j \leq n$.

Code for party P_i:

3. For all $j \in [n]$: Wait until $\mathbf{Gen}_{D,i,j}$ is completed; then, initiate $\mathbf{Ver}_{i,j}$.
4. If $\mathbf{Ver}_{i,j} = 1$ for $2t+1$ parties P_j, then initiate broadcast `(OK,` P_i`)` .
5. Upon completing participation in $2t+1$ different broadcasts of the form `(OK,` P_k`)` , terminate.

Figure 5-2: The A-RS sharing protocol

[a] In this and all subsequent sharing protocols, the dealer also executes the code of a regular party.

Protocol A-RS-Rec[ϵ]

Code for party P_i:

1. Broadcast the share ϕ_i.
2. For each party P_j for whom the broadcast of his share has been completed (with ϕ'_j), initiate $\mathbf{Auth}_{j,i}$.
   If $\mathbf{Auth}_{j,i} = \phi'_j$, then initiate the broadcast `(`P_i `authenticates` P_j`)` .
3. Consider a share, ϕ'_j, legal if at least $t+1$ broadcasts of the form `(`P_k `authenticates` P_j`)` have been completed. Create a set called IS_i (standing for Interpolation Set. Add every legal share to IS_i.
4. Once $|IS_i| = t+1$, compute, using interpolation, the secret s defined by the shares in IS_i. Broadcast `(`P_i `suggests secret` s`)` .
5. Wait until completing $n-t$ `(`P_* `suggests secret` s'`)` broadcasts.
   If there is a value, s', which appears in at least $n-2t$ such broadcasts, then output s'. Otherwise, output *null*.

Figure 5-3: The A-RS reconstruction protocol

Termination (2): If one honest party has completed A-RS-Share, then he has completed participation in $2t+1$ `(OK,` P_j`)` broadcasts. By the properties of broadcast (Definition 4.6), every honest party will complete these $2t+1$ broadcasts, and will thus complete A-RS-Share.

Termination (3): Let E be the event that no errors occur in all the invocations of ICP. (That is, all the requirements listed in Section 5.3 hold with no probability of error.) We know that event E occurs with probability at least $1 - n^2\epsilon' = 1 - \epsilon$.

If some honest party, P_i, has completed the A-RS-Share protocol, then $2t+1$ parties have broadcasted `(OK, ...)` in Step 3 of A-RS-Share. Out of these broadcasts, at least $t+1$ have originated with honest parties. Let G_i be this set of at least $t+1$ honest parties, relative to party P_i.

For each party $P_j \in G_i$ there exist $t+1$ honest parties who have broadcasted `(OK,` P_j`)` . Now, assume event E occurs. By the properties of ICP, for each $P_j \in G_i$ there will be $t+1$ parties P_k who broadcast `(`P_k `authenticates` P_j`)` . Thus, the share of each party $P_j \in G_i$ will be considered legal by each honest party. Therefore every honest party will have $t+1$ legal shares in Step 3 of A-RS-Rec and will suggest some secret. Consequently, each honest party will have $n-t$ suggested secrets, and will complete protocol A-RS-Rec.

Correctness (1): If the dealer is honest and event E occurs, then each share ϕ_j in the set IS_i of each honest P_i, has originated with D (namely, $\phi_j = f(j)$). Therefore, the shares in IS_i define the secret, s, shared by the dealer, and P_i will broadcast `(`P_i `suggests secret` s`)` . Consequently, each honest party will complete $n-t$ such broadcasts, out of which at least $n-2t$ suggest the secret s. Hence, each honest party will output s.

Correctness (2): Let P_i be the first honest party to complete $n-t$ broadcasts of the form `(`P_k `suggests secret ...)` . Let S denote the set of secrets suggested in these $n-t$ broadcasts. Now, consider another honest party, P_j. At least $n-2t$ out of the $n-t$ broadcasts completed by P_j in Step 5 of A-RS-Rec are of values in S. P_j outputs a non-null value r only if this value appears $n-2t$ times in the broadcasts which he completes. However, in this case it must be that $r \in S$.

Secrecy: If the dealer is honest and no honest party has initiated protocol A-RS-Rec, then any set of $t' \leq t$ parties have only t' shares of the secret, along with the data received during the ICP Generation and Verification Protocols with respect to the other parties' shares of the secret. The secrecy properties of ICP and Shamir's secret sharing scheme imply that the shared secret is independent of this data. □

5.5 Asynchronous Weak Secret Sharing — AWSS

We construct an asynchronous secret sharing scheme called Asynchronous Weak Secret Sharing (AWSS). An AWSS scheme satisfies the *termination* and *secrecy* requirements of AVSS (Definition 5.2 on page 102). The correctness requirement is as follows. If the dealer is honest then the honest parties always reconstruct the secret shared by the dealer, as in AVSS. The difference from AVSS lies in the case where the dealer is faulty. For AWSS we only require that:

Correctness of AWSS for the case that the dealer is faulty: *Once an honest party completes the sharing protocol, a value $r \in \mathcal{F} \cup \{null\}$ is fixed. With probability $1-\epsilon$, each honest party outputs either r or nullupon completing the reconstruction protocol.*

Remarks:

- We stress that if $r \neq null$ then some of the honest parties may output r and some may output *null*. The adversary can decide, during the execution of the reconstruction protocol, which parties will output *null*.

- Our construction of A-RS, described in the previous section, does not satisfy the requirements from an AWSS scheme. There, in case that the dealer is faulty, the adversary has the power to decide, *at the reconstruction stage,* which value each honest party reconstructs (out of the predefined set of size $2t + 1$).

Our construction. Our construction follows the synchronous WSS version of [TRa, RB]. The idea is to make sure that, at the end of the sharing protocol, each party will have an interpolation set of parties whose shares will be available in the reconstruction stage. (The availability of these shares is guaranteed using A-RS, as described below.) The interpolation sets of every two honest parties P_i and P_j will have an intersection of at least $t+1$ parties. This way, if the shares of the parties in the interpolation sets of P_i and P_j define some secret, then they define the same secret.

In the reconstruction stage the shares of all the parties are jointly reconstructed. Next, each party computes and outputs the secret defined by the shares of the parties in his interpolation set. If these shares do not define a secret then the party outputs *null.*

Sharing protocol. The dealer, sharing a secret $s \in \mathcal{F}$, chooses a random polynomial $f(x)$ of degree t for s. For each i, the dealer sets $\phi_i \triangleq f(i)$. In addition, for each share ϕ_i and for each party P_j, the dealer executes the generation phase of ICP with party P_i acting as the Intermediary, and party P_j acting as the Receiver. (This part is the same as in A-RS-Share.)

Next, each party shares each value received from the dealer (including the values received in the ICP-Generation Protocol), using A-RS-Share (Figure 5-2). Party P_i creates a dynamic set, $\mathcal{C}_i$. (See Section 4.1.3 on page 53 for a definition of dynamic sets). Once P_i completes all the A-RS-Shares of P_j's values, P_j is added to $\mathcal{C}_i$. Next party P_i initiates the ICP-Verification Protocol for ϕ_i with each party $P_j \in \mathcal{C}_i$, and with the appropriate allowed-error parameter, ϵ'. (We denote this execution by $\mathbf{Ver}_{i,j}[\epsilon'](\phi_i)$).

Let A_i be P_i's set of parties P_j such that $\mathbf{Ver}_{i,j}[\epsilon'](\phi_i) = 1$. Once $|A_i| = 2t+1$, party P_i broadcasts A_i. (This *acceptance set,* A_i, is the set of parties who will later accept ϕ_i with high probability.)

Let $\mathcal{E}_i$ be P_i's dynamic set of parties $P_j \in \mathcal{C}_i$ whose broadcast of A_j has been completed, and $A_j \subseteq \mathcal{C}_i$. Once $|\mathcal{E}_i| \geq 2t+1$, party P_i broadcasts $IS_i \triangleq \mathcal{E}_i^{(2t+1)}$. (This *eligibility set,* $\mathcal{E}_i$, is the set of parties whose share will be later considered either legal or *null.* The *interpolation set,* IS_i, is the set of parties whose shares will later define the secret associated with P_i.)

Let F_i be P_i's set of parties P_j whose IS_j broadcast has been completed, and $IS_j \subseteq \mathcal{E}_i$. Once $|F_i| = n - t$, party P_i completes the sharing protocol. (This *final set,* F_i, is the set of parties, with whom P_i will later associate a secret.)

Reconstruction protocol. (Remark: it will be seen that when party P executes the reconstruction protocol, he essentially doubles up as each other party, i.e. he locally executes the protocols of the other parties). Initially, each party P_i reconstructs the values shared, using A-RS-Share, by each party $P_j \in \mathcal{E}_i$, as follows. Recall that each party in $\mathcal{E}_i$ shared, in the sharing protocol, values he received as an intermediary of ICP, and values he received as a receiver of ICP. Party P_i first invokes A-RS-Rec for all the values that the parties in $\mathcal{E}_i$ received as intermediaries in ICP. Call these invocations of A-RS-Rec *preliminary.* For each party $P_j \in \mathcal{E}_i$, once all the preliminary A-RS-Rec have been completed, P_i invokes the A-RS-Rec of (the rest of) the values shared by each $P_l \in A_j$. *We note that the order in which the A-RS-Rec are invoked is crucial for the correctness of the scheme.*

For each party $P_j \in \mathcal{E}_i$ and for each party $P_l \in A_j$, once all the necessary values are reconstructed, P_i computes P_l's expected result of $\mathbf{Auth}_{j,l}[\epsilon']$ (by 'doubling up' as P_l). Next, P_i associates a value with party P_j as follows. If P_i finds out that $t+1$ parties in A_j have outputted $\mathbf{Auth}_{j,l} = \phi_j$ then P_i associates ϕ_j with P_j. (We then say that ϕ_j is *legal*). Otherwise, P_i associates the value "null" with P_j.

Now, for each party $P_k \in F_i$, once the values associated with all the parties in IS_k are computed, if all the legal shares in IS_k define a secret, s, then P_i considers s to be the secret *suggested* by P_k. Otherwise, the secret suggested by P_k is *null.*

Once the values shared by all the parties in $\mathcal{E}_i$ are reconstructed, P_i computes the secrets suggested by all the parties in F_i. (Again, this is done by 'doubling up' as each party in F_i.) If all the parties in F_i suggest the same secret, s, then P_i outputs s. Otherwise, P_i outputs *null.* (Equivalently, P_i can output the secret defined by the reconstructed shares of the parties in $\cup_{P_j \in F_i} IS_j$. If these shares do not define a secret then P_i outputs *null.*)

The AWSS-Share and AWSS-Rec are described in Figures 5-4 and 5-5 respectively.

Protocol AWSS-Share$[\epsilon]$

Code for the dealer, on parameter ϵ and secret s:

1. Choose a random polynomial $f(x)$ for s. Set ϵ' and ϵ'' to the appropriate values, depending on ϵ.
2. Set $\phi_i \triangleq f(i)$. For each two parties P_i and P_j execute $\mathbf{Gen}_{D,i,j}[\epsilon''](\phi_i)$.

Code for party P_i, on parameter ϵ:

3. Invoke A-RS-Share$[\epsilon']$ (as a dealer) for each value received from D in Step 2. Participate in all other invocations of A-RS-Share$[\epsilon']$.

   Create a dynamic Communication Set, $\mathcal{C}_i$. Add party P_j to $\mathcal{C}_i$ if all the A-RS Shares initiated by P_j have been completed.
4. For each party P_j invoke $\mathbf{Ver}_{i,j}[\epsilon''](\phi_i)$.

   Create an Acceptance Set, A_i. Add party $P_j \in \mathcal{C}_i$ to A_i if $\mathbf{Ver}_{i,j}[\epsilon''](\phi_i) = 1$.
5. Once $|A_i| = 2t+1$ broadcast `(Acceptance Set,`P_i`,`A_i`)`.

   Create a dynamic Eligibility Set, $\mathcal{E}_i$, which will contain the parties P_j for whom the broadcast of the form `(Acceptance Set,`P_j`,`A_j`)` has been completed, and $P_j \in \mathcal{C}_i$ and $A_j \subseteq \mathcal{C}_i$.
6. Define the Interpolation Set $IS_i \triangleq \mathcal{E}_i^{(n-t)}$. Broadcast `(Interpolation set,`$P_i$`,`$IS_i$`)`.

   Create a Final Set, F_i, which will contain the parties P_j for whom the broadcast of the form `(Interpolation Set,`P_j`,`IS_j`)` has been completed, and $IS_j \subseteq \mathcal{E}_i$.
7. Once $|F_i| = n-t$ terminate.

Figure 5-4: The AWSS Sharing Protocol

Theorem 5.7 *Let* $n \geq 3t+1$. *Then for every* $\epsilon > 0$, *the pair* (AWSS-Share$[\epsilon]$,AWSS-Rec$[\epsilon]$) *is a* $(1-\epsilon)$*-correct, t-resilient AWSS scheme for n parties.*

Protocol AWSS-Rec[ϵ]

Code for party P_i, on parameter ϵ:

1. For each party $P_j \in \mathcal{E}_i$ and for each party $P_l \in A_j$:
   (a) Execute A-RS-Rec[ϵ'] (Figure 5-3) of ϕ_j, and of P_j's values needed for $\mathbf{Auth}_{j,l}[\epsilon'']$. (These values were shared by P_j in Step 3 of AWSS-Share.)
   (b) Upon ending the previous step, execute the A-RS-Rec of P_l's values needed for $\mathbf{Auth}_{j,l}[\epsilon'']$ (These values were shared by P_l in Step 3 of AWSS-Share.)
   (c) Using the reconstructed values from the two previous steps simulate $\mathbf{Auth}_{j,l}[\epsilon'']$ acting both as P_j and P_l. If at least $t+1$ $\mathbf{Auth}_{j,*}[\epsilon''] = (\phi_j)$ then associate the share ϕ_j with party P_j, else associate "null" with P_j.
2. Let $B = \{\phi \mid \phi \neq null,\ \phi \text{ is associated with party} P_l, \text{and } P_l \in IS_j,\ P_j \in F_i\}$, If all the shares of B define a secret s, then output s, else output *null*.

Figure 5-5: The AWSS - Reconstruction Protocol

As in the proof of Theorem 5.6, let E be the event that no errors occur in all the invocations of ICP. (That is, all the requirements listed in Section 5.3.1 hold with no probability of error.) In proving Theorem 5.7 we assume that event E occurs. For convenience and clarity, we partition the proof of Theorem 5.7 to several short lemmas. (The Termination and Secrecy properties are taken from Definition 5.2.)

Lemma 5.8 *If P_i, P_j are honest, then eventually $\mathcal{C}_i = \mathcal{C}_j$.*

Proof: Let us look at some $P_l \in \mathcal{C}_i$. If party P_i has placed P_l in $\mathcal{C}_i$ (Step 3) then P_i has completed P_l's A-RS-Share of Step 3. Due to the Termination property of the A-RS protocol we know that eventually P_j will also conclude that A-RS-Share has ended and then P_j will add P_l to $\mathcal{C}_j$. □

Lemma 5.9 *If P_i, P_j are honest, then eventually $\mathcal{E}_i = \mathcal{E}_j$.*

Proof: Let us look at some $P_l \in \mathcal{E}_i$. If party P_i placed P_l in $\mathcal{E}_i$ then P_i has completed the broadcast (`Acceptance set`,P_l,A_l) and $P_l \in \mathcal{C}_i$ and $A_l \subseteq \mathcal{C}_i$. Due to the broadcast property P_j will also complete the above broadcast, and eventually we will have (due to Lemma 5.8) $P_l \in \mathcal{C}_i = \mathcal{C}_j$, and $A_l \subseteq \mathcal{C}_i = \mathcal{C}_j$. Then, P_j will add P_l to $\mathcal{E}_j$. □

Lemma 5.10 (Termination (1):) *If the dealer is honest then each honest party will complete protocol AWSS-Share.*

Proof: All honest parties will eventually be in the sets $\mathcal{C}_i$ and A_i of every honest party P_i. Thus A_i will eventually be of size $2t+1$, and every honest party P_i will broadcast (`Acceptance Set`,P_i,...) in Step 5 of AWSS-Share. Thus all honest parties will eventually be in the set $\mathcal{E}_i$ of every honest party. Consequently, every honest party P_i will broadcast (`Interpolation Set`,P_i,IS_i) in Step 6. Thus every honest party will eventually have $|F_i| \geq 2t+1$ and will complete protocol AWSS-Share. □

Lemma 5.11 (Termination (2)): *If some honest party, P_i, completes protocol AWSS-Share, then each honest party P_j will eventually complete protocol AWSS-Share.*

Proof: Assume that party P_i has completed AWSS-Share. Thus, $|F_i| = n-t$, which means that P_i has completed $n-t$ broadcasts of the form `(Interpolation set,`P_l`,` IS_l`)`, and $IS_l \subseteq \mathcal{E}_i$. Due to the properties of broadcast P_j will eventually complete these invocations of broadcast. Due to Lemma 5.9, we have that eventually $\mathcal{E}_j \subseteq \mathcal{E}_i$, hence, the requirement that $IS_l \subseteq \mathcal{E}_j$ will be satisfied and P_j will complete AWSS-Share. □

Lemma 5.12 (Termination (3)): *If one honest party has completed protocol AWSS-Share, then each honest party will complete protocol AWSS-Rec.*

Proof: Consider an honest party P_i. For each $P_j \in F_i$ we have $P_j \in \mathcal{C}_i$, thus P_i has completed all the A-RS-Share where P_j is the dealer. Thus P_i will complete, in Steps 1(a) and 1(b) of AWSS-Rec, the A-RS-Rec of all the values shared by P_j. Consequently, P_i will associate a value, in Step 2 of AWSS-Rec, with each party $P_j \in F_i$, and will complete AWSS-Rec. □

The Correctness property is proven in Lemma 5.13 through Lemma 5.16.

Lemma 5.13 *Let ϕ_j be the share that party P_j received from the dealer in Step 2 of AWSS-Share. With overwhelming probability, the value which an honest party P_i associates with a party $P_j \in \mathcal{E}_i$ in Step 1(c) of AWSS-Rec is as follows:*

	honest dealer	faulty dealer
P_j honest	ϕ_j	ϕ_j
P_j faulty	ϕ_j *or null*	—

Proof: Party P_i will associate a value ϕ'_j with party P_j if the following two requirements hold:

1. ϕ'_j is the value which was reconstructed by the A-RS-Rec of P_j's share.
2. There are at least $t+1$ parties in A_j for whom $\mathbf{Auth}_{j,*} = \phi'_j$.

Consider first the case where P_j is honest. In this case P_j shared the value ϕ_j in Step 3 of AWSS-Share. From the correctness of A-RS we have that P_i will reconstruct, in Step 1(a) of AWSS-Rec, the correct values of ϕ_j and of P_j's data needed for executing $\mathbf{Auth}_{j,l}$ for each party $P_l \in A_j$. Furthermore, at least $t+1$ out of the parties in A_j are honest. The data of these $t+1$ parties will also be correctly reconstructed, in Step 1(b) of AWSS-Rec. Thus it follows from the properties of ICP that P_i will successfully verify, in Step 1(c), that $\mathbf{Auth}_{j,l} = \phi_l$ for at least $t+1$ parties $P_l \in A_j$, and will associate ϕ_j with P_j.

Consider the case where P_j is faulty and the dealer is honest, and assume that the value ϕ'_j reconstructed in the A-RS-Rec of P_j's share is different than ϕ_j. Party P_i will, in Step 1(c) of AWSS-Rec, asociate a non-null value with P_j only if P_i has $\mathbf{Auth}_{j,l} = \phi'_j$ for at least $t+1$ parties P_l; one of these $t+1$ parties must be honest. We show that if P_l is honest then P_i computes $\mathbf{Auth}_{j,l} = \phi'_j$ only with negligible probability.

Let $\text{I}_{i,j,l}$ (resp., $\text{R}_{i,j,l}$) denote the value that P_i reconstructs, in Step 1(a) (resp., 1(b)) of AWSS-Rec, for P_l's (resp., P_j's) data relevant to $\mathbf{Auth}_{j,l}$. (This data was respectively shared by P_j and P_l, in Step 3 of AWSS-Share. Recall that here P_j acts as the Intermediary in ICP, and P_l acts as the Receiver.) Since P_l is honest, A-RS ensures that $\mathrm{R}_{i,j,l}$ is indeed P_l's data in $\mathbf{Auth}_{j,l}$. Therefore, if it were true that $\mathrm{I}_{i,j,l}$ is determined by the adversary independently of $\mathrm{R}_{i,j,l}$, then Property 3 of ICP (see Section 5.3) would imply that P_i sets $\mathbf{Auth}_{j,l} = \phi'_j$ with negligible probability. However, since P_j is faulty, A-RS ensures only a weaker requirement on $\mathrm{I}_{i,j,l}$. We show that this property suffices.

A-RS allows $\mathrm{I}_{i,j,l}$ to be set by the adversary during the reconstruction stage. However, once the first honest party completes A-RS-Rec, a small set S of $2t+1$ possible values for $\mathrm{I}_{i,j,l}$ is fixed. By the time that S is fixed no honest party has yet started the reconstruction of $\mathrm{R}_{i,j,l}$ (i.e., of the data shared by P_l in Step 3 of AWSS-Share). Thus, $\mathrm{R}_{i,j,l}$ is unknown to the adversary when S is fixed. It follows that the probability that for a given P_i and P_j there exist an honest P_l and a value $\phi'_j \neq \phi_j$ such that P_i computes $\mathbf{Auth}_{j,l} = \phi'_j$ is exponentially small. □

We define the following value, associated with each party P_i who has completed AWSS-Share. Let

$$V_i = \bigcup_{P_j \in F_i} \{(k, \phi_k) \mid \phi_k \text{ is the share of an } \textit{honest} \text{ party } P_k,\ P_k \in IS_j\}.$$

Let

$$r_i = \begin{cases} v_i & \text{if } V_i \text{ defines a secret } v_i \\ null & \text{otherwise} \end{cases}$$

We stress that P_i does not know V_i. Still, V_i is a useful tool in our analysis.

We set the value r from the Correctness requirement to be the value r_i associated with the first honest party P_i who completes AWSS-Share with $r_i \neq$*null*. Note that $|V_i| \geq t+1$, and that V_i (and consequently r_i) are defined once P_i has completed AWSS-Share.

Lemma 5.14 *If the values r_i and r_j of two honest parties P_i and P_j do not equal null, then $r_i = r_j$.*

Proof: Consider a party $P_l \in F_i \cap F_j$. ($F_i \cap F_j \neq \emptyset$ since both F_i and F_j are of size $n-t$.) In IS_l there are at least $t+1$ honest parties, the shares of which define a single secret. Since V_i and V_j define a secret, it must be the same secret defined by the shares in IS_l. Hence, $r_i = r_j$. □

Lemma 5.15 *Every honest party P_i outputs, upon completing AWSS-Rec, either r_i or null.*

Proof: It follows from Lemma 5.13 that the value that P_i associates with each honest party in F_i is ϕ_i, even if the dealer is faulty. Note that F_i contains at least $t+1$ honest parties. Thus if the values associated with the parties in F_i define a secret then this secret is r_i. Consequently P_i outputs either r_i or *null.* □

Lemma 5.16 *If the dealer is honest, sharing a secret s, then every honest party P_i outputs s.*

Proof: It follows from the definition of r_i that if the dealer is honest then $r_i = s$. Furthermore, Lemma 5.13 implies that if the dealer is honest then the values associated with the parties in F_i always define a secret. Since the dealer is honest, P_i does *not* output *null.* It now follows from Lemma 5.15 that P_i outputs s. □

Lemma 5.17 (Secrecy): *If the dealer is honest then the information gathered by the adversary during AWSS-Share is independent of the shared secret.*

Proof: The proof is the same as the proof of Secrecy for the A-RS protocol (Theorem 5.6). We add that the distribution of the information gathered by the adversary in the invocations of A-RS-Share during AWSS-Share is independent of the shared secret. □

5.5.1 Two&Sum-AWSS

In our AVSS scheme we do not use AWSS as such. Instead, we use a slight variation of AWSS, called Two&Sum-AWSS. A dealer shares a secret s, together with a set $\{a_1, \ldots, a_k\}$ of *auxiliary secrets,* in a way that allows the parties to separately reconstruct the secret, any one of the auxiliary secrets, and the sum $s + a_i$ of the secret with any one of the auxiliary secrets. (That is, *several* invocations of the reconstruction protocol will use the shares obtained in a *single* invocation of the sharing protocol.) For each i, reconstructing any single value out of $\{s, a_i, s + a_i\}$ reveals no information on the other two. The correctness property of Two&Sum-AWSS is stated more formally as follows.

Correctness of Two&Sum-AWSS:

1. *If the dealer is honest, sharing $s, \{a_1, \ldots, a_k\}$, then, with probability $1 - \epsilon$, when reconstructing the secret (resp., the ith auxiliary secret, or the sum of the secret and the ith auxiliary secret), each honest party outputs s (resp., a_i, or $s + a_i$).*
2. *Once an honest party completes the sharing protocol, the values*

$$r_s, \{r_{a_1}, \ldots, r_{a_k}\}, \{r_{s+a_1}, \ldots, r_{s+a_k}\} \in \mathcal{F} \cup \{null\}$$

   are fixed.

   (a) *With probability $1 - \epsilon$, when reconstructing the secret (resp., the ith auxiliary secret, or the sum of the secret and the ith auxiliary secret), each honest party outputs either r_s (resp., r_{a_i}, or r_{s+a_i}) or null.*

   (b) *If r_s is null then, for each i, at least one of $\{r_{a_i}, r_{s+a_i}\}$ is null. If r_s, r_{a_i} and r_{s+a_i} are not null then $r_{s+a_i} = r_s + r_{a_i}$.*
3. *Even if for each i one value out of $\{a_i, s + a_i\}$ are reconstructed, the adversary gathers no information about the secret s.*

Our construction of Two&Sum-AWSS, presented in Figure 5-6, is a straightforward generalization of our AWSS scheme. (The scheme is based on the *synchronous* Two&Sum-WSS presented in [TRa].)

The reconstruction protocol is identical to AWSS-Rec, with the addition of parameters specifying which value should be reconstructed. We denote by AWSS-$\mathrm{Rec}_{\mathrm{S}}$ (resp., AWSS-$\mathrm{Rec}_{\mathrm{A}_i}$, AWSS-$\mathrm{Rec}_{\mathrm{S+A}_i}$) the invocation for reconstructing the secret (resp., the ith auxiliary secret, the sum of the secret and the ith auxiliary secret).

The correctness of Two&Sum AWSS is proven in a way similar to the proof of correctness of the AWSS scheme. (Similar proofs appear in [TRa].)

Protocol Two&Sum AWSS-Share$[\epsilon, k]$

Code for the dealer: (on input $s, a_1, \ldots, a_k$ and ϵ)

1. Choose random polynomials $f(\cdot)$ for s and $g_l(\cdot)$ for each a_l. Let $\phi_i \triangleq f(i)$ and $\gamma_{l,i} \triangleq g_l(i)$ for $0 \leq l \leq k$ and $1 \leq i \leq n$. Let $\sigma_{l,i} \triangleq \phi_i + \gamma_{l,i}$.
   Send $\phi_i, \{\gamma_{1,i} \ldots, \gamma_{k,i}\}$ to each party P_i.
2. For every two parties P_i, P_j, invoke: $\mathbf{Gen}_{D,i,j}(\phi_i)$, $\mathbf{Gen}_{D,i,j}(\gamma_{1,i}), \ldots \mathbf{Gen}_{D,i,j}(\gamma_{k,i})$, and $\mathbf{Gen}_{D,i,j}(\sigma_{1,i}), \ldots \mathbf{Gen}_{D,i,j}(\sigma_{k,i})$.

Code for party P_i

3. Invoke A-RS-Share (as a dealer) for each value received from the dealer in Steps 1 and 2. Participate in all other invocations of A-RS-Share.
   Create a dynamic Communication Set, $\mathcal{C}_i$, which is the set of all parties for whom all the invocations of A-RS Share have been completed.
4. For each party P_j invoke $\mathbf{Ver}_{D,i,j}(\phi_i)$, $\mathbf{Ver}_{D,i,j}(\gamma_{1,i}), \ldots \mathbf{Ver}_{D,i,j}(\gamma_{k,i})$, and $\mathbf{Ver}_{D,i,j}(\sigma_{1,i}), \ldots \mathbf{Ver}_{D,i,j}(\sigma_{k,i})$.
   Create an Acceptance Set, A_i. Add party $P_j \in \mathcal{C}_i$ to A_i if all the above invocations of $\mathbf{Ver}_{D,i,j}(*)$ were completed successfully.
5. Execute the rest of the code of AWSS-Share (i.e., Steps 5 through 7 of Figure 5-4).

Figure 5-6: The Two&Sum AWSS Sharing Protocol

5.6 Asynchronous Verifiable Secret Sharing — AVSS

The idea of the AVSS scheme is as follows. First, the dealer sends each party a share of the secret, as in the previous schemes. The parties will then 'commit' to their shares by re-sharing them using AWSS. Next, the parties will make sure, using a cut-and-choose method (as in [CCD, Fe]), that enough AWSS-Sharings have been successful and that the 'committed upon' shares indeed define a secret. We describe the scheme in some more detail. Full details appear in Figure 5-7.

Sharing protocol. The dealer, sharing a secret s, chooses a random polynomial $f(x)$ for s. For each i, the dealer sends $f(i)$ to P_i. In addition, the dealer chooses $k \cdot n \cdot t$ random polynomials $g_{1,1,1}(x), \ldots, g_{k,n,t}(x)$ of degree t, where k is an appropriate security parameter (polynomial in $\log 1/\epsilon$). For each such polynomial $g(x)$, and for each i, the dealer sends $g(i)$ to party P_i. Upon receiving all the expected values from the dealer, each P_i re-shares the values $f(i), \{g_{1,1,1}(i), \ldots, g_{k,n,t}(i)\}$ using Two&Sum-AWSS-Share with the appropriate allowed-error parameter (see Section 5.5.1).

Next, the dealer proves to the parties that their shares indeed define a secret. Each party P_i is looking for a set IS_i of at least $n-t$ parties, such that the Two&Sum-AWSS-Share's of these parties have been completed, and the corresponding values define a polynomial. For this purpose P_i participates in up to t iterations, as follows.[3] Initially, all parties are valid. At the beginning of each iteration P_i waits until he has completed the Two&Sum-AWSS-Share of $n-t$ valid parties. Let $IS_{i,r}$ denote this set of parties, in iteration r. Next P_i verifies, using cut-and-choose as described in the code, that the share of each party in $IS_{i,r}$ is valid (in a sense defined in Figure 5-7). P_i removes, from the set of valid parties, all the parties in $IS_{i,r}$ whose validation failed. If the validity of the shares of at least $n-t$ parties in $IS_{i,r}$ is confirmed, then we say that the iteration was successful. In this case, P_i broadcasts (P_i `confirms` P_j) for each party $P_j \in IS_{i,r}$. (Now P_i is assured that the shares of the parties in $IS_{i,r}$ define a polynomial.) When the verification procedure is completed, and P_i has completed the Two&Sum-AWSS-Share of at least one other valid party, P_i proceeds to the next iteration.

Let F_i be P_i's set of parties who were confirmed by at least $t+1$ parties. Party P_i completes the sharing protocol once $|F_i| \geq 2t+1$.

Reconstruction protocol. The reconstruction protocol is simple: The share of each party P_i is reconstructed using Two&Sum-AWSS-Rec$_{i,s}$.[4] Once $t+1$ shares of parties in F_i are reconstructed with a non-null value, party P_i computes the secret defined by these shares, and terminates.

Protocols AVSS-Share and AVSS-Rec are presented in Figures 5-7 and 5-8, respectively.

Theorem 5.18 *Let $n \geq 3t+1$. Then for every $\epsilon > 0$, the pair* (AVSS-Share[ϵ],AVSS-Rec[ϵ]) *is a $(1-\epsilon)$-correct, t-resilient AVSS scheme for n parties.*

[3] Partitioning the protocol into t iterations is done for clarity of exposition. All these iterations are completed in a constant number of asynchronous time steps.

[4] We let Two&Sum-AWSS-Rec$_{i,*}$ denote an invocation of Two&Sum-AWSS-Rec$_*$ that corresponds to the Two&Sum-AWSS-Share of P_i.

Protocol AVSS-Share[ϵ]

Code for the dealer, on parameter ϵ and secret s:

1. Choose a random polynomial $f(x)$ for s. Send $\phi_i \triangleq f(i)$ to each party P_i.
2. Set $k \triangleq O(n, \log \frac{1}{\epsilon})$. For $1 \le l \le k$, for $1 \le j \le n$, and for $1 \le r \le t$ do: Pick a random polynomial $g_{l,j,r}(x)$ of degree t, and send $g_{l,j,r}(i)$ to each party P_i.
3. Upon completing an broadcast `(Random vector,`$b_{1,j,r}, ..., b_{k,j,r}$`,iteration `$r$`,party,`$P_j$`)` (see Step 6a), broadcast `(Polynomials of `P_j`, iteration `r`, `$g_{1,i,r}(\cdot) + f(\cdot) \cdot b_{1,i,r}, ..., g_{k,i,r}(\cdot) + f(\cdot) \cdot b_{k,i,r}$`)`.

Code for party P_i, on parameter ϵ:

4. Set $\epsilon' = \epsilon/2n$. Share $\phi_i, \{g_{1,1,1}(i), \ldots, g_{k,n,t}(i)\}$ using Two&Sum-AWSS-Share[$\epsilon', n \cdot k \cdot t$] (see Figure 5-6). Participate in the Two&Sum-AWSS-Share of other parties.
5. Create a dynamic set $\mathcal{C}_i$. Add each party whose Two&Sum-AWSS-Share of the previous step has been completed to $\mathcal{C}_i$.

   Let V_i be the set of valid parties. Initially $V_i = \{1, \ldots, n\}$.
6. A local variable r is set to 0. Initialize sets $IS_{i,r} = F_i = \emptyset$.
   As long as $|F_i| < 2t + 1$, do:

   (a) Wait until $|\mathcal{C}_i \cap V_i| \ge n - t$ and $\mathcal{C}_i \cap V_i \ne IS_{i,r}$. Set $r := r+1$. Let $IS_{i,r} := \mathcal{C}_i \cap V_i$. Choose $b_{1,i,r}, \ldots, b_{k,i,r} \in_{\mathrm{R}} \{0,1\}$, and broadcast `(Random vector,`$b_{1,i,r}, ..., b_{k,i,r}$`, iteration `$r$`,party,`$P_i$`)`.

   (b) Upon completing an broadcast `(Polynomials of `P_j`,...)` (see Step 3), for $l = 1..k$, $m = 1..n$, if $b_{l,j,r} = 0$ execute Two&Sum-AWSS-Rec$_{m,\mathrm{A}_{l,j,r}}$. If $b_{l,j,r} = 1$ then execute Two&Sum-AWSS-Rec$_{m,\mathrm{S}+\mathrm{A}_{l,j,r}}$.

   (c) If any Two&Sum-AWSS-Rec of a share of party P_m terminates with *null*, or if any reconstructed share of P_m does not agree with the corresponding polynomial broadcasted by the dealer, then P_m is removed from V_i and from $IS_{i,r}$.

   Once all the relevant invocations of Two&Sum-AWSS-Rec of shares of parties in $IS_{i,r}$ are completed, if $|IS_{i,r}| \ge n - t$ then for each $P_m \in IS_{i,r}$ broadcast `(`P_i`,confirms party,`P_m`)`. (In this case we say that the iteration was successful.)

   * Once there are $t + 1$ broadcasts of the form `(...,confirms party,`P_m`)` for some P_m (see Step 6c) then add P_m to F_i.

Figure 5-7: The AVSS Sharing Protocol

Protocol AVSS-Rec[ϵ]

Code for party P_i, on parameter ϵ:

1. For each party P_j execute Two&Sum-AWSS-Rec$_{j,s}$. (P_j's share, ϕ_j, was shared by P_j in the Two&Sum-AWSS-Share of Step 4 of AVSS-Share.)
2. Take any $t+1$ reconstructed non-null shares of parties in F_i, and output the secret defined by these shares.

Figure 5-8: The AVSS Reconstruction Protocol

For convenience and clarity we partition the proof of Theorem 5.18 to several lemmas. Throughout the proof we assume that the following event E occurs. All invocations of Two&Sum-AWSS have been properly completed. That is, if an honest party has completed Two&Sum-AWSS-Share then all honest parties will complete all corresponding invocations of AWSS-Rec, outputting either the required value or *null*. (See the Correctness requirement of Two&Sum-AWSS, on page 115.) Event E occurs with probability at least $1-n\epsilon' = 1-\epsilon/2$.

Lemma 5.19 (Termination (1)): *If the dealer is honest then each honest party will complete protocol AVSS-Share.*

Proof: Each honest party P_i will complete the Two&Sum-AWSS-Share of the shares of each honest party P_j in Step 4. Thus P_j will eventually be in the set $IS_{i,r}$ of P_i for some iteration r. If the dealer is honest then P_j's reconstructed shares will agree, in Step 6c, with the polynomials broadcasted by the dealer. Thus, P_j will not be removed from $IS_{i,r}$ or V_i. Hence there will exist an iteration r where all $n-t$ honest parties are in $IS_{i,r}$, causing each honest party P_i to broadcast `(`P_i`,confirms,`P_j`)`. Consequently, each honest party will be in the final set F_k of each honest P_k. Thus P_k will have $|F_k| \geq 2t+1$ and will complete AVSS-Share. We note that all honest parties terminate in a constant number of (asynchronous) rounds. □

Lemma 5.20 (Termination (2)): *If some honest party P_i completes protocol AVSS-Share, then each honest party will eventually complete protocol AVSS-Share.*

Proof: Assume an honest party P_i completed protocol AVSS-Share. Then $|F_i| \geq 2t+1$. It follows from the correctness properties of broadcast that, for any honest party P_j, any party in F_i will eventually be in F_j. Thus P_j will complete AVSS-Share. □

Lemma 5.21 (Termination (3)): *If AVSS-Share has been completed by the honest parties, then each honest party will complete protocol AVSS-Rec.*

Proof: Each honest party P_i has at least $t+1$ honest parties in F_i. It follows from the correctness of Two&Sum-AWSS that the share of each honest party $P_j \in F_i$ will be successfully reconstructed by P_i in Step 1 of AVSS-Rec. Thus, P_i will have at least $t+1$ non-null shares in Step 2 of AVSS-Rec, and will complete the protocol. □

Lemma 5.22 (Secrecy): *If the dealer is honest then the information gathered by the adversary during AVSS-Share is independent of the shared secret.*

Proof: Assuming that Two&Sum-AWSS-Share is secure, the relevant information gathered by the adversary in AVSS-Share consists of up to t shares of the polynomial $f(\cdot)$ shared by the dealer in Step 1, and up to t shares of each of the random polynomials shared in Step 2. Furthermore, for each random polynomial $g_*(\cdot)$, only one of $g_*(\cdot)$ or $f(\cdot) + g_*(\cdot)$ is known. It can be verified that this information is distributed independently of the shared secret. □

The Correctness property is proven via Lemmas 5.23 through 5.25.

Lemma 5.23 *Once an honest party P_i executes Step 6a of AVSS-Share in iteration r, a value $\phi'_m \in \mathcal{F} \cup null$ is fixed for each $P_m \in IS_{i,r}$. Furthermore, if iteration r was successful (as defined in Step 6c of AVSS-Share) then the following properties hold with overwhelming probability:*

1. *For each $P_m \in IS_{i,r}$, $\phi'_m \neq null$.*
2. *The set $C_{i,r} \triangleq \{(m, \phi'_m) | P_m \in IS_{i,r}\}$ defines a secret s'.*
3. *If the dealer is honest then s' is the secret s shared by the dealer.*
4. *When reconstructing P_m's share using Two&Sum-AWSS-Rec$_{m,\text{S}}$ in Step 1 of AVSS-Rec, each honest party outputs $\phi'_m \cup null$.*

Proof: Define the value ϕ'_m to be the value fixed for the secret shared by P_m in the Two&Sum-AWSS-Share of Step 4. Once P_i executes Step 6a, in iteration r, he has completed the Two&Sum-AWSS-Share of each $P_m \in IS_{i,r}$. It follows from the correctness of Two&Sum-AWSS that the value ϕ'_m is fixed. Part 4 of the lemma also follows.

Part 1. It follows from the correctness of Two&Sum-AWSS that if ϕ'_m $=$*null* then for each $l = 1..k$ at least one out of {Two&Sum-AWSS-Rec$_{m,\text{A}_{l,i,r}}$, Two&Sum-AWSS-Rec$_{m,\text{S}+\text{A}_{l,i,r}}$} will have output *null*. Thus for each $l = 1..k$ P_i will have *null* output with probability at least $\frac{1}{2}$. The probability that P_i has, in Step 6c, non-null output of all the k invocations of Two&Sum-AWSS-Rec is at most 2^{-k}. Thus, executions where ϕ'_m $=$*null* and iteration r is successful occur only with negligible probability.

Part 2. We use the same cut-and-choose argument as in part 1. Assume that $C_{i,r}$ does not define a secret. It follows that for each $l = 1..k$ there exists a P_m such that either the output of Two&Sum-AWSS-Rec$_{m,\text{A}_{l,i,r}}$ is not equal to $h_{l,i,r}(m)$ or the output of Two&Sum-AWSS-Rec$_{m,\text{S}+\text{A}_{l,i,r}}$ is not equal to $h_{l,i,r}(m)$ (where $h_{l,i,r}$ is the corresponding polynomial broadcasted by the dealer). Thus for each $l = 1..k$ P_i will detect an error with probability at least $\frac{1}{2}$. The probability that P_i has not detected an error in all the k invocations of Two&Sum-AWSS-Rec in Step 6c is at most 2^{-k}. Thus, iteration r is successful only with negligible probability.

Part 3. The correctness of Two&Sum-AWSS assures that the value reconstructed in Two&Sum-AWSS-Rec$_{m,\text{S}+\text{A}_{l,i,r}}$ equals the sum of the values reconstructed in Two&Sum-AWSS-Rec$_{m,\text{A}_{l,i,r}}$ and in Two&Sum-AWSS-Rec$_{m,\text{S}}$. Since the dealer is honest, he broadcasts, in Step 3 of AVSS-Share, the same polynomials that he shared in Step 1. Part 3 follows using the same cut-and-choose argument as in parts 1 and 2. □

Lemma 5.24 *Let $C \triangleq \{(m, \phi'_m) | P_m \in F_i$ and P_i is honest$\}$, where ϕ'_m is the value fixed for P_m (See Lemma 5.23). Then the following properties hold with overwhelming probability:*

1. *The set C defines a secret s'.*

2. *If the dealer is honest then s' is the shared secret, s.*

Proof: **1.** Consider two parties P_{m_1} and P_{m_2} such that $(m_1, \phi'_{m_1}), (m_2{}', \phi'_{m_2}) \in C$. Then both P_{m_1} and P_{m_2} were confirmed by honest parties. Assume P_{m_1} was confirmed by honest party P_{j_1} in iteration r_1, and P_{m_2} was confirmed by honest party P_{j_2} in iteration r_2. Let $C_{j_1,r_1} \triangleq \{(m, \phi'_m) | P_m \in IS_{j_1,r_1}\}$, and let $C_{j_2,r_2} \triangleq \{(m, \phi'_m) | P_m \in IS_{j_2,r_2}\}$. Then the intersection $I \triangleq C_{j_1,r_1} \cap C_{j_2,r_2}$ is of size at least $t+1$. Lemma 5.23 implies that the values in I are non-null with overwhelming probability, even if the corresponding parties are faulty. It follows that C_{j_1,r_1} and C_{j_2,r_2} define the same secret. Part 1 follows.
Part **2** follows from part 3 of Lemma 5.23. □

Lemma 5.25 *Let r denote the secret defined by the set C (see Lemma 5.24). An honest party has output different than r of AVSS-Rec only with negligible probability.*

Proof: Consider an honest party P_i. The set $\{(m, \phi'_m) | P_m \in F_i\}$ is a subset of C. Thus it defines the same secret r. P_i will have at least $t+1$ honest parties P_j in F_i; the corresponding reconstructed values ϕ'_j will be non-null. Thus P_i will be able to interpolate a value, and this value will be r. □

5.7 Common Coin

We define an asynchronous common coin primitive, and describe an construction. We employ the AVSS scheme described in Section 5.6. We first present a definition of a common coin primitive.

Definition 5.26 *Let π be a protocol, where each party has local random input and binary output. We say that π is a $(1-\epsilon)$-*terminating*, t-*resilient Common Coin* protocol if the following requirements hold for every t-adversary:*

- **Termination.** *With probability $1-\epsilon$, all the honest parties terminate.*

- **Correctness.** *For every value $\sigma \in \{0, 1\}$, with probability at least $\frac{1}{4}$ all the honest parties output σ.*

Our construction (with 'termination parameter' ϵ). Roughly speaking, the protocol consists of two stages. First, each party shares n random secrets, using the AVSS-Share protocol of our AVSS scheme, with allowed error parameter $\epsilon' \triangleq \frac{\epsilon}{n^2}$. Say that the ith secret shared by each party is assigned to party P_i. Once a party, P_i, completes $t+1$ AVSS-Share protocols of secrets assigned to him, he broadcasts the identity of the dealers of these secrets. We say that these $t+1$ secrets are attached to P_i. (Later, the value associated with P_i will be computed based on the secrets attached to him.)

Upon completing the AVSS-Share of all the secrets attached to some P_j, party P_i is certain that a fixed (and yet unknown) value is attached to P_j. (The way in which this value will be computed is described in the protocol.) Once P_i is assured that the values attached to enough parties has been fixed, he starts reconstructing the relevant secrets. (This process of ensuring that enough values have been fixed is at the heart of the protocol.) Once all the relevant secrets are reconstructed, each party locally computes his output based on the reconstructed secrets, in a special way described in the sequel. The protocol is presented in Figure 5-9 below.

Protocol Common-Coin$[\epsilon]$

Code for party P_i, given 'termination' parameter, ϵ:

1. Let $\epsilon' \triangleq \frac{\epsilon}{n^2}$. For $1 \le j \le n$, choose a random secret $x_{i,j} \in_R \mathcal{F}$ and execute AVSS-Share for this value. Denote the execution by AVSS-Share$_{i,j}(x_{i,j})$.
   Participate in all the other AVSS-Share protocols.
2. Create a dynamic communication set $\mathcal{C}_i$. Add party P_j to $\mathcal{C}_i$ if AVSS-Share$_{j,l}$ has been completed for all $1 \le l \le n$.
   Wait until $|\mathcal{C}_i| \ge t+1$. Then, set $C_i = \mathcal{C}_i^{(t+1)}$ and broadcast `(Attach` A_i `to` P_i`)` .
   (We say that the secrets $\{x_{j,i} | P_j \in C_i\}$ are the secrets attached to party P_i.)
3. Accept a party P_j if the broadcast `(Attach` A_j `to` P_j`)` has been completed, and $C_j \subseteq \mathcal{C}_i$. Let $\mathcal{G}_i$ be the dynamic set of accepted parties.
   Wait until $|\mathcal{G}_i| \ge n-t$. Then, let $G_i = \mathcal{G}_i^{(n-t)}$ and broadcast `(`P_i `accepts` H_i`)` .
4. Say that party P_j is supportive, if the `(`P_j `accepts` H_j`)` broadcast has been received, and each party in G_j is accepted (namely, if $G_j \subseteq \mathcal{G}_i$).
   Wait until $n-t$ parties are supportive. Then, raise flag `'reconstruct enabled'` .
   Let H_i denote the current contents of $\mathcal{G}_i$.
   (Note that a party P_j who was not considered supportive since some $P_k \in G_j$ was not in $\mathcal{G}_i$ can become supportive later if P_k is added to $\mathcal{G}_i$.)
5. Wait until the flag `'reconstruct enabled'` is raised. Then, reconstruct the secrets attached to all the accepted parties. That is, for each $P_k \in C_j$ such that $P_j \in \mathcal{G}_i$ invoke AVSS-Rec$_{k,j}$, and let $r_{k,j}$ be the corresponding output.
   (Note that some parties may become accepted *after* the flag `'reconstruct enabled'` has been raised. The corresponding AVSS-Rec protocols are invoked immediately.)
6. Let $u \triangleq \lceil 0.87n \rceil$. For every party $P_j \in \mathcal{G}_i$, let V_j, the value associated with P_j, be the sum modulo u of all the secrets attached to P_j. That is, $V_j = (\sum_{k \in C_j} r_{k,j}) \bmod u$.
7. Wait until the values associated with all the parties in G_i are computed. If there exists a party $P_j \in H_i$ where $V_j = 0$, output 0.
   Otherwise, output 1.

Figure 5-9: The Common Coin protocol

Theorem 5.27 *Let $n \ge 3t+1$. Then, for every $0 < \epsilon \le 0.2$ protocol* Common-Coin *is a $(1-\epsilon)$-terminating, t-resilient common coin protocol for n parties.*

The Termination property is asserted in Lemma 5.28. The Correctness property is asserted in Lemmas 5.29 through 5.31. Throughout the proof we assume that the following event E occurs. All invocations of AVSS have been properly completed. That is, if an honest party has completed AVSS-Share then a value s' is fixed. All honest parties will complete the corresponding invocation of AVSS-Rec, outputting s'. If the dealer is honest then s' is the shared secret. (See Definition 5.2.) Event E occurs with probability at least $1 - n^2\epsilon' = 1 - \epsilon$.

Lemma 5.28 *All the honest parties complete protocol* Common-Coin[ϵ] *in constant time.*

Proof: Assume event E occurs, and let P_i be an honest party. Then, P_i will complete all the AVSS-Share protocols initiated by honest parties in Step 1. Thus, $\mathcal{C}_i$ will eventually be of size $t+1$, and Step 2 will be completed.

For every honest party P_j, the broadcast `(Attach` A_j `to` P_j`)` will be received by P_i. Furthermore, since P_j completed the AVSS-Share$_{k,j}$ protocol for every $P_k \in C_j$, then P_i will complete these AVSS-Share$_{k,j}$ protocols as well. Therefore, every honest party will eventually be considered accepted by P_i (namely, added to the set $\mathcal{G}_i$). Thus, $\mathcal{G}_i$ will eventually be of size $n-t$, and Step 3 will be completed. Similar reasoning implies that every honest party will eventually be considered supportive by every honest party in Step 4. Consequently, every honest party will raise his `'reconstruct enabled'` flag, and will invoke his AVSS-Recs of Step 5.

It remains to be shown that all the AVSS-Rec protocols invoked by each honest party will be completed. If an honest party received an `(Attach` A_j `to` P_j`)` broadcast, then all the honest parties will receive this broadcast. Thus, if an honest party invokes AVSS-Rec$_{j,k}$, then all the honest parties will invoke AVSS-Rec$_{j,k}$. Event E now assures us that all the honest parties will complete all their AVSS-Rec protocols. Therefore, all the honest parties will execute Step 6 and complete the protocol. (The invocations of AVSS-Share with faulty dealers need not be terminated. Once an honest party completes Step 6, he may abort all non-terminated invocations of AVSS-Share.)

Given event E, all invocations of AVSS-Share and AVSS-Rec terminate in constant time. Protocol broadcast terminates in constant time. Thus, protocol Common-Coin terminates in constant time as well. □

Lemma 5.29 *Let* $u \triangleq \lceil 0.87n \rceil$. *Let* P_i *be a party whose broadcast* `(Attach` A_i `to` P_i`)` *in Step 2 has been completed by some honest party. Then, there exists a value,* v_i*, such that all the honest parties associate* v_i *with* P_i *in Step 6. Furthermore,*

- v_i *is fixed once the first honest party has completed the* `(Attach` A_j `to` P_j`)` *broadcast.*
- v_i *is distributed uniformly over* $[1 \ldots u]$, *and is independent of the values associated with the other parties.*

Proof: Let P_i be a party whose `(Attach` A_i `to` P_i`)` broadcast of Step 2 has been completed by some honest party. Then, all the honest parties will complete this broadcast with output C_i. Furthermore, The definition of AVSS assures us that for each party $P_j \in C_i$, there exists a fixed value, $r_{j,i}$, such that all the honest parties reconstruct have $r_{j,i}$ as their output of AVSS-Rec$_{j,i}[\epsilon']$ in Step 5 (that is, $r_{j,i}$ is the value shared by P_j, and attached to P_i.) Consequently, the value that each honest party associates with P_i in Step 6 is $\sum_{P_j \in C_i} r_{j,i} \bmod u$; let v_i be this value. This value is fixed once C_i is fixed, namely by the time that the first honest party has completed the broadcast `(Attach` A_j `to` P_j`)` .

In remains to show that v_i is uniformly distributed over $[1 \ldots u]$, and is independent of the values associated with the other parties. Recall that an honest party starts reconstructing the secrets attached to P_i (namely, invokes the AVSS-Rec$_{j,i}[\epsilon']$ protocols for $P_j \in C_i$) only *after* it completes the `(Attach` A_i `to` P_i`)` broadcast. Namely, the set C_i is fixed *before* any honest party invokes an AVSS-Rec$_{k,i}$ for some k. The Privacy property of AVSS now assures us that the bad parties have no information on the secrets shared by the honest parties when the set C_i is fixed. Thus, the set C_i, as well as the values that were shared by bad parties, are independent of the values shared by honest parties. Furthermore, each set C_i contains at least one honest party, and honest parties share uniformly distributed, mutually independent values. Consequently, the sum v_i is uniformly and independently distributed over $[1 \dots u]$. □

Lemma 5.30 *Assume that some honest party has raised the* `'reconstruct enabled'` *flag. Then there exists a set, M, such that:*

1. *For each party $P_j \in M$, the* `(Attach` A_j `to` P_j`)` *broadcast of Step 2 has been completed by some honest party. (Note that Lemma 5.29 applies to each $P_j \in M$.)*
2. *When any honest party, P_j, raises his* `'reconstruct enabled'` *flag, it will hold that $M \subseteq H_j$.*
3. *$|M| \geq \frac{n}{3}$.*

Proof: Let P_i be the first honest party to raise his `'reconstruct enabled'` flag. Let M be the set of parties, P_k, for whom $P_k \in G_l$ for at least $t+1$ parties P_l who are considered supportive by P_i, in Step 4. We show that the set M has the properties required in the Lemma.

Clearly, $M \subseteq H_i$. Thus, party P_i has received the broadcast `(Attach` A_k `to` P_k`)` of every party $P_k \in M$. This asserts the first property of M. We now assert the second property. Let $P_k \in M$. An honest party P_j raises his `'reconstruct enabled'` flag, when he has found at least $n-t$ parties who are supportive in Step 4. However, $P_k \in G_l$ for at least $t+1$ of the `(`P_l `accepts` H_l`)` broadcasts; thus, there must exist a party P_l such that $P_k \in G_l$ and $G_l \subseteq H_j$. Consequently, $P_k \in H_j$.

It remains to show that $|M| \geq \frac{n}{3}$. We use a counting argument. Let $m \triangleq |H_i|$. We have $m \geq n-t$. Consider the $m \times n$ table T (relative to party P_i), where $T_{l,k} =$ `one` iff P_i has received the `(`P_l `accepts` H_l`)` broadcast and $P_k \in G_l$. The set M is the set of parties P_k such that the kth column in T has at least $t+1$ `one` entries. There are $(n-t)$ `one` entries in each row of T; thus, there are $m \cdot (n-t)$ `one` entries in T.

Let q denote the minimum number of columns in T that contain at least $t+1$ `one` entries. We show that $q \geq \frac{n}{3}$. Clearly, the worst distribution of the `one` entries in this table is letting q columns be all `one`'s (namely, each of the q columns has m `one` entries), and letting each of the remaining $(n-q)$ columns have t `one` entries. This distribution requires that the number of `one` entries be no more than $q \cdot m + (n-q) \cdot t$. However, there are $m \cdot (n-t)$ `one` entries in T. Thus, we must have:

$$q \cdot m + (n-q) \cdot t \geq m \cdot (n-t)$$

or, equivalently, $q \geq \frac{m(n-t)-nt}{m-t}$. Since $m \geq n-t$ and $n \geq 3t+1$, we have

$$q \geq \frac{m(n-t)-nt}{m-t} \geq \frac{(n-t)^2-nt}{n-2t} \geq \frac{(n-2t)^2+nt-3t^2}{n-2t} \geq n-2t+\frac{nt-3t^2}{n-2t} \geq n-2t+\frac{t}{n-2t} > \frac{n}{3}.$$

□

Lemma 5.31 *Let $\epsilon \leq 0.2$, and assume that all the honest parties have completed protocol* Common-Coin[ϵ]. *Then, for every value $\sigma \in \{0,1\}$, with probability at least 0.25, all the honest parties output σ.*

Proof: By Lemma 5.29 we have that for every party, P_j, who is accepted by some honest party, there exists a value, v_j, distributed uniformly and independently over $[1 \ldots u]$, such that with probability $1-\frac{\epsilon}{n}$ all the honest parties associate v_j with P_j in Step 6. Consequently, with probability $1-\epsilon$, all the honest parties agree on the value associated with each one of the parties.)

Consider the case $\sigma = 0$. Let M be the set of parties guaranteed by Lemma 5.30. Clearly, if $v_j = 0$ for some party $P_j \in M$ and all the honest parties associate v_j with P_j, then all the honest parties output 0. The probability that at least one party $P_j \in M$ has $v_j = 0$ is $1-(1-\frac{1}{u})^{|M|}$. Recall that $u = \lceil 0.87n \rceil$, and that $|M| \geq \frac{n}{3}$. Therefore, for all $n > 4$ we have $1-(1-\frac{1}{u})^{|M|} \geq 1-\mathrm{e}^{-0.38} \geq 0.316$. Thus, Prob(all the honest parties output 0) $\geq 0.316 \cdot (1-\epsilon) \geq 0.25$.

Consider the case $\sigma = 1$. Clearly, if no party P_j has $v_j = 0$ (and all the honest parties associate v_j with every P_j), then all the honest parties output 1. The probability of this event is at least $(1-\frac{1}{u})^n(1-\epsilon) \geq \mathrm{e}^{-1.15} \cdot 0.8 \geq 0.25$. Thus, Prob(all the honest parties output 1) ≥ 0.25. □

5.8 Byzantine Agreement

Before describing the Byzantine Agreement protocol, let us describe another protocol used in our construction. Roughly speaking, this protocol, denoted Vote, does 'whatever can be done deterministically' to reach agreement.

5.8.1 The Voting Protocol

In protocol Vote each party tries to find out whether there is a detectable majority for some value among the (binary) inputs of the parties. More precisely, each party's output of the protocol can take five different values. For $\sigma \in \{0,1\}$, the output $(\sigma, 2)$ stands for 'overwhelming majority for σ'. Output $(\sigma, 1)$ stands for 'distinct majority for σ'. Output $(\Lambda, 0)$ stands for 'non-distinct majority'. It will be shown that if all the honest parties have the same input, σ, then all honest parties output $(\sigma, 2)$. Furthermore, if some honest party outputs $(\sigma, 2)$ then every honest party will output either $(\sigma, 2)$ or $(\sigma, 1)$. If some honest party outputs $(\sigma, 1)$ then either all outputs are in $\{(\sigma, 2), (\sigma, 1)\}$, or all outputs are in $\{(\Lambda, 0), (\sigma, 1)\}$.

The protocol consists of three 'rounds', having similar structure. In the first round, each party broadcasts his input value, waits to complete $n-t$ broadcasts of other parties, and sets his vote to the majority value among these inputs. In the second round, each party broadcasts his vote (along with the identities of the $n-t$ parties whose broadcasted inputs were used to compute the vote), waits to complete $n-t$ broadcasts of other votes that are consistent with the broadcasted inputs of the first round, and sets his re-vote to the majority value among these votes. In the third round each party broadcasts his re-vote, along with the identities of the $n-t$ parties whose broadcasted votes were used to compute the re-vote, and waits to complete $n-t$ broadcasts of other re-votes that are consistent with the consistent votes of the second round.

Now, if all the consistent votes received by a party agree on a value, σ, then this party outputs $(\sigma, 2)$. Otherwise, if all the consistent re-votes received by the party agree on a value, σ, then the party outputs $(\sigma, 1)$. Otherwise, the party outputs $(\Lambda, 0)$.

Protocol Vote is presented in Figure 5-10 below.

Protocol Vote(x_i)

Code for party P_i, on input x_i:

1. Broadcast $(\texttt{Input}, P_i, x_i)$.
2. Define a dynamic set $\mathcal{A}_i$. Add (P_j, x_j) to $\mathcal{A}_i$ if the $(\texttt{Input}, P_j, x_j)$ broadcast is completed.

   Wait until $|\mathcal{A}_i| \geq n-t$. Set A_i to $\mathcal{A}_i^{(n-t)}$. Then, set v_i to the majority bit among $\{x_j | (P_j, x_j) \in A_i\}$, and broadcast $(\texttt{Vote}, P_i, A_i, v_i)$.
3. Define a dynamic set $\mathcal{B}_i$. Add (P_j, A_j, v_j) to $\mathcal{B}_i$ if the $(\texttt{Vote}, P_j, A_j, v_j)$ broadcast has been completed, $a_j \subseteq \mathcal{A}_i$, and v_j is the majority bit of A_j.

   Wait until $|\mathcal{B}_i| \geq n-t$. Set B_i to $\mathcal{B}_i^{(n-t)}$. Then, set rv_i to the majority bit among $\{v_j \; | (P_j, \text{vote}_j, A_j) \in B_i\}$, and broadcast $(\texttt{Re-vote}, P_i, B_i, rv_i)$.
4. Define a set C_i. Add (P_j, B_j, rv_j) to C_i if the $(\texttt{Re-vote}, P_j, B_j, rv_j)$ broadcast has been completed, $B_j \subseteq \mathcal{B}_i$, and rv_j is the majority bit of B_j.
5. Wait until $|C_i| \geq n-t$.
   If all the parties $P_j \in B_i$ had the same vote $v_j = \sigma$, then output $(\sigma, 2)$ and terminate. Otherwise, if all the parties $P_j \in C_i$ have the same re-vote, $rv_j = \sigma$, then output $(\sigma, 1)$ and terminate.

   Otherwise, output $(\Lambda, 0)$ and terminate.

Figure 5-10: The Vote protocol

In Lemmas 5.32 through 5.35 we assert the properties of protocol Vote, as describe above. The Lemmas hold for every input and every t-adversary.

Lemma 5.32 *All the honest parties terminate protocol Vote in constant time.*

Proof: The $(\texttt{Input}, P_j, x_j)$ broadcast of every honest party P_j in Step 1 will be completed. Thus, every honest party P_i will eventually have $|\mathcal{A}_i| = n-t$, in Step 2, and will broadcast $(\texttt{Vote}, P_i, A_i, v_i)$. For every honest party P_j the broadcast of Step 2 will be completed. An honest party P_i will add (P_j, A_j, v_j) to $\mathcal{B}_i$ for each honest P_j, in Step 3. Thus, every honest party P_i will eventually have $|\mathcal{B}_i| = n-t$, in Step 3, and will broadcast $(\texttt{Re-vote}, P_i, B_i, rv_i)$. Similarly, P_i will add every honest party P_j to C_i. Thus, every honest party P_i will eventually have $|C_i| = n-t$, in Step 5. Consequently, P_i will complete the protocol. We note that the protocol runs in constant time. □

Lemma 5.33 *If all the honest parties have input σ, then all the honest parties output $(\sigma, 2)$.*

Proof: Consider an honest party P_i. If all the honest parties have input σ, then at most t parties will broadcast $\bar{\sigma}$ as their input in Step 1. Therefore, each party P_k who was added to $\mathcal{B}_i$ in Step 3 has a majority for the value σ in his A_k set, and $v_k = \sigma$. Thus, P_i outputs $(\sigma, 2)$ in Step 5. □

Lemma 5.34 *If some honest party outputs* $(\sigma, 2)$*, then each honest party outputs either* $(\sigma, 2)$ *or* $(\sigma, 1)$.

Proof: Assume that honest party P_i outputs $(\sigma, 2)$. The size of B_i is $n - t$, hence for each other honest party P_j it holds that $B_i \cap B_j$ is of size at least $t + 1$. Thus, P_j will set his revote rv_j to σ. Therefore, every honest party outputs either $(\sigma, 2)$ or $(\sigma, 1)$ in Step 5. □

Lemma 5.35 *If some honest party outputs* $(\sigma, 1)$*, and no honest party outputs* $(\sigma, 2)$*, then each honest party outputs either* $(\sigma, 1)$*, or* $(\Lambda, 0)$.

Proof: Assume some honest party outputs $(\sigma, 1)$. Then, at most t parties P_j broadcasted a revote $rv_j = \bar{\sigma}$ in Step 3; therefore, no honest party P_j has a unanimous $\textsf{re-vote}_j = \bar{\sigma}$ in Step 4, and no honest party outputs $(\bar{\sigma}, 1)$. Furthermore, at least $t + 1$ parties P_k have broadcasted $\textsf{vote}_k = \sigma$ in Step 2; therefore, no honest party had a unanimous $\textsf{vote}$ in Step 3, and no honest party outputs $(\bar{\sigma}, 2)$. □

5.8.2 The Byzantine Agreement protocol

The Byzantine Agreement protocol proceeds in iterations. In each iteration, each party has a 'modified input' value; in the first iteration, the modified input of each party is his local input. In each iteration the parties invoke two protocols: Vote and Common-Coin. Protocol Common-Coin is invoked only *after* protocol Vote is completed. (The reason for this provision will become clear in the proof of Lemma 5.38 below.) If a party recognizes a 'distinct majority' for some value, σ, in the output of protocol Vote (namely output $(\sigma, 1)$ or $(\sigma, 2)$), then he sets his modified input for the next iteration to σ. Otherwise, he sets his modified input for the next iteration to be the output of the Common-Coin protocol. (Protocol Common-Coin is invoked by all parties in each iteration, regardless of whether their output is used.) Once a party recognizes an 'overwhelming majority' for some value, σ, (namely, output $(\sigma, 2)$ of protocol Vote), he broadcasts σ. Once a party completes $t + 1$ broadcasts for some value, σ, he terminates with output σ.

The code of the Byzantine Agreement protocol is presented in Figure 5-11 below.

In Lemmas 5.36 through 5.40 we assert the validity of protocol BA. These Lemmas hold for every input and every t-adversary.

Lemma 5.36 *If all the honest parties have input* σ*, then all the honest parties terminate and output* σ.

Proof: Assume that all the honest parties have input σ. By Lemma 5.33, every honest party P_i has $(y_1, m_1) = (\sigma, 2)$ by the end of Step 1 of the first iteration. Therefore, every honest party broadcasts `(Terminate with` σ`)` in Step 2 of the first iteration. Therefore, every honest party will receive at least $n - t$ `(Terminate with` σ`)` broadcasts, and at most t `(Terminate with` $\bar{\sigma}$`)` broadcasts. Consequently, every honest party will output σ. □

Protocol BA$[\epsilon](x_i)$

Code for party P_i, on input x_i, and termination parameter ϵ:

1. Set $r := 0$. Set $v_1 := x_i$.

Repeat until terminating:

2. Set $r := r + 1$. Set (y_r, m_r) :=Vote(v_r).
3. Wait until Vote(v_r) is completed. Then, set c_r :=Common-Coin$[\frac{\epsilon}{4}]$.
4. (a) If $m_r = 2$, set $v_r := y_r$ and broadcast `(Terminate with` v_r`)` . Participate in only one more Vote protocol and only one more Common-Coin protocol. [a]
   (b) If $m_r = 1$, set $v_{r+1} := y_r$.
   (c) Otherwise, set $v_{r+1} := c_r$.

- Upon receiving $t + 1$ `(Terminate with` σ`)` broadcasts for some value σ, output σ and terminate.

Figure 5-11: The Byzantine Agreement protocol

[a] The purpose of this restriction is to prevent the parties from participating in an unbounded number of iterations before enough `(Terminate with` σ`)` broadcasts are completed.

Lemma 5.37 *If an honest party terminates with output σ, then all honest parties terminate with output σ.*

Proof: Let us first establish that if an honest party broadcasts `(Terminate with` σ`)` for some value σ, then all honest parties eventually broadcast `(Terminate with` σ`)` . Let k be the first iteration in which an honest party initiated a `(Terminate with` σ`)` broadcast for some value σ. By Lemma 5.34, every honest party P_i has $y_k = \sigma$ and either $m_k = 2$ or $m_k = 1$. Therefore, no honest party broadcasts `(Terminate with` $\bar{\sigma}$`)` at iteration k. Furthermore, all the honest parties execute the Vote protocol of iteration $k + 1$ with input σ. Lemma 5.33 now implies that by the end of Step 1 of iteration $k+1$, every honest party has $(y_{k+1}, m_{k+1}) = (\sigma, 2)$. Thus, all the honest parties broadcast `(Terminate with` σ`)` , either at iteration k or at iteration $k + 1$.

Now, assume an honest party terminates with output σ. Thus, at least one honest party broadcasted `(Terminate with` σ`)` . Consequently, all the honest parties broadcast `(Terminate with` σ`)` . Hence, every honest party will receive at least $n - t$ `(Terminate with` σ`)` broadcasts, and at most t `(Terminate with` $\bar{\sigma}$`)` broadcasts. Therefore, every honest party will output σ. □

Lemma 5.38 *Assume all honest parties have initiated and completed some round k. Then, with probability at least $\frac{1}{4}$ all honest parties have the same value for v_{k+1}.*

Proof: We distinguish two cases. If all the honest parties execute Step 4(c) in iteration k, then all the honest parties set their v_{k+1} value to their (local) output of protocol Common-Coin. In this case all the parties have the same v_{k+1} value with probability at least $\frac{1}{2}$.

Otherwise, some honest party has set $v_{k+1} = \sigma$ for some $\sigma \in \{0,1\}$, either in Step 4(a) or Step 4(b) of iteration k. By Lemma 5.35, no honest party will use either Step 4(a) or Step 4(b) to set his v_{k+1} variable to $\bar{\sigma}$. Furthermore, with probability at least $\frac{1}{4}$, all the honest parties have output σ of the Common-Coin protocol of Step 3. Therefore, with probability at least $\frac{1}{4}$, all the honest parties have $v_{k+1} = \sigma$ at the end of iteration k. (Note that the parties' outputs of protocol Vote are fixed *before* Common-Coin is invoked. Were it not the case, the bad parties could force the output of protocol Vote to prevent agreement.) □

Let C_k denote the event that each honest party completes all the iterations he initiated, up to (and including) the kth iteration (that is, for each iteration $1 \le l \le k$ and for each party P, if P initiated iteration l then he computes v_{l+1}). Let C denote the event that C_k occurs for all k.

Lemma 5.39 *Conditioned on event C, all the honest parties complete protocol BA in constant expected time.*

Proof: We first establish that all the honest parties terminate protocol BA within constant time after the first honest party initiates a `(Terminate with` σ`)` broadcast in Step 4(a) of the protocol. Assume the first honest party initiates a `(Terminate with` σ`)` broadcast in iteration k. Then, all the honest parties participate in the Vote and Common-Coin protocols of all the iterations up to iteration $k+1$. We have seen in the proof of Lemma 5.37 that in this case, all the honest parties initiate a `(Terminate with` σ`)` broadcast by the end of iteration $k+1$. All these broadcasts terminate in constant time. Each honest party terminates upon completing $t+1$ of these broadcasts.

Let the random variable τ count the number of iterations until the first honest party broadcasts `(Terminate with` σ`)` . (If no honest party broadcasts `(Terminate with` σ`)` then $\tau = \infty$). Conditioned on event C, all the honest parties terminate each iteration in constant time. It is left to show that $\mathrm{E}(\tau|C)$ is constant. We have

$$\mathrm{Prob}(\tau > k|C_k) \le \mathrm{Prob}(\tau \ne 1|C_k) \cdot \ldots \cdot \mathrm{Prob}(\tau \ne k|C_k \cap \tau \ne 1 \cap \ldots \cap \tau \ne k-1)$$

If follows from Lemma 5.38 that each one of the k multiplicands of the right hand side of the above equation is at most $\frac{3}{4}$. Thus, $\mathrm{Prob}(\tau > k|C_k) \le \left(\frac{3}{4}\right)^k$. It follows, via a simple calculation, that $\mathrm{E}(\tau|C) \le 16$. □

Lemma 5.40 $\mathrm{Prob}(C) \ge 1 - \epsilon$.

Proof: We have

$$\begin{aligned} \mathrm{Prob}(\overline{C}) &\le \sum_{k\ge 1} \mathrm{Prob}(\tau > k \cap \overline{C_{k+1}}|C_k) && (5.1)\\ &\le \sum_{k\ge 1} \mathrm{Prob}(\tau > k|C_k) \cdot \mathrm{Prob}(\overline{C_{k+1}}|C_k \cap \tau > k) && (5.2) \end{aligned}$$

We have seen in the proof of Lemma 5.39 that $\mathrm{Prob}(\tau > k|C_k) \le \left(\frac{3}{4}\right)^{k-1}$. We bound the term $\mathrm{Prob}(\overline{C_{k+1}}|C_k \cap \tau \ge k)$. If all the honest parties execute the kth iteration and complete the kth invocation of Common-Coin, then all the honest parties complete the kth iteration. Protocol Common-Coin is invoked with 'termination parameter' $\frac{\epsilon}{4}$. Thus, with probability

$1-\frac{\epsilon}{4}$, all the honest parties complete the kth invocation of Common-Coin. Therefore, for each k, $\text{Prob}(\overline{C_{k+1}}|C_k \cap \tau \geq k) \leq \frac{\epsilon}{4}$. Inequality 5.1 yields $\text{Prob}(\overline{C}) \leq \sum_{k\geq 1} \frac{\epsilon}{4}\left(\frac{3}{4}\right)^{k-1} = \epsilon$. □

We have thus shown:

Theorem 5.2 (Byzantine Agreement.) *Let $n \geq 3t+1$. Then, for every $0 < \epsilon \leq 0.2$, protocol* BA[ϵ] *is a $(1-\epsilon)$-terminating, t-resilient, asynchronous Byzantine Agreement protocol for n parties. Given that the parties terminate, they do so in constant expected time. Furthermore, the computational resources required of each party are polynomial in n and* $\log\frac{1}{\epsilon}$.

CHAPTER 6

Proactive security

We introduce a method for maintaining the security of systems in the presence of repeated, however transient break-ins to system components. We use secure multiparty computation as a formal setting, and use mobile adversaries as a vehicle for concentrating on the mechanism of recovery from break-ins. This construction has applications to key management schemes in actual security systems. See Section 1.6 for an introductory presentation.

In Section 6.1 we define PP protocols, and recall the definition of pseudorandom function families. In Sections 6.2 and 6.3 we describe our PP protocol and prove its correctness. In Section 6.4 we describe some modifications to our application to secure sign-on protocols. In Section 6.5 we offer an alternative definition of PP protocols, and show that it is implied by our first definition.

6.1 Definitions

The setting. We consider a synchronous network with secure channels, and computationally bounded mobile adversaries. (In Section 6.3.1 we describe a relaxation of this security requirement on the channels.) For simplicity, we assume that at the end of each round, each party can send a message to each other party; these messages are received at the beginning of the next round. It is simple to extend our results to more realistic communication and synchronization models. (We remark that here the rounds formalize the applications of the automatic recovery mechanism described in Section 1.6. These are different than communication rounds. Typically, a recovery round may take place every several days, where a communication round lasts only a fraction of a second.)

The Adversary. At the beginning of each round the adversary may corrupt parties. (The adversary adaptively decides which parties to corrupt at each round.) Upon corruption of a party, the entire contents of the party's memory becomes known to the adversary. Furthermore, the adversary can alter the party's memory and program. After some time the adversary leaves the party. Once the adversary has left, the party returns to execute its original program; however, its memory may have been altered. We call this adversary a mobile adversary. We say that a mobile adversary is t-limited if in each round at most

t parties are corrupted. (We stress that there may exist no party that has never been corrupted!)

A definition of proactive pseudorandomness. Consider a network of parties performing some computation in the presence of a mobile adversary. The parties have access to randomness only at the beginning of the computation. Once the interaction starts no additional randomness is available.

The parties will run a special deterministic protocol; this protocol will generate a new value within each party at each round. Given that the parties' initial inputs of this protocol are randomly chosen, the value generated within each party at each round will be indistinguishable from random from the point of view of a mobile adversary, *even if the values generated within all the other parties, at all rounds, are known.* We call such a protocol a proactive pseudorandomness (PP) protocol.

We stress that at each round the adversary may know, in addition to the data gathered by the adversary, the outputs of all the parties (including the uncorrupted ones) at all the previous rounds. Still, it cannot distinguish between the current output of an uncorrupted party and a random value.

More formally, consider the following attack, called an on-line attack, with respect to an n-party PP protocol. Let the input of each party be taken at random from $\{0,1\}^k$, where k is a security parameter (assume $n < k$). Furthermore, each party's output at each round is also a value in $\{0,1\}^k$.

On-line attack

The protocol is run in the presence of a mobile adversary for m rounds (m is polynomial in n and k), where in addition to the data gathered by the adversary, the adversary knows the outputs of all the parties at all the rounds. At a certain round, l (chosen "on-line" by the adversary), the adversary chooses a party, P, out of the uncorrupted parties at this round. The adversary is then given a test value, v, instead of P's output at this round. The execution of the protocol is then resumed for rounds $l+1,\ldots,m$. (Our definition will require that the adversary be unable to say whether v is P's output at round l, or a random value.)

F or an n-party protocol π, and a mobile adversary A, let $A(\pi,\mathrm{PR})$ (respectively, $(A(\pi,\mathrm{R}))$ denote the output of A after an on-line attack on π, and when the test value v given to A is indeed the output of the specified party (respectively, when v is a random value). Without loss of generality, we assume that $A(\pi,\mathrm{PR}) \in \{0,1\}$.

Definition 6.1 *Let π be a deterministic n-party protocol with security parameter k. We say that π is a t-resilient* proactive pseudorandomness protocol (PP) *if for every t-limited polynomial time mobile adversary A, for all $c > 1$ and all large enough k we have*

$$|\mathrm{Prob}(A(\pi,\mathrm{PR})=1) - \mathrm{Prob}(A(\pi,\mathrm{R})=1)| \le \frac{m}{k^c}$$

where m is the total number of rounds of protocol π, and the probability is taken over the parties' inputs of π and the choices of A. (We stress that m is polynomial in k.)

We say that π is efficient *if it uses resources polynomial in n and k.*

An alternative definition. Using Definition 6.1 above, it can be shown that the following property holds for any randomized application protocol α run by the parties. Consider a variant, α', of α that runs a PP protocol π along with α, and uses the output of π as random input for α at each round. Then, The parties' outputs of α and α' are indistinguishable; furthermore, running α' the adversary gains no knowledge it did not gain running α. In fact, this property may serve as an alternative definition for PP protocols. In Section 6.5 we present a more precise definition of this property.

Pseudorandom function families. Our constructions make use of pseudorandom functions families. We briefly sketch the standard definition.

Let $\mathcal{F}_k$ denote the set of functions from $\{0,1\}^k$ to $\{0,1\}^k$. Say that an algorithm D with oracle access distinguishes between two random variables f and g over $\mathcal{F}_k$ with gap $s(k)$, if the probability that D outputs 1 with oracle to f differs by $s(k)$ from the probability that D outputs 1 with oracle to g. Say that a random variable f over $\mathcal{F}_k$ is $s(k)$-pseudorandom if no polynomial time (in k) algorithm with oracle access distinguishes between f and $g \in_{\mathrm{R}} \mathcal{F}_k$ with gap $s(k)$. (Throughout the paper, we let $e \in_{\mathrm{R}} D$ denote the process of choosing an element e uniformly at random from domain D.)

We say that a function family $F_k = \{f_\kappa\}_{\kappa \in \{0,1\}^k}$ (where each $f_\kappa \in \mathcal{F}_k$) is $s(k)$-pseudorandom if the random variable f_κ where $\kappa \in_{\mathrm{R}} \{0,1\}^k$ is $s(k)$-pseudorandom. A collection $\{F_k\}_{k \in \mathbf{N}}$ is pseudorandom if for all $c > 0$ and for all large enough k, the family F_k is $\frac{1}{k^c}$-pseudorandom. We consider pseudorandom collections which are efficiently constructible. Namely, there exists a polytime algorithm that on input $\kappa, x \in \{0,1\}^k$ outputs $f_\kappa(x)$.

Pseudorandom function families and their cryptographic applications were introduced by Goldreich, Goldwasser and Micali [GGM2, GGM1]. Applications to practical key distribution and authentication protocols were shown by Bellare and Rogaway [BR1]. In [GGM2] it is shown how to construct pseudo-random functions from any pseudo-random generator, which in turn could be constructed from any one-way function [HILL]. However, practitioners often trust and use much simpler constructions based on DES or other widely available cryptographic functions.

6.2 The Protocol

In this section we describe the basic protocol. Several modifications useful for the application to secure sign-on are described in Section 6.4.

Consider a network of n parties, $P_1, \ldots, P_n$, having inputs $x_1, \ldots, x_n$ respectively. Each input value x_i is uniformly distributed in $\{0,1\}^k$, where k is a security parameter. We assume that parties have agreed on a predefined pseudorandom function family $F = \{f_\kappa\}_{\kappa \in \{0,1\}^k}$, where each $f_\kappa : \{0,1\}^k \to \{0,1\}^k$.

In each round l each party P_i computes an internal value (called a key), $\kappa_{i,l}$, in a way described below. P_i's output at round l, denoted $r_{i,l}$, is set to be $r_{i,l} = f_{\kappa_{i,l}}(0)$, where 0 is an arbitrary fixed value.

The key $\kappa_{i,l}$ is computed as follows. Initially, P_i sets its key to be its input value, namely $\kappa_{i,0} = x_i$. At the end of each round $l \geq 0$, party P_i sends $f_{\kappa_{i,l}}(j)$ to each party P_j. Next, P_i *erases* its key for round l and sets its key for round $l+1$ to the bitwise exclusive or of the values received from all the parties at this round:

$$\kappa_{i,l+1} = \oplus_{j=1}^{n} f_{\kappa_{j,l}}(i) \tag{6.1}$$

We stress that it is crucial that the parties *erase* the old keys. In fact, if parties *cannot erase* their memory, proactive pseudo-randomness is impossible. In particular, once each party has been corrupted in the past, the adversary has complete information on the system at, say, the first round. Now the adversary can predict all the subsequent outputs of this deterministic protocol.

6.3 Analysis

We first offer some intuition for the security of our protocol. This intuition is based on an inductive argument. Assume that, at round l, the key of an uncorrupted party is pseudorandom from the point of view of the adversary. Therefore, the value that this party sends to each other party is also pseudorandom. Furthermore, the values received by different parties seem unrelated to the adversary; thus, the value that each party receives from an uncorrupted party is pseudorandom from the point of view of the adversary, even if the values sent to other parties are known. Thus, the value computed by each uncorrupted party at round $l+1$ (being the bitwise exclusive or of the values received from all the parties) is also pseudorandom.

Naturally, this argument serves only as intuition. The main inaccuracy in it is in the implied assumption that we do not lose any pseudo-randomness in the repeated applications of pseudorandom functions. A more rigorous proof of correctness (using known techniques) is presented below.

Theorem 6.2 *Our protocol, given a pseudorandom function family, is an efficient,* $(n-1)$*-resilient PP protocol.*

Proof: Let π denote our protocol (run for m rounds). Assume there exists a polytime mobile adversary A such that

$$|\mathrm{Prob}(A(\pi, \mathrm{PR}) = 1) - \mathrm{Prob}(A(\pi, \mathrm{R}) = 1)| > \frac{m}{k^c}$$

for some constant $c > 0$ and some value of k. For simplicity we assume that A corrupts exactly $n-1$ parties at each round, and that A always runs the full m rounds before outputting its guess. The proof can be easily generalized to all A. We show that F_k is not pseudorandom. Specifically, we construct a distinguisher D that distinguishes with gap $\frac{1}{2k^c}$ between the case where its oracle is taken at random from F_k and the case where its oracle is a random function in $\mathcal{F}_k$.

In order to describe the operation of D, we define hybrid probabilities as follows. First, define $m+1$ hybrid protocols, $H_0, \ldots, H_m$, related to protocol π. Protocol H_i instructs each party P_s to proceed as follows.

- In rounds $l \leq i$ party P_s outputs a random value and sends a random value to each other party P_t (instead of $f_{\kappa_{s,l}}(0)$ and $f_{\kappa_{s,l}}(t)$, respectively). In other words, P_s uses a random function from $\mathcal{F}_k$ instead of $f_{\kappa_{s,l}}$ for his computations.

- In rounds $l > i$ party P_s executes the original protocol, π.

Ofcourse, whenever a party is corrupted it follows the instructions of the adversary.

Distinguisher D, given oracle access to function g, operates as follows. First, D chooses at random a round number $l_0 \in_{\rm R} [0, \ldots, m-1]$. Next, D runs adversary A on the following simulated on-line attack on a network of n parties. The corrupted parties follow the instructions of A. The (single) uncorrupted party at each round l, denoted $P_{*(l)}$, proceeds as follows.

1. In rounds $l < l_0$, party $P_{*(l)}$ outputs a random value and sends a random value to each other party (as in the first steps of the hybrid interactions).
2. In round l_0, party $P_{*(l_0)}$ uses the oracle function g to compute its output and messages. Namely, it outputs $g(0)$ and sends $g(j)$ to each other party P_j.
3. In rounds $l > l_0$, party $P_{*(l)}$ follows protocol π.

(Note that D knows which parties are corrupted by A at each round.)

Once a round is completed, D reveals all the parties' outputs of this round to A (as expected by A in an on-line attack). When A asks for a test value v, D proceeds as follows. First, D chooses a bit $b \in_{\rm R} \{0,1\}$. If $b = 0$, then D sets v to the actual corresponding output of the party chosen by A. Otherwise, D sets v to a random value. Finally, if $b = 0$ then at the end of the simulated interaction D outputs whatever A outputs. If $b = 1$ then D outputs the *opposite value* to whatever A outputs.

The operation of D can be intuitively explained as follows. It follows from a standard hybrids argument that there must exist an i such that either $|\mathrm{Prob}(A(H_i, \mathrm{PR}) = 1) - \mathrm{Prob}(A(H_{i+1}, \mathrm{PR}) = 1)|$ is large or $|\mathrm{Prob}(A(H_i, \mathrm{R}) = 1) - \mathrm{Prob}(A(H_{i+1}, \mathrm{R}) = 1)|$ is large. Thus, if D chooses the "correct" values for l_0 and b it can use the output of A to distinguish between the two possible distributions of its oracle. We show that a similar distinction can be achieved if l_0 and b are chosen at random.

We analyze the output of D as follows. Let $\mathrm{PR}_i \stackrel{\triangle}{=} \mathrm{Prob}(A(H_i, \mathrm{PR}) = 1)$. (Namely, PR_i is the probability that A outputs 1 after interacting with parties running protocol H_i and when the test value given to A is indeed the corresponding output value of the party that A chose.) Similarly, let $\mathrm{R}_i \stackrel{\triangle}{=} \mathrm{Prob}(A(H_i, \mathrm{R}) = 1)$. Let ρ (resp., ϕ) be a random variable distributed uniformly over $\mathcal{F}_k$ (resp., over F_k). Assume that D is given oracle access to ρ. Then, at round l_0 party $P_{*(l_0)}$ outputs a random value and sends random values to all the other parties. Thus, the simulated interaction of A is in fact an on-line attack of A on protocol H_{l_0}. Therefore, if $b = 0$ (resp., if $b = 1$), then D outputs 1 with probability PR_{l_0} (resp., $1 - \mathrm{R}_{l_0}$). Similarly, D is given oracle access to ϕ then the simulated interaction of A is an on-line attack of A on protocol H_{l_0+1}. In this case, if $b = 0$ (resp., if $b = 1$), then D outputs 1 with probability PR_{l_0+1} (resp., $1 - \mathrm{R}_{l_0+1}$). Thus,

$$\begin{aligned}
\mathrm{Prob}(D^{\rho} = 1) - \mathrm{Prob}(D^{\phi} = 1) &= \frac{1}{2m}\sum_{i=0}^{m-1}(\mathrm{PR}_i + 1 - \mathrm{R}_i) - \frac{1}{2m}\sum_{i=0}^{m-1}(\mathrm{PR}_{i+1} + 1 - \mathrm{R}_{i+1}) \\
&= \frac{1}{2m}\sum_{i=0}^{m-1}[(\mathrm{PR}_i - \mathrm{R}_i) - (\mathrm{PR}_{i+1} - \mathrm{R}_{i+1})] \\
&= \frac{1}{2m}[(\mathrm{PR}_0 - \mathrm{R}_0) - (\mathrm{PR}_m - \mathrm{R}_m)].
\end{aligned}$$

Clearly, H_0 is the original protocol π. Thus, by the contradiction hypothesis, $|\mathrm{PR}_0 - \mathrm{R}_0| > \frac{m}{k^c}$. On the other end, in protocol H_m the parties output random values in all the m rounds, thus $\mathrm{PR}_m - \mathrm{R}_m = 0$. We conclude that $|\mathrm{Prob}(D^\rho = 1) - \mathrm{Prob}(D^\phi = 1)| > \frac{1}{2m} \cdot \frac{m}{k^c} = \frac{1}{2k^c}$. □
□

6.3.1 Insecure Links

When describing the model, we assumed that all the communication channels are secure (i.e., private and authenticated). Here, we discuss the effect of insecure channels on our protocol. We note that the protocol remains a PP protocol even if in each round l only a single uncorrupted party P_i has a single channel which is secure in round l to a party P_j that wasn't corrupted in round $l-1$ (and P_j had in round $l-1$ a secure channel to an uncorrupted party, etc.).

This security requirement on the channels is minimal in the following sense. If no randomness is allowed after the interaction begins then a mobile adversary that sees the entire communication can continue simulating each party that has once been corrupted, even after the adversary has left this party. Thus, after few rounds, the adversary will be able to simulate all parties and predict the output of each party at each subsequent round.

6.4 On the Application to Secure Sign-On

In subsection 1.6.1 we discussed reconstructible protocols and described an application of our PP protocol to proactive secure sign-on, using its reconstructability. However, as mentioned there, the protocol described in Section 6.2 is reconstructible only if all the parties (servers) follow their protocols at all times (that is, the adversary is only eavesdropping).

In this section we describe modifications of our protocol, aimed at two goals: one goal is to make the protocol more efficient for the user; the other goal is to maintain the reconstructability property for the case where the servers don't follow their protocols.

We start by describing a variant of the protocol which is more efficient for the user. Using the protocol described in Section 6.2, the user had to simulate the computation performed by the servers step by step. In case that many rounds have passed since the last time the user updated its keys, this may pose a considerable overhead. Using this variant, denoted ρ, the user can compute its updated key simulating only *one* round of computation of the servers. On the other hand, this variant has a weaker resilience property: it assures that the servers' keys be unpredictable by the adversary only if there exists a server that has never been corrupted.

The variant is similar in structure to the original protocol with the following modification. Each P_i has a **master key** which is never erased. This master key is set to be the initial key, $\kappa_{i,0}$ (derived, say, from the password). The master key is used as the index for the function at all the rounds. Namely, the key $\kappa_{i,l}$ at round l is computed by $\kappa_{i,l} = \oplus_{j=1}^{n} f_{\kappa_{j,0}}(i)$.

It is also possible to combine the original protocol with the variant described above, in order to reach a compromise between efficiency and security. We define a special type of round: a **major round**. (For instance, let every 10th round be a major round.) The parties now update their master keys, using the original protocol, only at major rounds. In non-major rounds, the servers use their current master key as the index for the function.

This combined protocol has the following properties. On one hand, the user has one tenth of the rounds to simulate than in the original scheme. On the other hand, we only need that in any period of 10 rounds there exists a server that has not been corrupted. We believe that such versions may be a more reasonable design for actual implementations.

Next, we describe additions to the protocol, aimed at maintaining the reconstructability property for the case where the servers don't follow their protocols. We note that it is possible to withstand crash failures of servers, if the servers cooperatively keep track of which servers crashed at each round (this coordination can be done using standard consensus protocols).

The following addition to the protocol handles the case of Byzantine faults, if variant ρ described above is used. The only way an adversary controlling party P_j could interfere with the reconstructability of variant ρ is by sending a wrong value instead of $f_{\kappa_{j,0}}(i)$, to some server P_i. However, P_i can, when unable to authenticate a user, compare each of the $f_{\kappa_{j,0}}(i)$ values with the user, e.g. using the 2PP protocol [BGH+2], without exposing any of the values. If there are more than half of the values which match, the server and the user may use the exclusive or of these values only. This technique requires that at any round, the majority of the servers are non-faulty (otherwise the server may end up using values which are all known to the adversary). We note that this idea does not work if the basic protocol (that of Section 6.2) is used instead of variant ρ.

6.5 An alternative definition of PP

We offer an alternative definition of a PP. This definition follows from Definition 6.1. Borrowing from the theory of secure encryption functions, we call Definition 6.1 "PP in the sense of indistinguishability" (or, in short, PP), whereas Definition 6.4 below is called "semantic PP" (or, in short, SPP). We first recall the standard definition of polynomial indistinguishability of distributions.

Definition 6.3 *Let $\mathcal{A} = \{A_k\}_{k\in\mathbf{N}}$ and $\mathcal{B} = \{B_k\}_{k\in\mathbf{N}}$ be two ensembles of probability distributions. We say that $\mathcal{A}$ and $\mathcal{B}$ are* polynomially indistinguishable *if there exists a constant $c > 0$ such that for every polytime distinguisher D and for all large enough k,*

$$|\text{Prob}(D(A_k) = 1) - \text{Prob}(D(B_k) = 1)| < \frac{1}{k^c}.$$

We colloquially let $A_k \approx B_k$ denote "$\{A_k\}$ and $\{B_k\}$ are polynomially indistinguishable".

Let α be some distributed randomized protocol, which is resilient against t-limited mobile adversaries, for some value of t. We wish to adapt protocol α to a situation where no randomness is available once the interaction starts. Namely, we want to construct a protocol α' in which the parties use randomness only before the interaction starts, and the parties' outputs of protocol α' are "the same as" their outputs of protocol α.

A general framework for a solution to this problem proceeds as follows. The parties run protocol α along with another *deterministic* protocol, π. Each party's local input of protocol π is chosen at random at the beginning of the interaction. In each round, each party sets the random input of protocol α for this round to be the current output of protocol π. We call π a semantically secure proactive pseudorandomness(SPP) protocol. We refer to protocol α as the application protocol.

We state the requirements from a SPP protocol π. Informally, we want the following requirement to be satisfied for every application α. Whatever an adversary can achieve by interacting with α combined with protocol π as described above, could also be achieved by interacting with the original protocol α when combined with a truly random oracle. More formally,

- for an n-party randomized application protocol α, a mobile adversary A, an input vector $\vec{x} = x_1, \ldots, x_n$, and $1 \leq i \leq n$, let $\alpha(\vec{x}, A)_i$ denote party P_i's output of protocol α with a random oracle when party P_j has input x_j and in the presence of adversary A. Let $\alpha(\vec{x}, A)_0$ denote A's output of this execution. Let $\vec{\alpha}(\vec{x}, A) \stackrel{\triangle}{=} \alpha(\vec{x}, A)_0, \ldots, \alpha(\vec{x}, A)_n$.

- For a randomized protocol α and a deterministic protocol π for which each party has an output at each round, let $\alpha\pi$ denote the protocol in which α and π are run simultaneously and each party at each round sets the random input of α to be the current output of π.

Definition 6.4 *We say that an n party deterministic protocol π is a t-resilient* SPP *protocol if for every (randomized) application protocol α and every t-limited mobile adversary A there exists a t-limited mobile adversary A' such that for every input vector $\vec{x}$ (for protocol α) we have*

$$\vec{\alpha\pi}(\vec{x}, A') \approx \vec{\alpha}(\vec{x}, A).$$

where the probabilities are taken over the inputs of π and the random choices of A and of the oracle of α.

Theorem 6.5 *If a protocol is a t-resilient PP protocol then it is a t-resilient SPP protocol.*

CHAPTER 7

Conclusion

In this brief chapter we present a personal view of our work and try to point out its merits. We also outline some directions for further research.

Merit of this work. An important contribution of this work is, in our opinion, the precise and workable definitions of secure multiparty computation (presented in Chapter 2 and Section 4.1). These definitions allow us, for the first time, to have a clear and precise notion of a secure multiparty protocol, and to prove security of our protocols.

We believe that without these definitions we would not have been able to come up with protocols that solve the adaptive security problem and the problem of secure computation in asynchronous networks. Hopefully, the framework and tools developed for these definitions will prove helpful in formulating precise definitions of primitives encountered in the future.

The long-sought solution for the adaptive security problem (Chapter 3) allows us to present secure solutions for an abundance of known protocol problems in the presence of adaptive adversaries (which seem to be a better model of reality than non-adaptive ones). Our solution stems from a better understanding of the nature of secure multiparty computation. This understanding resulted in a better modeling of the degree of trust the party have in each other, namely in the notion of a semi-honest party.

Our solution works only for non-erasing parties. Investigating this notion of semi-honesty only slightly further, one finds out that in a very natural (arguably, even the most natural) trust-model, namely that of honest-looking parties, we are unable to prove adaptive security of practically *any* non-trivial protocol (unless a trusted dealer is available in an initial preprocessing phase). This holds even in the presence of absolutely secure channels. This surprising phenomenon should be taken into account whenever adaptive adversaries are considered.

The main tool used in solving the adaptive security problem is non-committing encryption. The essence of these encryptions is separating *encryption* from *commitment.* Until this work, encryption was thought of as an inevitably committing primitive (in the sense that the ciphertext could serve as a *commitment* of the sender - and of the receiver - to the plaintext). We showed that encryption can be done without commitment, and demonstrated the benefits of this separation. We believe that non-committing encryption is of independent interest. In particular, non-committing encryption may prove to be a 'more secure' way to encrypt data than standard encryption in many other scenarios.

In the chapter on asynchronous secure computation (Chapter 4) we define what it means to securely compute (or, rather, approximate) a function in an asynchronous setting with faults. We also show, in a detailed way, how known techniques for constructing protocols for securely computing any function can be adapted to the asynchronous setting. Finally, to the best of our knowledge this is the first (and so far only) place where a *full proof of security* of a construction for securely computing any function appears, in *any* model of computation. In particular, a security proof of the [BGW] construction can be extracted from the proof presented here.

The chapter on asynchronous Byzantine agreement (Chapter 5) applies ideas and techniques from secure multiparty computation to constructing the first asynchronous Byzantine agreement (BA) protocol with optimal resilience and polynomial complexity. The construction is quite involved, using many layers and techniques. The main technical contributions are, in our opinion, two: First, we adapt techniques from [RB, TRa] to construct the first Asynchronous Verifiable Secret Sharing (AVSS) scheme with optimal resilience. Next, we slightly modify the [F] scheme (which was never published and unaccessible) for reaching BA given an AVSS scheme, and present it in a (hopefully) readable way. Indeed, this work is the first accessible source to any asynchronous BA protocol with linear resilience and polynomial complexity.

The chapter on proactive security (Chapter 6) introduces a new approach to maintaining the security of computer systems in the presence of transient and repeated break-ins (or failures). We hope and believe that this approach will become a standard in the effort to protect computer systems. In fact, a number of works have already followed this approach (e.g., [ChH, HJKY, CHH, HJJKY]).

Subsequent and future work. We mention two directions for subsequent research. The first deals with an additional security requirements from multiparty protocols. Namely, we require that the computation will not leave a 'trace' that can be later used against the parties. An example is a 'mafia' that records the transcript and later uses it to 'coerce' parties to reveal their inputs. Some research in this direction has been done ([BT, SK, CDNO, CG]); however many questions remain open.

Another issue, not addressed in this work at all, is how to deal with *unauthenticated* communication channels, that can be actively controlled by the adversary. In fact how, again, to precisely define the problem? (Here additional definitional problems are encountered. For instance, the adversary may always prevent the parties from completing the computation, by simply not delivering messages.) A somewhat restricted definition, as well as a solution for a certain (quite powerful) adversary model is presented in [CHH]. Still, many questions remain open.

Bibliography

[AFL] E. Arjomandi, M. Fischer and N. Lynch, “Efficiency of Synchronous Versus Asynchronous Distributed Systems”, *Journal of the ACM*, No. 3, Vol. 30, 1983, pp. 449-456.

[Aw] B. Awerbuch, “The complexity of Network Synchronization”, *JACM, Vol. 32, no. 4,* pp. 804-823, 1985.

[Ba] E. Bach, “How to generate factored random numbers”, *SIAM J. on Comput.,* Vol. 17, No. 2, 1988, pp. 179-193.

[Be] D. Beaver, “Foundations of Secure Interactive Computing”, *CRYPTO,* 1991.

[BH] D. Beaver and S. Haber, “Cryptographic Protocols Provably secure Against Dynamic Adversaries”, *Eurocrypt,* 1992.

[BR1] M. Bellare and P. Rogaway, “Entity authentication and key distribution”, *Advances in Cryptology: Proc. of Crypto 93*, August 1993, pp. 232-249.

[BR2] M. Bellare and P. Rogaway, “Random oracles are practical: A paradigm for designing efficient protocols”, *First ACM conf. on Computer and Comm. Security,* November 1993, pp. 62-73.

[BT] Josh Benaloh and Dwight Tunistra, “Receipt-Free Secret-Ballot Elections”, *26th STOC,* 1994, pp. 544-552.

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

[BE] M. Ben-Or and R. El-Yaniv, “Interactive Consistency in Constant Time”, submitted for publication, 1991.

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

[BKR] M. Ben-Or, B. Kelmer and T. Rabin, "Asynchronous Secure Computation with Optimal Resilience", *13th PODC,* 1994 pp. 183-192.

[BGH+1] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "A family of light-weight protocols for authentication and key distribution", Submitted to IEEE T. Networking, 1993.

[BGH+2] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "Systematic design of a family of attack-resistant authentication protocols", *IEEE Journal on Selected Areas in Communications*, 11(5) (Special issue on Secure Communications), June 1993, pp.679–693. See also a different version in *Crypto 91.*

[BM] M. Blum, and S. Micali, "How to generate Cryptographically strong sequences of pseudo-random bits", em SIAM J. on Computing, Vol. 13, 1984, pp. 850-864.

[Br] G. Bracha, "An Asynchronous $\lfloor (n-1)/3 \rfloor$-resilient Consensus Protocol", *3rd PODC,* 1984, pp. 154-162.

[BCC] G. Brassard, D. Chaum and C. Crepeau, "Minimum Disclosure Proofs of Knowledge", *Journal of Computing and System Sciences,* Vol. 37, No. 2, 1988, pp. 156-189.

[C] R. Canetti, "Asynchronous Secure Computation", *Technical Report no. 755,* CS department, Technion, 1992.

[CDNO] R. Canetti, C. Dwork, M. Naor and R. Ostrovsky, "Deniable Encryptions", manuscript.

[CFGN] R. Canetti, U. Feige, O. Goldreich and M. Naor, "Adaptively Secure Computation", *28th STOC,* 1996.

[CG] R. Canetti and R. Genaro, "Deniable Multiparty Computation", manuscript.

[CHH] R. Canetti, S. Halevi and A. Herzberg, "How to Maintain Authenticated Communication in the Presence of Break-ins", manuscript.

[CaH] R. Canetti and A. Herzberg, "Maintaining security in the presence of transient faults", *Crypto' 94*, 1994, pp. 425-439.

[CR] R. Canetti and T. Rabin, "Optimal Asynchronous Byzantine Agreement", *25th STOC,* 1993, pp. 42-51.

[CCD] D. Chaum, C. Crepeau and I Damgard, "Multiparty unconditionally secure protocols", *20th STOC,* 1988, pp. 11-19.

[CD] B. Chor and C. Dwork, "Randomization in Byzantine Agreement", *Advances in Computing Research, Vol. 5*, 1989, pp. 443-497.

[CGMA] B. Chor, S. Goldwasser, S. Micali and B. Awerbuch, "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults", *26th FOCS,* 1985, pp. 383-395.

[CK] B. Chor and E. Kushilevitz, "A Zero-One Law for Boolean Privacy", *SIAM J. on Disc. Math., Vol. 4, no. 1,* 1991, pp.36-47.

[CM] B. Chor and L. Moscovici, "Solvability in Asynchronous Environments", *30th FOCS,* 1989.

[ChH] C. Chow and A. Herzberg, "A reconstructible proactive pseudo-randomness protocol", Work in progress, June 1994.

[DP] A. De-Santis and G. Persiano, "Zero-Knowledge proofs of knowledge without interaction", *33rd FOCS,* pp. 427-436, 1992.

[DH] W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Trans. on Info. Theory*, IT-22(6), 1976, pp. 644-654.

[Ed] J. Edmonds, 'Paths, Trees, and Flowers", *Canadian J. of Math., Vol.17,* 1965, pp. 449-467.

[DDN] D. Dolev, C. Dwork and M. Naor, "Non-malleable Cryptography", *23rd STOC,* 1991.

[ER] M. Elchin and J. Rochlis, "With microscope and tweezers: An analysis of the internet virus of november 1988", *IEEE Symp. on Security and Privacy*, 1989, pp. 326–343.

[EGL] S. Even, O. Goldreich and A. Lempel, "A randomized protocol for signing contracts", *CACM,* vol. 28, No. 6,1985, pp. 637-647.

[Fe] P. Feldman, "Asynchronous Byzantine Agreement in Constant Expected Time", unpublished manuscript, 1989.

[FM] P. Feldman and S. Micali, "An Optimal Algorithm For Synchronous Byzantine Agreement", *20th STOC,* 1988, pp. 148-161.

[F] M. Fischer, "The Consensus Problem in Unreliable Distributed System", *Technical Report, Yale University*, 1983.

[FLP] M. Fischer, N. Lynch and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process", *JACM, Vol. 32, no. 2,* 1985, pp. 374-382.

[GJ] M. R. Garey and D. S. Johnson, *"Computers and Intractability : a guide to NP-Completeness",* W.H. Freeman ed., N.Y., 1979.

[G] O. Goldreich, "Foundations of Cryptography (Fragments of a Book)", ed. Dept. of Computer Science and Applied Mathemetics, Weizmann Institute, 1995.

[GGL] O. Goldreich, S. Goldwasser, and N. Linial, "Fault-Tolerant Computation in the Full Information Model", *32nd FOCS,* 1991, pp. 447-457.

[GGM1] O. Goldreich, S. Goldwasser, and S. Micali, "On the cryptographic applications of random functions", *Advances in Cryptology: Proc. of Crypto 84*, 1984, pp. 276-288.

[GGM2] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions" *J. ACM*, 33(4), 1986, pp. 792-807. Extended abstract in FOCS84.

[GILVZ] O. Goldreich, R. Impagliazzo, L. Levin, R. Venkatesan and D. Zuckerman, "Security Preserving Amplification of Hardness", FOCS 1990, pp. 318–326.

[GrL] O. Goldreich and L. Levin, "A Hard-Core Predicate to any One-Way Function", *21st STOC,* 1989, pp. 25-32.

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

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

[HILL] J. Hästad, R. Impagliazzo, L. Levin, and M. Luby, "Construction of pseudo-random generator from any one-way functions", Manuscript, see preliminary versions by Impagliazzo et al. in *21st STOC* and Hästad in *22nd STOC*, 1993.

[HJJKY] A. Herzberg, M. Jakonsson, S. Jarecki, H. Krawczyk and M. Yung, "Proactive Public-Key and Signature Systems", manusript.

[HJKY] A. Herzberg, S. Jarecki, H. Krawczyk and M. Yung, "Proactive Secret Sharing or: How to Cope with Perpetual Leakage", *CRYPTO* 1995.

[IR] R. Impagliiazzo and S. Rudich, "Limits on the provable consequences of one-way permutations", *21th STOC,* 1989, pp. 44-58.

[KY] A. Karlin and A. Yao, "Probabilistic Lower Bounds for Byzantine Agreement", unpublished manuscript, 1986.

[LE] T. A. Longstaff and S. E. Eugene, "Beyond preliminary analysis of the wank and oilz worms: A case of study of malicious code", *Computers and Security*, 12(1), 1993, pp. 61-77.

[MS] F. J. Macwiliams and N. J. A. Sloane, *"The Theory of Error Correcting Codes",* North-Holland, 1977.

[MR] S. Micali and P. Rogaway, "Secure Computation", in preparation. Preliminary version in *CRYPTO 91.*

[MV] S. Micali and V. Vazirani, "An $O(\sqrt{|V|} \cdot |E|)$ Algorithm for Finding Maximum Matching in General Graphs", *21st FOCS,* 1980.

[MNSS] S. P. Miller, C. Neuman, J. I. Schiller, and J. H. Saltzer, "Kerberos authentication and authorization system", *Project Athena Technical Plan.* Massachusetts Institute of Technology, July 1987.

[MT] R. H. Morris and K. Thompson, "Unix password security", *Comm. ACM*, 22(11), November 1979, pp. 594-597.

[OY] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks", *Proceedings of the* 10^{th} *Annual ACM Symposium on Principles of Distributed Computing,* 1991, pp 51–59.

[PSL] R. Pease, Shostak and L. Lamport, "Reaching Agreement in the Presence of Faults", *JACM, Vol. 27 No. 2,* 1980, pp. 228-234.

[MRa1] M. Rabin, "How to exchange secrets by oblivious transfer", Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[MRa2] M. Rabin, "Randomized Byzantine Generals", *24th FOCS,* 1983, pp. 403-409.

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

[TRa] T. Rabin , "Robust Sharing of Secrets When The Dealer is Honest or Faulty", *Journal of the ACM*, No. 6, Vol. 41, 1994, pp. 1089-1109.

[Re] R. Reischuk, "A new solution to the byzantine generals problem", *Information and Control*, 1985, pp. 23-42.

[SK] K. Sako and J. Kilian, "Receipt-Free Mix-Type Voting Scheme", *Eurocrypt* 1995, pp. 393-403.

[Sh] A. Shamir, "How to share a secret", *CACM, Vol. 22, No. 11,* 1979, pp. 612-613.

[St] C. Stoll, "How secure are computers in the u.s.a.?", *Computers and Security*, 7(6), 1988, pp. 543-547.

[Y1] A. Yao, "Protocols for Secure Computation", *23th FOCS,* 1982, pp. 160-164.

[Y2] A. Yao, "Theory and applications of trapdoor functions", *23rd FOCS,* 1982, pp. 80-91.