Secure Multiparty Computation and Secret Sharing
An Information Theoretic Approach

Ronald Cramer
Ivan Damgård
Jesper Buus Nielsen

Book Draft

April 30, 2014

Contents

Preface

This is a book on information theoretically secure Multiparty Computation (MPC) and Secret-Sharing, and about the intimate and fascinating relationship between the two notions.

We decided to write the book because we felt that a comprehensive treatment of unconditionally secure techniques for MPC was missing in the literature. In particular, because some of the first general protocols were found before appropriate definitions of security had crystallized, proofs of those basic solutions have been missing so far.

We present the basic feasibility results for unconditionally secure MPC from the late 80ties, generalizations to arbitrary access structures using linear secret sharing, and a selection of more recent techniques for efficiency improvements. We also present our own variant of the UC framework in order to be able to give complete and modular proofs for the protocols we present.

We also present a general treatment of the theory of secret-sharing and in particular secret-sharing schemes with additional algebraic properties, also a subject where a treatment on textbook level seems to be missing the literature. One of the things we focus on is asymptotic results for multiplicative secret sharing which has various interesting applications that we present in the MPC part.

Our ambition has been to create a book that will be of interest to both computer scientists and mathematicians, and can be used for teaching at several different levels where different parts of the book will be used. We have therefore tried to make both main parts be self-contained, even if this implied some overlap between the parts. This means that there are several different ways to read the book, and we give a few suggestions for this below. In particular, the concept of secret sharing of course appears prominently in both parts. In the first MPC part, however, it is introduced only as a tool, on a “need-to-know” basis. In the second part we reintroduce the notion, but now as a general concept that is interesting in its own right, and with a comprehensive treatment of the mathematical background.

The book is intended to be self-contained enough to be read by advanced undergraduate students, and the authors have used large parts of the material in the book for teaching courses at this level. By covering a selection of more advanced material, the book can also be used for a graduate course.

How to Use this Book

For a course on advanced undergrad level for computer science students, we recommend to cover Chapter 1-Chapter 5. This will include the basic feasibility results for unconditionally secure MPC and the UC model. For some extra perspective it may also be a good idea to cover chapter Chapter 7 which is basically a survey of cryptographically secure solutions.

For a graduate level computer science course, we recommend to include also Chapter 8 and Chapter 9 as they contains several recent techniques and survey some open problems.

For a course in mathematics on secret-sharing and applications, we recommend to cover first Chapter 1, Chapter 3 and Chapter 6 . This will give an intuition for what secret-sharing is and how it is used in MPC. Then Part 2 should be covered to present the general theory of algebraic secret sharing. Finally, the last part of Chapter 9 can be used to present some of the more advanced applications.

Acknowledgements

We thank Irene Giacomelli, Morten Dahl Jørgensen, Antonio Macedone and Sarah Zakarias for doing an enormous amount of work in proof reading and commenting. We also thank Sarah Zakarias for writing some of the simulation proofs and helping us avoid many more blunders than are present now.

Part I

Secure Multiparty Computation

1

Introduction

Contents

1.1 Private Information, Uses and Misuses

In a modern information-driven society, the everyday life of individuals and companies is full of cases where various kinds of private information is an important resource. While a cryptographer might think of PIN-codes and keys in this context, this type of secrets is not our main concern here. Rather, we will talk about information that is closer to the "primary business" of an individual or a company. For a private person, this may be data concerning his economic situation, such as income, loans, tax data, or information about health, such as deceases, medicine usage etc. For a company it might be the customer database, or information on how the business is running, such as turnover, profit, salaries, etc.

What is a viable strategy for handling private information? Finding a good answer to this question has become more complicated in recent years. When computers were in their infancy, in the 1950's and 1960's, electronic information security was to a large extent a military business. A military organization is quite special in that confidential information needs to be communicated almost exclusively between its own members, and the primary security goal is to protect this communication from being leaked to external enemies. While it may not be trivial to reach this goal, at least the overall purpose is quite simple to phrase and understand.

In modern society, things get much more complicated: using electronic media, we need to interact and do business with a large number of parties, some of whom we never met, and where many of them may have interests that conflict with ours. So how do you handle your confidential data if you cannot be sure that the parties you interact with are trustworthy?

Of course, one could save the sensitive data in a very secure location and never access it, but this is of course unreasonable. Our private data usually has value only because we want to use them for something. In other words, we have to have ways of controlling leakage of confidential data while this data is being stored, communicated or computed on, *even in cases where the owner of the data does not trust the parties (s)he communicates with.*

A very interesting aspect that makes this problem even more important is that there are many scenarios where a large amount of added value can be obtained by *combining confidential information from several sources*, and from this compute some result that all parties are interested in. To illustrate what is meant by this, we look at a number of different example scenarios in the following subsections.

1.1.1 Auctions

Auctions exist in many variants and are used for all kinds of purposes, but we concentrate here on the simple variant where some item is for sale, and where the highest bid wins. We assume the auction is conducted in the usual way, where the price starts at some preset amount and people place increasing bids until no one wants to bid more than the holder of the currently highest bid. When you enter such an auction, you usually have some (more or less precisely defined) idea of the maximal amount you are willing to pay, and therefore when you will stop bidding. On the other hand, every bidder of course wants to pay as small a price as possible for the item.

Indeed, the winner of the auction may hope to pay less than his maximal amount. This will happen if all other bidders stop participating long before the current bid reaches this maximum.

For such an auction to work in a fair way, it is obvious that the maximum amount you are willing to pay should be kept private. For instance, if the auctioneer knows your maximum and is working with another bidder, they can force the price to be always just below your maximum and so force you to pay more than if the auction had been honest. Note that the auctioneer has an incentive to do this to increase his own income, which is often a percentage of the price the item is sold for.

On the other hand, the result of the auction could in principle be computed, if one was given as input the true maximum value each bidder assigns to the item on sale.

1.1.2 Procurement

A procurement system is a sort of inverted auction, where some party (typically a public institution) asks companies to bid for a contract, i.e., to make an offer on the price for doing a certain job. In such a case, the lowest bid usually wins. But on the other hand, bidders are typically interested in getting as large a price as possible.

It is obvious that bids are private information: a participating company is clearly not interested in the competitors learning its bid before they have to make theirs. This would allow them to beat your bid by always offering a price that is slightly lower that yours. This is also against the interests of the institution offering the contract because it will tend to make the winning bid larger.

On the other hand, the result of the process, namely who wins the contract, can in principle be computed given all the true values of the bids.

1.1.3 Benchmarking

Assume you run a company. You will naturally be interested in how well you are doing compared to other companies in the same line of business as yours. The comparison may be concerned with a number of different parameters, such as profit relative to size, average salaries, productivity, etc. Other companies will most likely have similar interests in such a comparison, which is known as a benchmark analysis. Such an analysis takes input from all participating companies. Based on this it tries to compute information on how well a company in the given line of business should be able to perform, and finally each company is told how its performance compares to this "ideal".

It is clear that each company will insist that its own data is private and must not be leaked to the competitors. On the other hand the desired results can be computed from the private data: there are several known methods from information economics for doing such an analysis efficiently.

1.1.4 Data Mining

In most countries, public institutions such as the tax authorities or the healthcare system keep databases containing information on citizens. In many cases there are advantages one can get from coordinated access to several such databases. Researchers may be able to get statistics they could not get otherwise, or institutions might get an administrative advantage from being able to quickly gather the information they need on a certain individual.

On the other hand, there is clearly a privacy concern here: access to many different databases by a single party opens the possibility to compile complete dossiers on particular citizens, which would be a violation of privacy. In fact, accessing data on the same person in several distinct databases is forbidden by law in several countries precisely because of this concern.

1.2 Do We Have to Trust Someone?

We are now in a position to extract some common features of all the scenarios we looked at. One way to describe them all is as follows: we have a number of parties that each possess some private data. We want to do some computation that needs all the private data as input. The parties are interested in learning the result, or at least some part of it, but still want to keep their private data as confidential as possible.

Hopefully, it is clear from the previous section that if we can find a satisfactory solution to this problem, there are a very large number of applications that would benefit. Moreover, this leads to an extremely intriguing theoretical question, as we now explain:

One possible – and trivial – solution would be to find some party T that everyone is willing to trust. Now all parties privately give their input to T, he does the required computation, announces the result to the parties, and forgets about the private data he has seen. A moment's thought will show that this is hardly a satisfactory solution: first, we have created a single point of attack from where all the private data can potentially be stolen. Second, the parties must all completely trust T, both with respect to privacy and correctness of the results. Now, the reason why there are privacy concerns is that the parties do not trust each other in the first place, so why should we believe that they can find a new party they all trust?

In some applications one may pay a party T for doing the computation; if the amount paid is thought to be larger than what T could gain from cheating, parties may be satisfied with the solution. This seems to work in some cases, for instance when a consultancy house is paid a large fee for doing a benchmark analysis – but this is of course a very expensive solution.

Thus, we are left with a fundamental question: *can the problem be solved without relying on a trusted party?*

At first sight, it may seem that this cannot be possible. We want to compute a result that depends on private data from *all* involved parties. How could one possibly do this unless data from several parties become known to someone, and hence we have to trust that party?

Nevertheless, as we shall see, the problem is by no means impossible to solve, and solutions do exist that are satisfactory, both from a theoretical and a practical point of view.

1.3 Multiparty Computation

Let us be slightly more precise about the problem we have to solve: the parties, or *players* that participate are called $\mathsf{P}_1, \dots, \mathsf{P}_n$. Each player P_i holds a secret input x_i, and the players agree on some function f that takes n inputs. Their goal is to compute $y = f(x_1, \dots, x_n)$ while making sure that the following two conditions are satisfied:

- Corretness: the correct value of y is computed; and
- Privacy: y is the *only* new information that is released.

Regarding the latter property, note that since the purpose of the whole exercise is that we want to learn y, the best we can hope for in terms of privacy is that nothing but y is leaked. Computing f such that privacy and correctness are achieved is referred to as computing f *securely.* Later in the book, we will be precise about what secure computing is; for now, we will be content with the above intuitive idea. Note also that one may consider a more general case where each player gets his own private output. We will do so later, for now we focus on a single, public output for simplicity.

As one example of how this connects to the scenarios from the previous section, one may think of x_i as a number, namely P_i's bid in a auction, and $f(x_1, ..., x_n) = (z, j)$ where $x_j = z$ and $z \geq x_i, i = 1, \ldots, n$, i.e., f outputs the highest bid, and the identity of the corresponding bidder. If we do not want the winner to pay his own bid, but the bid of the second highest bidder, we simply change z to be this value, which is again a well-defined function of the inputs. This would give us a function implementing a so-called second price auction.

In this section, we give a first intuition on how one might compute a function securely without relying on trusted parties. This requires that we specify a *protocol*, i.e., a set of instructions that players are supposed to follow to obtain the desired result. For simplicity, we will assume for now that players always follow the protocol. We will later address the case where some parties may deviate from the protocol, in order to get more information than they are supposed to or cause the result to be incorrect. We will also assume that any pair of players can communicate securely, i.e., it is possible for P_i to send a message m to P_j, such that no third party sees m and P_j knows that m came from P_i. We discuss later how this can be realized in practice.

1.3.1 Secure Addition and Voting

Let us first look at a simple special case, namely where each x_i is a natural number, and $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$. Secure computation of even such a simple function can have very meaningful applications. Consider the case where $\mathsf{P}_1, ..., \mathsf{P}_n$ want to vote on some yes/no decision. Then we can let x_i represent the vote of P_i where $x_i = 0$ means "no" and $x_i = 1$ means "yes". If we can compute the sum of the x_i securely, this exactly means we get a way to vote with the properties we usually expect: the result $\sum_{i=1}^{n} x_i$ is indeed the result of the vote, namely the number of yes-votes. Moreover, if the computation is secure, no information is leaked other than $\sum_{i=1}^{n} x_i$, in particular, no information is revealed on how a particular player voted.

We will now design a protocol for the voting application. To be consistent with the next example we give, we set $n = 3$. An exercise below shows how to construct a voting solution for any n.

Secret Sharing

Before we can solve the problem, we need to look at an important tool known as *secret-sharing*. The term may seem self-contradictory at first sight: how can anything be secret if you share it with others? Nevertheless, the name makes good sense: the point is that secret sharing provides a way for a party, say P_1, to spread information on a secret number s across all the players, such that they together hold full information on s, yet no player (except of course P_1) has any information on s. First, we choose a prime p, and we define $\mathbb{Z}_p$ as $\mathbb{Z}_p = \{0, 1, ..., p-1\}$[1]. In the following, we will think of the secret s as a number in $\mathbb{Z}_p$.

In order to *share the secret* s, P_1 chooses numbers r_1, r_2 uniformly at random in $\mathbb{Z}_p$, and sets

$$r_3 = s - r_1 - r_2 \bmod p \ .$$

Put another way: he chooses r_1, r_2, r_3 randomly from $\mathbb{Z}_p$, subject to the constraint that $s = r_1 + r_2 + r_3 \bmod p$. Note that this way of choosing r_1, r_2, r_3 means that each of the three numbers is uniformly chosen in $\mathbb{Z}_p$: for each of them, all values in $\mathbb{Z}_p$ are possible and equally likely. Now P_1 sends privately r_1, r_3 to P_2, r_1, r_2 to P_3, and keeps r_2, r_3 himself.

The r_j's are called the *shares* of the *secret* s.

The process we have described satisfies two basic properties: First, the secret s is kept private in the sense that neither P_2 nor P_3 knows anything about that secret. As a result if some hacker breaks into the machine of P_2 or P_3 (but not both) he will learn nothing about s. Second, the secret s can be reconstructed if shares from at least two players are available. Let argue that this is true in a more precise way:

Privacy. Even though P_1 has distributed shares of the secret s to the other players, neither P_2 nor P_3 has any idea what s is. For P_2 we can argue as follows: he knows r_1, r_3 (but not r_2) and that $s = r_1 + r_2 + r_3 \bmod p$. Take any $s_0 \in \mathbb{Z}_p$. From P_2's point of view, could it be that $s = s_0$? The answer is yes, for if $s = s_0$ it would have to be the case that $r_2 = s_0 - r_1 - r_3 \bmod p$. This is certainly a possibility, recall that r_2 is uniformly chosen in $\mathbb{Z}_p$, so all values are possible. However, any other choice, say $s = s_0' \neq s_0$ is also a possibility. If this was the answer we would have $r_2 = s_0' - r_1 - r_3 \bmod p$, which is a value that is different from $s_0 - r_1 - r_3 \bmod p$, but just as likely. We conclude that, from P_2's point of view, all values of s in $\mathbb{Z}_p$ remain possible and are equally likely. A similar argument shows that P_3 has no idea what s is.

Correctness. If two of the three parties pool their information, the secret can be reconstructed, since then all three shares will be known and one can simply add them modulo p.

Note that the privacy property is *information theoretic*: as long a party does not know all three summands, no amount of computing power can give him any information on the corresponding secret. In this book, we focus primarily on protocols with this type of security. The secret sharing technique shown above is a special case of so-called replicated secret sharing. There are many ways to realize secret sharing with other desirable properties than the method we show

[1] To be more precise, $\mathbb{Z}_p$ is another name for $\mathbb{Z}/p\mathbb{Z}$ where we identify $i \in \mathbb{Z}_p$ with the residue class of numbers that are congruent to i modulo p. See more details in the preliminaries section.

here, and we look at several such techniques later, as well as a more general definition of what secret sharing is.

A Protocol for Secure Addition

Of course players P_2 and P_3 can distribute shares of their private values x_2, x_3 in exactly the same way as P_1. It turns out that one can now compute the sum securely by locally adding shares and announcing the result. The complete protocol is shown below.

Protocol Secure Addition

Participants are $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$, input for P_i is $x_i \in Z_p$, where p is a fixed prime agreed upon in advance.

1. Each P_i computes and distributes shares of his secret x_i as described in the text: he chooses $r_{i,1}, r_{i,2}$ uniformly at random in $\mathbb{Z}_p$, and sets $r_{i,3} = x_i - r_{i,1} - r_{i,2} \bmod p$.
2. Each P_i sends privately $r_{i,2}, r_{i,3}$ to P_1, $r_{i,1}, r_{i,3}$ to P_2, and $r_{i,1}, r_{i,2}$ to P_3 (note that this involves P_i sending "to himself"). So P_1, for instance, now holds $r_{1,2}, r_{1,3}$, $r_{2,2}, r_{2,3}$ and $r_{3,2}, r_{3,3}$.
3. Each P_j adds corresponding shares of the three secrets – more precisely, he computes, for $\ell \neq j$, $s_\ell = r_{1,\ell} + r_{2,\ell} + r_{3,\ell} \bmod p$, and announces s_ℓ to all parties (hence two values are computed and announced).
4. All parties compute the result $v = s_1 + s_2 + s_3 \bmod p$.

To analyze the secure addition protocol, let us first see why the result v is indeed the correct result. This is straightforward:

$$v = \sum_j s_j \bmod p = \sum_j \sum_i r_{i,j} \bmod p = \sum_i \sum_j r_{i,j} \bmod p = \sum_i x_i \bmod p \ .$$

This shows that the protocol computes the sum modulo p of the inputs, no matter how the x_i's are chosen. However, if we let the parties choose $x_i = 1$ for yes and $x_i = 0$ for no, and make sure that $p > 3$, then $\sum_i x_i \bmod p = \sum_i x_i$, because all x_i are 0 or 1 and so their sum cannot be larger than p. So in this case, v is indeed the number of yes-votes.

Now, why is it the case that no new information other than the result v is leaked to any player? Let us concentrate on P_1 for concreteness. Now, in step 1, x_1, x_2 and x_3 are secret shared, and we have already argued above that this tells P_1 nothing whatsoever about x_2, x_3. In the final step, s_1, s_2, s_3 are announced. Note that P_1 already knows s_2, s_3, so s_1 is the only new piece of information. However, we can argue that seeing s_1 will tell P_1 what v is and nothing more. The reason for this is that, if one is given s_2, s_3 and v, one can compute $s_1 = v - s_2 - s_3 \bmod p$. Put another way: given what P_1 is supposed to know, namely v, we can already compute what he sees in the protocol, namely s_1, and therefore seeing the information from the protocol tells him nothing beyond v.

This type of reasoning is formalized later in the book and is called a *simulation argument*: given what a player is supposed to know, we show how to efficiently compute (simulate) everything he sees in the protocol, and from this, we conclude that the protocol tells him nothing beyond what we wanted to tell him.

Note that given the result, P_1 is in fact able to compute some information about other people's votes. In particular, he can compute $v - x_1 = x_2 + x_3$, i.e., the sum of the other players' votes. It is easy to get confused and think that because of this, something must be wrong with the protocol, but in fact there is no problem: it is true that P_1 can compute the sum of the votes of P_2 and P_3, but this follows from information P_1 is *supposed to know*, namely the result and his own input. There is nothing the protocol can do to deprive him of such information – in other words, the best a protocol can do is to make sure players only learn what they are supposed to learn, and this includes whatever can be derived from the player's own input and the intended result.

1.3.2 Secure Multiplication and Match-Making

To do general secure computation, we will of course need to do more than secure addition. It turns out that the secret sharing scheme from the previous subsection already allows us to do more: we can also do secure multiplication.

Suppose two numbers $a, b \in \mathbb{Z}_p$ have been secret shared as described above, so that $a = a_1 + a_2 + a_3 \bmod p$ and $b = b_1 + b_2 + b_3 \bmod p$, and we wish to compute the product $ab \bmod p$ securely. We obviously have

$$ab = a_1b_1 + a_1b_2 + a_1b_3 + a_2b_1 + a_2b_2 + a_2b_3 + a_3b_1 + a_3b_2 + a_3b_3 \bmod p \ .$$

It is now easy to see that if the a_i's and b_i's have been distributed as described above, it is the case that for each product a_ib_j, there is at least one player among the three who knows a_i and b_j and therefore can compute a_ib_j. For instance, P_1 has been given a_2, a_3, b_2, b_3 and can therefore compute a_2b_2, a_2b_3, a_3b_2 and a_3b_3. The situation is, therefore, that the desired result ab is the sum of some numbers where each summand can be computed by at least one of the players. But now we are essentially done, since from Protocol Secure Addition, we already know how to add securely!

The protocol resulting from these observations is shown below. To argue why it works, one first notes that correctness, namely $ab = u_1 + u_2 + u_3 \bmod p$, follows trivially from the above. To show that nothing except $ab \bmod p$ is revealed, one notes that nothing new about a, b is revealed in the first step, and because Protocol Secure Addition is private, nothing except the sum of the inputs is revealed in the last step, and this sum always equals $ab \bmod p$.

It is interesting to note that even in a very simple case where both a and b are either 0 or 1, secure multiplication has a meaningful application: consider two parties Alice and Bob. Suppose Alice is wondering whether Bob wants to go out with her, and also Bob is asking himself if Alice is interested in him. They would very much like to find out if there is mutual interest, but without running the risk of the embarrassment that would result if for instance Bob just tells Alice that he is interested, only to find that Alice turns him down. The problem can be solved if we let Alice choose $a \in Z_p$ where $a = 1$ if she is interested in Bob

Protocol Secure Multiplication

Participants are $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$, input for P_1 is $a \in \mathbb{Z}_p$, input for P_2 is $b \in \mathbb{Z}_p$, where p is a fixed prime agreed upon in advance. P_3 has no input.

1. P_1 distributes shares a_1, a_2, a_3 of a, while P_2 distributes shares b_1, b_2, b_3 of b.
2. P_1 locally computes $u_1 = a_2b_2 + a_2b_3 + a_3b_2 \bmod p$, P_2 computes $u_2 = a_3b_3 + a_1b_3 + a_3b_1 \bmod p$, and P_3 computes $u_3 = a_1b_1 + a_1b_2 + a_2b_1 \bmod p$.
3. The players use Protocol Secure Addition to compute the sum $u_1 + u_2 + u_3 \bmod p$ securely, where P_i uses u_i as input.

and $a = 0$ otherwise. In the same way, Bob chooses b to be 0 or 1. Then we compute the function $f(a, b) = ab \bmod p$ *securely*. It is clear that the result is 1 if and only if there is mutual interest. But on the other hand if, for instance, Alice is not interested, she will choose $a = 0$ and in this case she learns *nothing new* from the protocol. To see why, notice that security of the protocol implies that the only (possibly) new information Alice will learn is the result $ab \bmod p$. But she already knows that result will be 0! In particular, she does not learn whether Bob was interested or not, so Bob is safe from embarrassment. By a symmetric argument, this is of course also the case for Alice.

This argument assumes, of course, that both players choose their inputs honestly according to their real interests. In the following section we discuss what happens if players do not follow the instructions and what we can do about the problems resulting from this.

From Protocol Secure Multiplication, we see that if Alice and Bob play the roles of P_1 and P_2, respectively, they just need to find a third party to help them, to do the multiplication securely. Note that this third party is not a *completely trusted* third party of the kind we discussed before: he does not learn anything about a or b other than $ab \bmod p$. Alice and Bob do have to trust, however, that the third party does not share his information with Bob or with Alice.

It is an obvious question whether one can do secure multiplication such that *only* Alice and Bob have to be involved? The answer turns out to be yes, but then information theoretic security is not possible, as we shall see. Instead one has to use solutions based on cryptography. Such solutions can always be broken if one party has enough computing power, but on the other hand, this is an issue with virtually all the cryptographic techniques we use in practice.

For completeness, we remark that Alice and Bob's problem is a special case of the so-called match-making problem which has somewhat more serious applications than secure dating. Consider a set of companies where each company has a set of other companies it would prefer to do business with. Now we want that each pair of companies finds out whether there is mutual interest, but without forcing companies to reveal their strategy by announcing their interests in public.

Exercise 1.1 Consider the third party helping Alice and Bob to do secure multiplication. Show that the Protocol SECURE MULTIPLICATION is indeed insecure if he reveals what he sees in the protocol to Alice or Bob.

EXERCISE 1.2 We have used replicated secret sharing where each player receives two numbers in $\mathbb{Z}_p$, even though only one secret number is shared. This was to be able to do both secure addition and multiplication, but for secure addition only, something simpler can be done. Use the principle of writing the secret as a sum of random numbers to design a secret sharing scheme for any number of parties, where each party gets as his share only a single number in $\mathbb{Z}_p$. Use your scheme to design a protocol for secure addition. How many players can go together and pool their information before the protocol becomes insecure?

EXERCISE 1.3 You may have asked yourself why Protocol SECURE MULTIPLICATION uses Protocol SECURE ADDITION as a subroutine. Why not just announce u_1, u_2 and u_3, and add them to get the result? The reason is that this would reveal too much information. Show that if P_1 was given u_2, he could - in some cases - compute P_2's input b. What is the probability that he will succeed?

1.3.3 What if Players Do Not Follow Instructions?

Until now we assumed that players always do what they are supposed to. But this is not always a reasonable assumption, since a party may have an interest in doing something different from what he is instructed to do.

There are two fundamentally different ways in which players could deviate from expected behavior: First, they could choose their inputs in a way different from what was expected when the protocol was designed. Second, while executing the protocol, they could do something different from what the protocol instructs them to do. We will consider the two issues separately:

Choice of Inputs

Consider the matchmaking application from above as an example. Let us assume that Alice is not really interested in Bob. We argued above that since her input should then be 0, the output is always 0 and she does not learn whether Bob was interested. The reader may have noticed that there seems to be a way Alice could cheat Bob, if she is willing to choose an input that does not represent her actual interests: she could *pretend* to be interested by choosing $a = 1$, in which case the output will be $ab \bmod p = b$ so she now learns Bob's input and breaks his privacy.

A moment's thought will convince the reader that there is no way we could possibly solve this issue by designing a more secure protocol: whatever the protocol is, a player can of course always choose to execute it with any input he wants to. Since this book is about protocol design, this issue of choice of inputs is out of scope for us.

Therefore, if Bob is worried that Alice might behave as we just described, the only answer we can give is that then he should not play the game at all! If he goes ahead, this has to be based on an assumption that Alice (as well as he himself) has an interest in choosing inputs in a "reasonable way". A possible justification for such an assumption might be that if Alice really thinks Bob is a looser, then the prospect of being stuck with him the rest of the night should be daunting enough to make her choose her input according to her actual preference!

More seriously (and generally): to do secure computation, we have to assume that players have an incentive to provide inputs that will lead to a "meaningful" result they would like to learn. If one can describe and quantify these incentives, it is sometimes possible to analyze what will happen using a different mathematical discipline called game theory, but that is out of scope for this book.

Deviation from the Protocol

Regardless of how the inputs are chosen, it might be that deviating from the protocol could enable a player to learn more information than he was supposed to get, or it could allow him to force the computation to give a wrong result. For instance, this way he could appoint himself the winner of an auction at a low price. This is of course undesirable, but (in contrast to the issue of input choice) this is indeed an issue we can handle.

One solution is to add mechanisms to the protocol which ensure that any deviation from the protocol will be detected. To exemplify this we look at Protocol Secure Addition.

In Protocol Secure Addition we first ask each party to distribute shares of their secret. Looking at P_1, we ask it to pick shares $r_{1,1}, r_{1,2}, r_{1,3}$ such that $x_1 = r_{1,1} + r_{1,2} + r_{1,3} \bmod p$ and then send $r_{1,1}, r_{1,3}$ to P_2 and send $r_{1,1}, r_{1,2}$ to P_3. Here there are two ways to deviate:

First, P_1 could pick $r'_{1,1}, r'_{1,2}, r'_{1,3}$ such that $x_1 \neq r'_{1,1} + r'_{1,2} + r'_{1,3} \bmod p$. This is not a problem, as it just corresponds to having used the input $x'_1 \overset{\text{def}}{=} r'_{1,1} + r'_{1,2} + r'_{1,3} \bmod p$, and as mentioned we cannot (and should not) prevent P_1 from being able to pick any input it desires. The second way to deviate is that P_1 could send $r_{1,1}, r_{1,3}$ to P_2 and send $r'_{1,1}, r_{1,2}$ to P_3 with $r'_{1,1} \neq r_{1,1}$. This is more serious, as now the input x_1 of P_1 is not well-defined. This might or might not lead to an attack, but it is at least a clear deviation from the protocol. There is, however, a simple way to catch this deviation: when P_2 and P_3 receive their shares from P_1, then P_2 sends its value of $r_{1,1}$ to P_3 and P_3 sends its own value of $r_{1,1}$ to P_2. Then they check that they hold the same value. In a similar way P_1 and P_3 can check that P_2 sends consistent shares and P_1 and P_2 can check that P_3 sends consistent shares. In general, having players reveal more information to each other could make a protocol insecure. However, in this case no new information leaks because a player only send information which the receiver should already have.

After the sharing phase, we then ask the parties to add their shares and make the sums public. Looking again at P_1, we ask it to compute s_2 and s_3 and make these public. Here P_1 might deviate by sending, e.g., $s'_2 \neq s_2$. This could lead to a wrong result, $v = s_1 + s'_2 + s_3 \bmod p$. Note, however, that P_3 will compute s_1 and s_2 and make these public. So, both P_1 and P_3 are supposed to make s_2

public. Hence, the players can simply check that P_1 and P_3 make the same value of s_2 public. And similarly they can check that the two versions of s_1 and the two versions of s_3 are identical.

This means Protocol Secure Addition has the following property: if any single party does not do what he is supposed to, the other two players will always be able to detect this. This idea of checking other players to ensure that the protocol is followed is something we will see many times in the following.

1.3.4 Towards General Solutions

We have now seen how to do secure multiplication and addition of numbers in $\mathbb{Z}_p$ – although under various rather strong assumptions. We have assumed that players always follow the protocol (although we have seen a partial answer on how to deal with deviations). Furthermore we have only considered the harm that a *single* player can do to the protocol, and not what happens if several players go together and try, for instance, to compute information on the other players' inputs.

Nevertheless, it is well known that multiplication and addition modulo a prime is sufficient to efficiently simulate any desired computation. This implies, on a very fuzzy and intuitive level, that we can hope to be able to do *any* computation securely, if that computation was feasible in the first place – at least under certain assumptions.

It turns out that this is indeed the case, but of course lots of questions remain. To name a few: how do we define security in a precise way? how do we scale solutions from 3 players to any number of players? what if players do not all follow the protocol? if several players pool their information to learn more than they were supposed to, how many such players can we tolerate and still have a secure protocol? On the following pages, we will arrive at answers to these questions.

2

Preliminaries

Contents

In this chapter we introduce some basic notions used throughout the book, like random variables, families of random variables, interactive systems, and the statistical and computational indistinguishability of these objects. The primary purpose of the chapter is to fix and make precise our notation. A reader who has followed a course in basic cryptography will probably be familiar with most of the notions covered in this chapter, but is encouraged to do a quick reading of the chapter to get familiar with the book's notation. A possible exception is the material on interactive systems. This part is used intensively in Chapter 4. For later use, a cheat sheet with notation can be found in Chapter 12.10.

2.1 Linear Algebra

Let $\mathbb{F}$ be any ring. We use $\mathbf{a} = (a_1, \ldots, a_n)$ to denote column vectors. Let $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{F}^n$, $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{F}^n$ and $\alpha \in \mathbb{F}$. We use the following standard notation from linear algebra.

$$\mathbf{a} + \mathbf{b} = (a_1 + b_1, \ldots, a_n + b_n) \ , \tag{2.1}$$

$$\alpha\mathbf{a} = (\alpha a_1, \ldots, \alpha a_n) \ , \tag{2.2}$$

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i \ , \tag{2.3}$$

$$\mathbf{a} * \mathbf{b} = (a_1 b_1, \ldots, a_n b_n) \ . \tag{2.4}$$

The operator $\cdot$ is called the inner product and $*$ is called the Schur product and entrywise multiplication.

We use capital letters like A to denote matrices. We use a_{ij} to denote the entry in row i, column j in matrix A. We use $\mathbf{a}_i$ and $\mathbf{a}_{i*}$ to denote row i of matrix A, and $\mathbf{a}_{*j}$ to denote column j of A. We use A^{T} for the matrix transpose and AB for matrix multiplication. We write $\ker A$ for the kernel of A.

2.2 Random Variables

The range of a random variable X is the set of elements x for which the random variable X outputs x with probability greater than 0. When we say random variable in this book we always mean *random variable with finite range.* For a random variable X and an element x we use $\Pr[X = x]$ to denote the probability that X outputs x. I.e., for a random variable X there is a finite set D such that $\sum_{x \in D} \Pr[X = x] = 1$. The range of a set of random variables $X_1, \ldots, X_\ell$ is the union of their ranges.

Definition 2.1 (statistical distance) *Let X_0 an X_1 be two random variable with range D. We call*

$$\delta(X_0, X_1) \overset{\text{def}}{=} \frac{1}{2} \sum_{d \in D} |\Pr[X_0 = d] - \Pr[X_1 = d]|$$

the statistical distance between X_0 and X_1.

Statistical distance is also known as total variation distance.

A randomized algorithm A is an algorithm A which takes two inputs, an input x and randomness r. We only consider algorithm which always terminate. We use $A(x; r)$ to denote that we run A on x and r. We use the semi-colon to signal that the part of the input after the semi-colon is not a real input, but an input used to make the random choices that A need to make during the computation. Typically we run A with r being a uniformly random string. So, if A, e.g., needs to flip a fair coin, it uses the next fresh bit from r. If A is a randomized algorithm using randomness from $\{0,1\}^\ell$, then we use $A(x)$ to denote the random variable defined as follows:

1. Sample $r \in_{\mathrm{R}} \{0,1\}^\ell$.
2. Compute $a \leftarrow A(x; r)$.
3. Output a.

We write $a \leftarrow A(x)$ for sampling $A(x)$ in this way.

If X is a random variable and A is a randomized algorithm, then $A(X)$ is the random variable defined as follows:

1. Sample $x \leftarrow X$.
2. Sample $a \leftarrow A(x)$.
3. Output a.

We call two random variables X_0 and X_1 with range D disjoint random variables if it holds for all $x \in D$ that either $\Pr[X_0 = x] = 0$ or $\Pr[X_1 = x] = 0$. I.e., we call them disjoint iff there is no element in the range which they can both output.

PROPOSITION 2.2 *Let A be any (possibly randomized) algorithm and let X_0, X_1, X_2 be any random variables with the same sample space and domain. Then the following holds:*

1. $\delta(X_0, X_1) \in [0, 1]$.
2. $\delta(X_0, X_1) = 0$ *iff* $\Pr[X_0 = x] = \Pr[X_1 = x]$ *for all x.*
3. $\delta(X_0, X_1) = 1$ *iff X_0 and X_1 are disjoint.*
4. $\delta(X_0, X_1) = \delta(X_1, X_0)$.
5. $\delta(X_0, X_2) \leq \delta(X_0, X_1) + \delta(X_1, X_2)$.
6. $\delta((X_0, X_2), (X_1, X_2)) \geq \delta(X_0, X_1)$, *with equality if X_2 is independent of X_0 and X_1.*
7. $\delta(A(X_0), A(X_1)) \leq \delta(X_0, X_1)$.

This, in particular, means that statistical distance is a metric on probability distributions. The last property shows that computation cannot make statistical distance increase.

EXERCISE 2.1 Prove Proposition 2.2.

2.2.1 Distinguishing Advantage of an Algorithm

For an algorithm A and two random variables X_0 and X_1 we need a measure of the ability of A to distinguish between X_0 and X_1. For this purpose we play a game, where we give A either a sample from X_0 or a sample from X_1 and then ask it to guess which random variable it received a sample from.

1. Sample a uniformly random bit $b \leftarrow \{0, 1\}$.
2. Sample $x \leftarrow X_b$.
3. Sample $c \leftarrow A(x)$.

We think of c as A's guess at b. A fixed guess, $c = 0$ say, will allow A to be correct with probability $\frac{1}{2}$ as $b = 0$ with probability $\frac{1}{2}$. When we measure how good A is at distinguishing X_0 from X_1 we are therefore only interested in how much better than $\frac{1}{2}$ it does. For technical reasons we consider the absolute value, i.e., we use the measure $|\Pr[c = b] - \frac{1}{2}|$. Note that this is a value between 0 and $\frac{1}{2}$. Again for technical reason we prefer the measure to be between 0 and 1, so we scale by a factor of 2, and we get the measure $2|\Pr[c = b] - \frac{1}{2}|$, which we call the advantage of A.

Note that if A can always guess b correctly, then $\Pr[c = b] = 1$ and the advantage will be 1. If A makes a random guess, a fixed guess or any other guess independent of b, then $\Pr[c = b] = \frac{1}{2}$ and the advantage will be 0. We therefore think of the advantage as measuring how well A can distinguish X_0 and X_1, with 1 meaning *perfectly* and 0 meaning *not at all.*

Before we make a formal definition it is convenient to rewrite the advantage. For this purpose, observe that if we restrict A to outputting $c \in \{0,1\}$, then

$$\begin{aligned}
\Pr[c=b] &= \frac{1}{2}\Pr[c=b|b=0] + \frac{1}{2}\Pr[c=b|b=1] \\
&= \frac{1}{2}(\Pr[c=0|b=0] + \Pr[c=1|b=1]) \\
&= \frac{1}{2}(\Pr[A(X_0)=0] + \Pr[A(X_1)=1]) \\
&= \frac{1}{2}(\Pr[A(X_0)=0] + (1-\Pr[A(X_1)=0])) \\
&= \frac{1}{2} + \frac{1}{2}(\Pr[A(X_0)=0] - \Pr[A(X_1)=0]) \ ,
\end{aligned}$$

where the first equation is just an application of the law of total probability, and $\Pr[A(X_1)=1] = 1 - \Pr[A(X_1)=0]$ as A outputs either 0 or 1. This means that

$$2\left|\Pr[c=b] - \frac{1}{2}\right| = |\Pr[A(X_0)=0] - \Pr[A(X_1)=0]| \ . \tag{2.5}$$

We can thus think of the advantage as the difference between the probability that A guesses 0 when it sees X_0 and the probability that A guesses 0 when it sees X_1. It is usual to use the rewritten expression for defining the advantage.

Definition 2.3 (advantage) *Let X_0 an X_1 be two random variables. Let A be any algorithm outputting a bit $c \in \{0,1\}$ (we call such an algorithm a* **distinguisher***). The* **advantage** *of A in distinguishing X_0 and X_1 is*

$$\textsc{Adv}_A(X_0,X_1) \overset{\text{def}}{=} |\Pr[A(X_0)=0] - \Pr[A(X_1)=0]| \ .$$

The reason why the rewritten expression is typically used for the formal definition is that it is easily related to the statistical distance between the random variables $A(X_0)$ and $A(X_1)$, which is sometimes convenient when working with the notion of advantage. Namely, note that

$$\begin{aligned}
\delta(A(X_0), A(X_1)) &= \frac{1}{2}\sum_{c=0,1} |\Pr[A(X_0)=c] - \Pr[A(X_1)=c]| \\
&= \frac{1}{2}(|\Pr[A(X_0)=0] - \Pr[A(X_1)=0]| + \\
&\qquad |\Pr[A(X_0)=1] - \Pr[A(X_1)=1]|) \\
&= \frac{1}{2}(|\Pr[A(X_0)=0] - \Pr[A(X_1)=0]| + \\
&\qquad |(1-\Pr[A(X_0)=0]) - (1-\Pr[A(X_1)=0])|) \\
&= \frac{1}{2}(|\Pr[A(X_0)=0] - \Pr[A(X_1)=0]| + \\
&\qquad |\Pr[A(X_1)=0] - \Pr[A(X_0)=0]|) \\
&= \frac{1}{2}(|\Pr[A(X_0)=0] - \Pr[A(X_1)=0]| + \\
&\qquad |\Pr[A(X_0)=0] - \Pr[A(X_1)=0]|) \\
&= |\Pr[A(X_0)=0] - \Pr[A(X_1)=0]| \ .
\end{aligned}$$

This shows that

$$\textsc{Adv}_A(X_0, X_1) = \delta(A(X_0), A(X_1)) \ . \tag{2.6}$$

We can therefore think of the advantage of an algorithm A as the statistical distance between its outputs when it gets X_0 as input respectively when it gets X_1 as input. It is clear that if we allowed A to output an arbitrarily long bit-string, then it could simply output its input. In that case $A(X_0) = X_0$ and $A(X_1) = X_1$ and we would have $\textsc{Adv}_A(X_0, X_1) = \delta(X_0, X_1)$. I.e., the advantage would simply be the statistical distance. We have, however, required that A outputs a single bit, which means that in general the advantage $\textsc{Adv}_A(X_0, X_1)$ could be smaller than the statistical distance $\delta(X_0, X_1)$. In this light, the job of a distinguisher is so to say to boil down the statistical distance between X_0 and X_1 to a single bit.

The relation between advantage and statistical distance is convenient as it tells us that advantage inherits all the nice properties of statistical distance. As an example Proposition 2.2 directly allows us to conclude the following.

Corollary 2.4 *Let A be any (possibly randomized) algorithm and let X_0, X_1, X_2 be any random variables. Then the following holds:*

1. $\textsc{Adv}_A(X_0, X_1) \in [0, 1]$.
2. $\textsc{Adv}_A(X_0, X_1) = 0$ *if X_0 and X_1 are identically distributed.*
3. $\textsc{Adv}_A(X_0, X_1) = \textsc{Adv}_A(X_1, X_0)$.
4. $\textsc{Adv}_A(X_0, X_2) \leq \textsc{Adv}_A(X_0, X_1) + \textsc{Adv}_A(X_1, X_2)$.
5. $\textsc{Adv}_A(X_0, X_1) \leq \delta(X_0, X_1)$.

We have just seen that advantage can be phrased in terms of a statistical distance. Conversely, statistical distance can be phrased in terms of advantage:

It is possible to show that

$$\delta(X_0, X_1) = \max_A \textsc{Adv}_A(X_0, X_1) \ , \tag{2.7}$$

where the maximum is taken over all possible distinguishers. To show this, one considers the particular maximum likelihood distinguisher A_{ML} which on input x outputs $c = 0$ if $\Pr[X_0 = x] \geq \Pr[X_1 = x]$ and otherwise outputs $c = 1$. One can argue that $\delta(A_{\text{ML}}(X_0), A_{\text{ML}}(X_1)) = \delta(X_0, X_1)$, which establishes Eq. 2.7 via Eq. 2.6.

EXERCISE 2.2 Prove Eq. 2.7 by fleshing out the details of the argument following that equation.

2.2.2 Distinguishing Advantage

The notion of an algorithm distinguishing random variables can be extended to classes of algorithms simply by considering the best distinguisher in the class.

DEFINITION 2.5 (ADVANTAGE OF A CLASS OF ALGORITHMS) *Let X_0 an X_1 be two random variables. Let $\mathcal{A}$ be any class of algorithms outputting a bit $c \in \{0, 1\}$. The advantage of $\mathcal{A}$ in distinguishing X_0 and X_1 is*

$$\textsc{Adv}_{\mathcal{A}}(X_0, X_1) \overset{\text{def}}{=} \max_{A \in \mathcal{A}} \textsc{Adv}_A(X_0, X_1) \ .$$

From Eq. 2.7 we know that $\delta(X_0, X_1) = \textsc{Adv}_{\mathcal{A}}(X_0, X_1)$ when $\mathcal{A}$ is the class of all algorithms. In general $\textsc{Adv}_{\mathcal{A}}$ can be much smaller than the statistical distance.

2.3 Families of Random Variables

By a family of random variables we mean a function X from the non-negative integers into random variables. I.e., for each $\kappa \in \mathbb{N}$, $X(\kappa)$ is a random variable. We write a family of random variables as

$$X = \{X(\kappa)\}_{\kappa \in \mathbb{N}} \ .$$

If $X = \{X(\kappa)\}_{\kappa \in \mathbb{N}}$ is a family of random variables and A is a two-input algorithm which takes the security parameter κ as its first input, then we let

$$A(X) \overset{\text{def}}{=} \{A(\kappa, X(\kappa))\}_{\kappa \in \mathbb{N}} \ .$$

I.e., $A(X)$ is again a family of random variables and the random variable associated to κ is $A(\kappa, X(\kappa))$, which is defined by first sampling $x \leftarrow X(\kappa)$ and then running $A(\kappa, x)$ to define the output of $A(\kappa, X(\kappa))$.

2.3.1 Statistical Indistinguishability

We will say that two families of random variables X_0 and X_1 are statistically indistinguishable if the statistical distance between $X_0(\kappa)$ and $X_1(\kappa)$ goes to 0 very quickly as κ grows. By very quickly we will mean that the statistical distance goes to 0 faster than any inverse polynomial – i.e., it is a so-called negligible function, as defined below.

DEFINITION 2.6 (NEGLIGIBLE FUNCTION) *We call a function $\delta : \mathbb{N} \to [0,1]$ negligible in κ if for all $c \in \mathbb{N}$ there exists $\kappa_c \in \mathbb{N}$ such that $\delta(\kappa) \leq \kappa^{-c}$ for all $\kappa \geq \kappa_c$.*

DEFINITION 2.7 (STATISTICAL INDISTINGUISHABILITY) *We say that X_0 and X_1 are statistically indistinguishable, written $X_0 \overset{\text{stat}}{\equiv} X_1$, if $\delta(X_0(\kappa), X_1(\kappa))$ is negligible in κ. If X_0 and X_1 are* not *statistically indistinguishable, we write $X_0 \overset{\text{stat}}{\not\equiv} X_1$.*

We say that X_0 and X_1 are perfectly indistinguishable, written $X_0 \overset{\text{perf}}{\equiv} X_1$, if $\delta(X_0(\kappa), X_1(\kappa)) = 0$ for all κ. If X_0 and X_1 are not *perfectly indistinguishable, we write $X_0 \overset{\text{perf}}{\not\equiv} X_1$.*

Two families of random variables which are perfectly indistinguishable cannot be distinguished by looking at their output, as they make the same output with the same probability. Two families of random variables which are statistically indistinguishable are very close to being perfectly indistinguishable—by moving a negligible amount of probability mass on their output distributions they can be made to be perfectly indistinguishable—and we therefore think of them as being essentially impossible to distinguish by looking at they output. We formalize this later.

When working with indistinguishability of families of random variables, it is convenient to know that the notion is transitive and maintained under computation:

PROPOSITION 2.8 *Let A be any (possibly randomized) algorithm and let X_0, X_1, X_2 be any families of random variables. Then the following holds:*

1. *$X_0 \overset{\text{stat}}{\equiv} X_0$.*
2. *If $X_0 \overset{\text{stat}}{\equiv} X_1$ and $X_1 \overset{\text{stat}}{\equiv} X_2$, then $X_0 \overset{\text{stat}}{\equiv} X_2$.*
3. *If $X_0 \overset{\text{stat}}{\equiv} X_1$, then $A(X_0) \overset{\text{stat}}{\equiv} A(X_1)$.*

The same properties hold for $\overset{\text{perf}}{\equiv}$.

EXERCISE 2.3 Prove Proposition 2.8 using Proposition 2.2.

2.3.2 Distinguishing Advantage of a Class of Algorithms

We now look at the advantage of an algorithm A in distinguishing two families of random variables $X_0 = \{X_0(\kappa)\}_{\kappa \in \mathbb{N}}$ and $X_1 = \{X_0(\kappa)\}_{\kappa \in \mathbb{N}}$. We are interested in how well A can distinguish $X_0(\kappa)$ and $X_1(\kappa)$ as κ grows. For this purpose we define a function

$$\textsc{Adv}_A(X_0, X_1) : \mathbb{N} \to [0, 1]$$

which for each $\kappa \in \mathbb{N}$ measures how well A distinguishes $X_0(\kappa)$ and $X_1(\kappa)$. I.e.,

$$\textsc{Adv}_A(X_0, X_1)(\kappa) \overset{\text{def}}{=} \textsc{Adv}_{A(\kappa,\cdot)}(X_0(\kappa), X_1(\kappa)) ,$$

where $A(\kappa, \cdot)$ is the algorithm that takes one input x and runs $A(\kappa, x)$.

We say that A statistically cannot distinguish X_0 and X_1 if $\textsc{Adv}_{A(\kappa,\cdot)}(X_0(\kappa), X_1(\kappa))$ goes to 0 very quickly as κ grows. We say that A perfectly cannot distinguish X_0 and X_1 if $\textsc{Adv}_{A(\kappa,\cdot)}(X_0(\kappa), X_1(\kappa)) = 0$ for all κ.

Exercise 2.4 Shows that $\textsc{Adv}_A(X_0, X_1)$ is negligible in κ iff $A(X_0) \overset{\text{stat}}{\equiv} A(X_1)$ and that $\textsc{Adv}_A(X_0, X_1)(\kappa) = 0$ for all κ iff $A(X_0) \overset{\text{perf}}{\equiv} A(X_1)$.

The above exercise shows that we really do not need a notion of distinguishing advantage between families of random variables X_0 and X_1. We can simply work with statistical distance between families of random variables $A(X_0)$ and $A(X_1)$. It turns out that it is more convenient to work directly with statistical distance, so this is what we will do. For intuition it is, however, convenient to bare in mind that when A is a distinguisher (an algorithm outputting a single bit) then $A(X_0) \overset{\text{perf}}{\equiv} A(X_1)$ essentially means that A perfectly cannot distinguish X_0 and X_1 and $A(X_0) \overset{\text{stat}}{\equiv} A(X_1)$ means that A only has a negligible advantage in distinguishing X_0 and X_1.

Typically we are in cryptography not interested in whether one *specific* algorithm can distinguish two families of random variables. The distinguisher is typically modeling the adversary, which can have any behavior. We are therefore more interested in whether *some* algorithm can distinguish two families of random variables. We do, however, often restrict the adversary to performing only realistic computation, like at most 2^{100} basic operations. For this purpose, and others, it is convenient to have a notion of indistinguishability for some *class* of distinguishers.

Definition 2.9 (indistinguishability by a class of algorithms) *We let $X_0 = \{X_0(\kappa)\}_{\kappa\in\mathbb{N}}$ and $X_1 = \{X_1(\kappa)\}_{\kappa\in\mathbb{N}}$ be families of random variables and let $\mathcal{A}$ be any class of distinguishers (algorithms outputting a single bit). We say that X_0 and X_1 are indistinguishable by $\mathcal{A}$ if $A(X_0) \overset{\text{stat}}{\equiv} A(X_1)$ for all $A \in \mathcal{A}$. We write this as*

$$X_0 \overset{\mathcal{A}}{\equiv} X_1 .$$

If $X_0 \overset{\mathcal{A}}{\equiv} X_1$, where $\mathcal{A}$ is the class of all poly-time algorithms, then we say that X_0 and X_1 are computationally indistinguishable *and write*

$$X_0 \overset{\text{comp}}{\equiv} X_1 .$$

If X_0 and X_1 are not *computationally indistinguishable, we write $X_0 \overset{\text{comp}}{\not\equiv} X_1$.*

Example 2.1 The following example shows that two families of random variables might be perfectly distinguishable in the statistical sense yet computationally *in*distinguishable.

Let (G, E, D) be a public-key encryption scheme. On input the security parameter κ, the key generator G generates a key-pair $(pk, sk) \leftarrow G(\kappa)$, where pk is the public-key and sk is the secret key and κ specifies the desired security level of the key. On input pk and a message m, the encryption algorithm outputs a ciphertext $C \leftarrow E_{pk}(m)$. Note the E is allowed to use internal randomness and that C depends on this randomness. On input C and sk the decryption algorithm outputs a message $m \leftarrow D_{sk}(C)$. We require that $D_{sk}(C) = m$ when $C \leftarrow E_{pk}(m)$.

For $b \in \{0, 1\}$ let $X_b = \{X_b(\kappa)\}$, where $X_b(\kappa)$ is defined as follows: Sample $(pk, sk) \leftarrow G(\kappa)$, samples $C \leftarrow E_{pk}(b)$ and let $X_b(\kappa) = (pk, C)$. In words, $X_b(\kappa)$ outputs a random public key and a random encryption of b, using κ as the security level.

We first consider the statistical distance between X_0 and X_1. Since a given ciphertext C cannot be both an encryption of 0 and an encryption of 1 there is no value which both $X_0(\kappa)$ and $X_1(\kappa)$ could output, i.e., they are disjoint random variables. By Proposition 2.2 this means that $\delta(X_0(\kappa), X_1(\kappa)) = 1$. This is of course trivial: the distinguisher is given (pk, C) and then simply checks whether C is an encryption of 0 or 1, and outputs the corresponding guess c. It might do this check by doing, e.g., an exhaustive search for sk and then decrypting.

We then consider the "computational distance" between X_0 and X_1. For public-key cryptosystem a standard security notion is that of semantic security. Semantic secure cryptosystems are known to exist under a number of assumptions, like the RSA assumption. The notion of semantic security essentially requires that no efficient algorithm can distinguish an encryption of 0 from an encryption of 1. Technically this is defined by requiring that the distinguishing advantage of any poly-time adversary is negligible in the security parameter. This directly implies that $X_0 \overset{\text{comp}}{\equiv} X_1$.

All in all, this shows that X_0 and X_1 have as large a statistical distance as is possible, yet they are computationally indistinguishable. In particular, $X_0 \overset{\text{perf}}{\not\equiv} X_1$ and $X_0 \overset{\text{stat}}{\not\equiv} X_1$, yet $X_0 \overset{\text{comp}}{\equiv} X_1$. The explanation is that the statistical distance between X_0 and X_1 cannot be noticed by a poly-time distinguisher. $\triangle$

Let $\mathcal{A}$ be any class of algorithms. For any two algorithms A and B let $A \circ B$ be the algorithm which runs as follows on input x:

1. Sample $b \leftarrow B(x)$.
2. Sample $a \leftarrow A(b)$.
3. Return a.

We let $\mathcal{A}^\circ$ be the class of algorithms B for which $\mathcal{A} \circ B \subseteq \mathcal{A}$, by which we simply mean that it holds for all $A \in \mathcal{A}$ that $A \circ B \in \mathcal{A}$. In words, $\mathcal{A}^\circ$ are those algorithms B for which it holds that if it is composed with an algorithm from $\mathcal{A}$ it is again an algorithm from $\mathcal{A}$. As an example, if $\mathcal{A}$ is the set of all poly-time algorithms, then $\mathcal{A}^\circ$ is also the class of all poly-time algorithms.

Theorem 2.10 *Let X_0 and X_1 be families of random variables and let $\mathcal{A}$ be any class of distinguishers. Then the following holds.*

1. $X_0 \overset{\mathcal{A}}{\equiv} X_0$.
2. *If* $X_0 \overset{\mathcal{A}}{\equiv} X_1$ *then* $X_1 \overset{\mathcal{A}}{\equiv} X_0$.
3. *If* $X_0 \overset{\mathcal{A}}{\equiv} X_1$ *and* $X_1 \overset{\mathcal{A}}{\equiv} X_2$*, then* $X_0 \overset{\mathcal{A}}{\equiv} X_2$.
4. *If* $X_0 \overset{\mathcal{A}}{\equiv} X_1$ *and* $B \in \mathcal{A}^\circ$*, then* $B(X_0) \overset{\mathcal{A}}{\equiv} B(X_1)$.
5. *If* $X_0 \overset{\mathcal{A}}{\equiv} X_1$ *and* $B \in \mathcal{A}^\circ$*, then* $B(X_0) \overset{\mathcal{A}}{\equiv} B(X_1)$.

Proof The first three properties follows directly from Proposition 2.2. To show that $B(X_0) \overset{\mathcal{A}}{\equiv} B(X_1)$, we have to show that $A(B(X_0)) \overset{\text{stat}}{\equiv} A(B(X_1))$ for all $A \in \mathcal{A}$. So, let A be any algorithm from $\mathcal{A}$. Since $A \in \mathcal{A}$ and $B \in \mathcal{A}^\circ$ we have that $C \in \mathcal{A}$ when $C = A \circ B$. From $C \in \mathcal{A}$ and $X_0 \overset{\mathcal{A}}{\equiv} X_1$ it follows that $C(X_0) \overset{\text{stat}}{\equiv} C(X_1)$. Combining this with $C(X_b) = (A \circ B)(X_b) = A(B(X_0))$ it follows that $A(B(X_0)) \overset{\text{stat}}{\equiv} A(B(X_1))$ for all $A \in \mathcal{A}$, as desired. □

Exercise 2.5 Show that if $X_0 \overset{\text{comp}}{\equiv} X_1$ and B is a poly-time algorithm, then $B(X_0) \overset{\text{comp}}{\equiv} B(X_1)$.

2.4 Interactive Systems

An interactive agent A is a computational device which receives and sends messages on named ports and which holds an internal state.

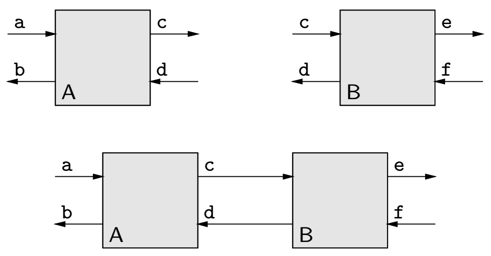


Figure 2.1 An interactive agent A with $\text{In}(A) = \{\texttt{a}, \texttt{d}\}$ and $\text{Out}(A) = \{\texttt{b}, \texttt{c}\}$; An interactive agent B with $\text{In}(B) = \{\texttt{c}, \texttt{f}\}$ and $\text{Out}(B) = \{\texttt{d}, \texttt{e}\}$; And the interactive system $\mathcal{IS} = A \diamond B$ with $\text{Out}(\mathcal{IS}) = \{\texttt{a}, \texttt{f}\}$ and $\text{In}(\mathcal{IS}) = \{\texttt{b}, \texttt{e}\}$.

More formally, an interactive agent is a tuple, $\mathsf{A} = (\text{In}, \text{Out}, \text{State}, \text{Msg}, T, \sigma_0)$, where In is a finite set of names of inports, Out is a finite set of names of outports, State is a set of possible states, Msg is a set of possible messages with at least $0, 1 \in \text{Msg}$ and T is the transition algorithm: it is a randomized algorithm which takes an input (κ, σ, I), where $\kappa \in \mathbb{N}$ is the security parameter, $\sigma \in \text{State}$ is the current state and $I \in \mathrm{Msg}$ or $I \in \mathrm{In}$ or $I = \mathtt{EOQ}$ or $I = \mathtt{SNT}$. If the agent just tried to read on one of its inports, then it receives $I \in \mathrm{Msg}$ if there were messages ready and otherwise $I = \mathtt{EOQ}$. The input $I = \mathtt{SNT}$ is given to the agent when it just sent a message. The input $I \in \mathrm{In}$ is for the first activation of an agent, to tell it on which port it was activated — this is called the activation port. The output of T is of one of the following forms:

- $(\mathtt{send}, \mathtt{P}, m)$, where $\mathtt{P} \in \mathrm{Out}$ is the port to send on and $m \in \mathrm{Msg}$ is the message to send.
- $(\mathtt{read}, \mathtt{P})$, where $\mathtt{P} \in \mathrm{In}$ is the port to read on.
- $(\mathtt{return}, \mathtt{RP})$, where $\mathtt{RP} \in \mathrm{Out}$ is the so-called return port.

The purpose of the return port is to specify which agent to activate next in a larger system. The agent connected to the return port will be activated next.

In the following we let $\mathrm{In}(\mathsf{A})$ be the component In from A, we let $\mathrm{Out}(\mathsf{A})$ be the component Out from A, and we let

$$\mathrm{Ports}(\mathsf{A}) \overset{\text{def}}{=} \mathrm{In}(\mathsf{A}) \cup \mathrm{Out}(\mathsf{A}) \ .$$

Running an agent is called an activation. In each activation the agent can read on ports several times and send on ports several times and update its current state — the initial state is σ_0. At the end of the activation it then specifies a return port. See Algorithm Activate Agent for the details.

Algorithm Activate Agent

The activation of an agent A takes as input the value κ of the security parameter, a current state σ, an activation port $\mathtt{AP} \in \mathrm{In}$ and a queue $\mathrm{Q}_{\mathtt{P}}$ for each $\mathtt{P} \in \mathrm{Ports}(\mathsf{A})$.

```
1. Let $I = \mathtt{AP}$.
2. Let $(\sigma', c) \leftarrow T(\kappa, \sigma, I)$.
3. Update the current state: $\sigma \leftarrow \sigma'$.
4. Process the command $c$ as follows:
   send    If $c = (\mathtt{send}, \mathtt{P}, m)$, then enter $m$ to the end of the queue $\mathrm{Q}_{\mathtt{P}}$, let $I \leftarrow \mathtt{SNT}$ and
           go to Step 2.
   read    If $c = (\mathtt{read}, \mathtt{P})$, then let $I \leftarrow \mathtt{EOQ}$ if the queue $\mathrm{Q}_{\mathtt{P}}$ is empty. Otherwise, remove
           the first element from $\mathrm{Q}_{\mathtt{P}}$ and let $I$ be this element. Then go to Step 2.
   return  If $c = (\mathtt{return}, \mathtt{RP})$, then the activation returns. The output is the new current
           state $\sigma$ and the return port $\mathtt{RP}$. A side effect of the activation is that the
           queues $\{\mathrm{Q}_{\mathtt{P}}\}_{\mathtt{P} \in \mathrm{Ports}(\mathsf{A})}$ were updated.
```

We describe agents it is useful with a little short hand. If we say that an agent *sends the activation token on* $\mathtt{P}$ we mean that it returns on $\mathtt{P}$, i.e., it outputs $(\mathtt{return}, \mathtt{P})$. If we say that and agent *sends* m *on* $\mathtt{P}$ and we don't say anything else we mean that it sends m on $\mathtt{P}$ and then sends the activation token on $\mathtt{P}$. If we describe part of an agent by saying *on input* m *on* $\mathtt{P}$ *do* $A(m)$, we mean that when the agent is activated, it reads on $\mathtt{P}$ and if it gets back $m \neq \mathtt{EOQ}$, then it executes $A(m)$, where $A(m)$ is some action which depends on m.

We are often interested in how efficient agents are to compute, for which the following definition is handy.

Definition 2.11 (responsive, poly-time) *We call an agent* responsive *if it holds for all contexts (i.e., all $\kappa \in \mathbb{N}$, all states $\sigma \in$ State, all activation ports $\mathsf{AP} \in$ In and all queues $\{\mathrm{Q}_{\mathsf{P}}\}_{\mathsf{P} \in \mathrm{Ports}(\mathsf{A})}$) that if we activate A in this context, then it will eventually return. We only allow responsive agents. We call an agent* poly-responsive *if it holds for all contexts that if we activate A in this context, then it will return after having executed at most a number of commands c which is polynomial in κ. We call an agent* step-wise poly-time *if T can be computed in expected poly-time in κ. We call an agent* poly-time *if it is step-wise poly-time and poly-responsive.*

It is straight forward to see that the activation of a poly-time agent can be computed in expected polynomial time.

We connect a set of interactive agents to become an interactive system simply by connecting outports and inports with the same name. For this procedure to be well-defined we need that no two agents have identically named inports, and similarly for outports — we say that they are compatible.

Definition 2.12 *We say that interactive agents $\mathsf{A}_1, \ldots, \mathsf{A}_n$ are* port compatible *if $\forall i, j \in [n], j \neq i : \mathrm{In}(\mathsf{A}_i) \cap \mathrm{In}(\mathsf{A}_j) = \emptyset \wedge \mathrm{Out}(\mathsf{A}_i) \cap \mathrm{Out}(\mathsf{A}_j) = \emptyset$. If $\mathsf{A}_1, \ldots, \mathsf{A}_n$ are port compatible we call $\mathcal{IS} = \{\mathsf{A}_1, \ldots, \mathsf{A}_n\}$ an* interactive system. *We say that interactive systems are* port compatible *if all their interactive agents are port compatible. If two interactive systems $\mathcal{IS}_1$ and $\mathcal{IS}_2$ are port compatible, then we define their* composition *to be $\mathcal{IS}_1 \diamond \mathcal{IS}_2 \stackrel{\text{def}}{=} \mathcal{IS}_1 \cup \mathcal{IS}_2$; Otherwise we let $\mathcal{IS}_1 \diamond \mathcal{IS}_2 = \perp$. For any interactive system $\mathcal{IS}$ we let $\mathcal{IS} \diamond \perp \stackrel{\text{def}}{=} \perp$ and $\perp \diamond \mathcal{IS} \stackrel{\text{def}}{=} \perp$.*

The following proposition is convenient when reasoning about composed interactive systems.

Proposition 2.13 *The following holds for all interactive systems $\mathcal{IS}_1, \mathcal{IS}_2, \mathcal{IS}_3$:*

1. $\mathcal{IS}_1 \diamond \mathcal{IS}_2 = \mathcal{IS}_2 \diamond \mathcal{IS}_1$.
2. $(\mathcal{IS}_1 \diamond \mathcal{IS}_2) \diamond \mathcal{IS}_3 = \mathcal{IS}_1 \diamond (\mathcal{IS}_2 \diamond \mathcal{IS}_3)$.

For an interactive system $\mathcal{IS}$ and a port name P which is an inport of an agent in $\mathcal{IS}$, we use A_{P} to denote the agent from $\mathcal{IS}$ which has an inport named P.

For an interactive system $\mathcal{IS}$ we let $\mathrm{In}(\mathcal{IS})$ be the inports which are not connected to outports and we let $\mathrm{Out}(\mathcal{IS})$ be the outports which are not connected to inports. I.e.,

$$\mathrm{In}(\mathcal{IS}) \stackrel{\text{def}}{=} (\cup_{\mathsf{A} \in \mathcal{IS}} \mathrm{In}(\mathsf{A})) \setminus (\cup_{\mathsf{A} \in \mathcal{IS}} \mathrm{Out}(\mathsf{A})) ,$$

$$\mathrm{Out}(\mathcal{IS}) \stackrel{\text{def}}{=} (\cup_{\mathsf{A} \in \mathcal{IS}} \mathrm{Out}(\mathsf{A})) \setminus (\cup_{\mathsf{A} \in \mathcal{IS}} \mathrm{In}(\mathsf{A})) .$$

We call $\mathrm{In}(\mathcal{IS})$ and $\mathrm{Out}(\mathcal{IS})$ the open ports of the system. We say that an interactive system is closed if $\mathrm{In}(\mathcal{IS}) = \emptyset$ and $\mathrm{Out}(\mathcal{IS}) = \emptyset$. We say that $\mathcal{IS}$ is executable if it is closed and there is some agent A in $\mathcal{IS}$ which has an inport named ϵ and an outport named ϵ — this is a technical requirement to have some well-defined initial activation port and final return port: the execution will begin by activating on the port ϵ and will end the first time an agent returns on the port ϵ.

Algorithm EXECUTE SYSTEM

```
1. Initialize the current state of all agents to be the initial state: For all $\mathsf{A} \in \mathcal{IS}$, do
   $\sigma_{\mathsf{A}} \leftarrow \sigma_0(\mathsf{A})$
2. Initialize an empty queue for all ports: For all $\mathtt{P} \in \cup_{\mathsf{A}\in\mathcal{IS}} \mathrm{Ports}(\mathsf{A})$, do $\mathrm{Q}_{\mathtt{P}} \leftarrow \epsilon$ .
3. Let the initial activation port be $\mathtt{AP}$.
4. Let $\mathsf{A}_{\mathtt{AP}}$ denote the agent with an inport named $\mathtt{AP}$.
5. Activate $\mathsf{A}_{\mathtt{AP}}$ on $(\kappa, \sigma_{\mathsf{A}}, \mathtt{AP}, \{\mathrm{Q}_{\mathtt{P}}\}_{\mathtt{P}\in\mathrm{Ports}(\mathsf{A})})$ — see Algorithm ACTIVATE AGENT. This
   will update the queues $\{\mathrm{Q}_{\mathtt{P}}\}_{\mathtt{P}\in\mathrm{Ports}(\mathsf{A})}$ and make $\mathsf{A}_{\mathtt{AP}}$ output a return port $\mathtt{RP}$ and a
   new current state $\sigma'$.
6. Update the current state of $\mathsf{A}_{\mathtt{AP}}$: $\sigma_{\mathsf{A}} \leftarrow \sigma'$.
7. Update the activation port: $\mathtt{AP} \leftarrow \mathtt{RP}$.
8. If $\mathtt{RP} \neq \epsilon$, go to Step 4. If $\mathtt{RP} = \epsilon$, then the execution stops with output $m$, where
   $m = 0$ if $\mathrm{Q}_\epsilon$ is empty and were $m$ is the front element in $\mathrm{Q}_\epsilon$ otherwise.
```

The execution is activation driven. In each step we activate an agent. Initially we activate the agent A_ϵ with inport ϵ, and we do it on the port ϵ. After this, the next agent to be activated is the one which has an inport with the name that the previous agent specified as return port, and the agent is activated on that port. We think of this as some activation token @ being passed around.

DEFINITION 2.14 (EXECUTION OF INTERACTIVE SYSTEM) *An interactive system is called executable if it is closed and contains an agent A with $\epsilon \in \mathrm{In}(A)$ and $\epsilon \in \mathrm{Out}(A)$. For an executable interactive system $\mathcal{IS}$ and a given value $\kappa \in \mathbb{N}$ of the security parameter, we use $\mathcal{IS}(\kappa)$ to denote the message m output in Step 8 in Algorithm* EXECUTE SYSTEM *when executing $\mathcal{IS}$ as specified in Algorithm* EXECUTE SYSTEM. *If the execution does not stop, such that m is not defined, we let $\mathcal{IS}(\kappa) = \infty$ for some reserved symbol ∞. Note that this makes $\mathcal{IS}(\kappa)$ a random variable with range* $\mathrm{Msg} \cup \{\infty\}$, *where* Msg *is the message space of A_ϵ. We sometimes use $\mathcal{IS}$ to denote the family of random variables $\{\mathcal{IS}(\kappa)\}_{\kappa\in\mathbb{N}}$.*

EXAMPLE 2.2 Let A be an interactive agent with $\mathrm{In}(\mathsf{A}) = \{\epsilon, \mathtt{1}\}$ and $\mathrm{Out}(\mathsf{A}) = \{\epsilon, \mathtt{1}\}$. On any input on ϵ or $\mathtt{1}$ it flips $r \in \{0,1\}^\ell$ for $\ell = \lceil \log_2(\kappa) \rceil$ and interprets it as an integer $R \in \mathbb{Z}_{2^\ell}$. If $R < \kappa$, then it sends R on ϵ. Otherwise, it sends `retry` on $\mathtt{1}$. Let $\mathcal{IS} = \{\mathsf{A}\}$. Then $\mathcal{IS}$ is executable and $\Pr[\mathcal{IS}(\kappa) = \infty] = 0$, so $\mathcal{IS}(\kappa)$ is a random variable with range $\mathbb{Z}_\kappa$, and it is uniformly random on that range. △

EXAMPLE 2.3 Let A_1 be an interactive agent with $\mathrm{In}(\mathsf{A}_1) = \{\epsilon, \mathtt{1}\}$ and $\mathrm{Out}(\mathsf{A}_1) = \{\epsilon, \mathtt{2}\}$. On any input on ϵ it samples a uniformly random bit $c \in \{0,1\}$ and sends c on $\mathtt{2}$. On input $e \in \{0,1\}$ on $\mathtt{1}$ it sends e on ϵ. Let A_2 be an interactive agent with $\mathrm{In}(\mathsf{A}_2) = \{\mathtt{2}\}$ and $\mathrm{Out}(\mathsf{A}_2) = \{\mathtt{1}\}$. On input $c \in \mathbb{N}$ on $\mathtt{2}$, it samples $d \leftarrow \{0,1\}$ and sends $c + d$ on $\mathtt{1}$. Let $\mathcal{IS} = \{\mathsf{A}_1, \mathsf{A}_2\}$. Then $\mathcal{IS}$ is executable and $\Pr[\mathcal{IS}(\kappa) = 0] = \frac{1}{4}$, $\Pr[\mathcal{IS}(\kappa) = 1] = \frac{1}{2}$, and $\Pr[\mathcal{IS}(\kappa) = \infty] = \frac{1}{4}$. $\triangle$

We will not only be interested in the execution of closed system. To allow to talk about the execution of an open system we introduce the notion of a closure.

Definition 2.15 (closure) *Let $\mathcal{IS}$ be an interactive system. A closure of $\mathcal{IS}$ is an interactive system $\mathcal{Z}$ where $\mathcal{IS} \diamond \mathcal{Z} \neq \perp$ and for which $\mathcal{IS} \diamond \mathcal{Z}$ is executable. This, in particular, means that $\mathrm{In}(\mathcal{Z}) = \mathrm{Out}(\mathcal{IS})$ and $\mathrm{Out}(\mathcal{Z}) = \mathrm{In}(\mathcal{IS})$.*

We reserve the port name ϵ for closures, i.e., we assume that normal interactive systems like $\mathcal{IS}$ do not use this port name for internal communication. The reason is that we want the closure $\mathcal{Z}$ to be the one that defines the output of the execution.

Above we implicitly defined the output of an execution to be ∞ if the execution runs forever. In some of our later definitions, in particular the definition of indistinguishability of interactive systems, handling systems which run forever is definitional cumbersome. Since we do not need to study such systems we will restrict our attention to systems which do not run forever. We call such systems responsive. Sometimes we also need to restrict systems to be poly-time.

Definition 2.16 (responsive, poly-time) *We call an executable interactive system $\mathcal{IS}$ responsive if $\Pr[\mathcal{IS}(\kappa) = \infty] = 0$ for all κ.*

Let $\mathcal{IS}$ be an open interactive system not using the port name ϵ. We call $\mathcal{IS}$ responsive if it holds for all closures $\mathcal{Z}$ of $\mathcal{IS}$ and all values $\kappa \in \mathbb{N}$ of the security parameter that whenever the activation token is turned over from an agent in $\mathcal{Z}$ to an agent in $\mathcal{IS}$, in the execution $(\mathcal{IS} \diamond \mathcal{Z})(\kappa)$, then with probability 1 the activation token will later be turned over from an agent in $\mathcal{IS}$ to an agent in $\mathcal{Z}$. In words, the probability that $\mathcal{IS}$ at some point keeps the activation token forever is 0.

We say that $\mathcal{IS}$ is poly-responsive if there exists a polynomial $\tau : \mathbb{N} \to \mathbb{N}$ such that it holds for all closures $\mathcal{Z}$ of $\mathcal{IS}$ and all values $\kappa \in \mathbb{N}$ of the security parameter that whenever the activation token is turned over from an agent in $\mathcal{Z}$ to an agent in $\mathcal{IS}$, in the execution $(\mathcal{IS} \diamond \mathcal{Z})(\kappa)$, then the expected number of activations of agents in $\mathcal{IS}$ until the activation token is again turned over from an agent in $\mathcal{IS}$ to an agent in $\mathcal{Z}$ is bounded by $\tau(\kappa)$. In other words, when $\mathcal{IS}$ is activated on an open inport, then the expected number of internal agent activations before it returns the activation token on an open outport is polynomial.

We say that $\mathcal{IS}$ is a poly-time interactive system if all agents are poly-time and $\mathcal{IS}$ is poly-responsive.

The main important fact to remember about this definition is the following:

Proposition 2.17 *A poly-time interactive system $\mathcal{IS}$ can be executed in expected polynomial time. More precisely, for any activation of $\mathcal{IS}(\kappa)$ the expected running* *time spent until the activation token is returned on an open port grows at most polynomially in κ.*

EXERCISE 2.6 Prove Proposition 2.17. The crucial step is the following:

Let $B(\kappa, \cdot)$ be a stateful algorithm taking the security parameter as input plus an additional input y. Stateful means, as usual, that B keeps a state that is stored between calls to B and the output depends on the state as well as the input. Assume that B runs in expected polynomial time. I.e., there exists a polynomial $q(\kappa)$ such that the expected running time of $B(\kappa, y)$ is bounded by $q(\kappa)$ for all states and inputs y.

Let $A(\kappa, \cdot)$ be an algorithm taking the security parameter as input plus an additional input x, and assume that $A(\kappa, \cdot)$ uses $B(\kappa, \cdot)$ as sub-routine, i.e., $A(\kappa, x)$ might call $B(\kappa, y)$, and it might do so several times and on different inputs y. Assume that $A(\kappa, \cdot)$ calls $B(\kappa, \cdot)$ an expected polynomial number of times. In other words, there exists a polynomial $p(\kappa)$ such that the expected number of times that $A(\kappa, x)$ calls $B(\kappa, \cdot)$ is bounded by $p(\kappa)$ for all inputs x. Let $t(\kappa, x)$ be the expected value of the sum of the running times of all the calls that $A(\kappa, x)$ makes to $B(\kappa, \cdot)$.

As a first step in your proof, you should show that there exists a polynomial $t(\kappa)$ such that $t(\kappa, x) \leq t(\kappa)$ for all inputs x.

2.4.1 Indistinguishable Interactive Systems

We say that two interactive systems are (behaviorly) indistinguishable if one cannot tell the difference between them by sending and receiving messages over the open ports of the systems. For this purpose we work with the notion of an environment, which is just an interactive system which closes the system and makes it executable. We think of the execution as the environment playing with the system over the open ports. Its job is then to guess which system it is connected to. The closure therefore acts as a distinguisher. For this reason we require that it outputs a bit c, which is its guess at which system it is playing with.

DEFINITION 2.18 *Let $\mathcal{IS}_0$ and $\mathcal{IS}_1$ be responsive interactive systems with* $\mathrm{In}(\mathcal{IS}_0) = \mathrm{In}(\mathcal{IS}_1)$ *and* $\mathrm{Out}(\mathcal{IS}_0) = \mathrm{Out}(\mathcal{IS}_1)$*. We call an interactive system $\mathcal{Z}$ an* environment *for $\mathcal{IS}_0$ and $\mathcal{IS}_1$ if $\mathcal{Z}$ is a closure of both $\mathcal{IS}_0$ and $\mathcal{IS}_1$ and $\mathcal{IS}_0 \diamond \mathcal{Z}$ and $\mathcal{IS}_1 \diamond \mathcal{Z}$ are responsive.*[1] *We also require that when $\mathcal{Z}$ produces an output on the port ϵ, then it is a bit c. We call ϵ the* guess *of the environment. The responsiveness, and the fact that $\mathcal{Z}$ outputs a bit on ϵ means that $(\mathcal{IS}_0 \diamond \mathcal{Z})(\kappa)$ and $(\mathcal{IS}_1 \diamond \mathcal{Z})(\kappa)$ are random variables with range $\{0, 1\}$.*

- *We say that the systems are perfectly indistinguishable, written $\mathcal{IS}_0 \overset{\text{perf}}{\equiv} \mathcal{IS}_1$, if*

[1] Since we require that $\mathcal{IS}_0$ and $\mathcal{IS}_1$ are responsive, the requirement that $\mathcal{IS}_0 \diamond \mathcal{Z}$ and $\mathcal{IS}_1 \diamond \mathcal{Z}$ are responsive just requires that $\mathcal{Z}$ does not make an infinite number of internal activations.

it holds that $\{(\mathcal{Z} \diamond \mathcal{IS}_0)(\kappa)\}_{\kappa\in\mathbb{N}} \overset{\text{perf}}{\equiv} \{(\mathcal{Z} \diamond \mathcal{IS}_1)(\kappa)\}_{\kappa\in\mathbb{N}}$ *for all environments* $\mathcal{Z}$ *for* $\mathcal{IS}_0$ *and* $\mathcal{IS}_1$.

- *We say that the systems are statistically indistinguishable, written* $\mathcal{IS}_0 \overset{\text{stat}}{\equiv} \mathcal{IS}_1$, *if it holds that* $\{(\mathcal{Z} \diamond \mathcal{IS}_0)(\kappa)\}_{\kappa\in\mathbb{N}} \overset{\text{stat}}{\equiv} \{(\mathcal{Z} \diamond \mathcal{IS}_1)(\kappa)\}_{\kappa\in\mathbb{N}}$ *for all environments* $\mathcal{Z}$ *for* $\mathcal{IS}_0$ *and* $\mathcal{IS}_1$, *such that* $\mathcal{Z}$ *makes a polynomial (in* κ*) number of activations of* $\mathcal{IS}_0$ *(*$\mathcal{IS}_1$*).*
- *For a class of interactive systems* Env *we say that* $\mathcal{IS}_0$ *is indistinguishable from* $\mathcal{IS}_1$ *for* Env *(written* $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_1$*) if* $\{(\mathcal{Z} \diamond \mathcal{IS}_0)(\kappa)\}_{\kappa\in\mathbb{N}} \overset{\text{stat}}{\equiv} \{(\mathcal{Z} \diamond \mathcal{IS}_1)(\kappa)\}_{\kappa\in\mathbb{N}}$ *for all* $\mathcal{Z} \in \text{Env}$ *for which* $\mathcal{Z}$ *is an environment for* $\mathcal{IS}_0$ *and* $\mathcal{IS}_1$.
- *We say that* $\mathcal{IS}_0 \overset{\text{comp}}{\equiv} \mathcal{IS}_1$ *if* $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_1$ *for the class* Env *of poly-time interactive systems.*

The case of statistical indistinguishability is meant to capture systems $\mathcal{IS}_0, \mathcal{IS}_1$ that behave "almost" in the same way, for instance where the behavior is exactly the same, except if some event E occurs, where $\Pr[E]$ is negligible. To capture such cases, we need the restriction to a polynomial number of activations. Otherwise an unbounded environment could distinguish easily: it would just keep activating the system until the "error event" occurs.

For a class of interactive systems Env we let $\text{Env}^\diamond$ be the class of interactive systems $\mathcal{IS}$ for which $\mathcal{Z} \diamond \mathcal{IS} \in \text{Env}$ whenever $\mathcal{Z} \in \text{Env}$ and $\mathcal{Z}$ and $\mathcal{IS}$ are port compatible.

Proposition 2.19 *The following holds for all interactive systems* $\mathcal{IS}_0, \mathcal{IS}_1, \mathcal{IS}_2$ *and all classes of environments* Env.

1. $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_0$.
2. *If* $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_1$, *then* $\mathcal{IS}_1 \overset{\text{Env}}{\equiv} \mathcal{IS}_0$.
3. *If* $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_1$ *and* $\mathcal{IS}_1 \overset{\text{Env}}{\equiv} \mathcal{IS}_2$, *then* $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_2$.
4. *If* $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_1$ *and* $\mathcal{IS}_2 \in \text{Env}^\diamond$, *then* $\mathcal{IS}_2 \diamond \mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_2 \diamond \mathcal{IS}_1$.
5. *If* $\mathcal{IS}_0 \overset{\text{Env}_1}{\equiv} \mathcal{IS}_1$ *and* $\mathcal{IS}_2 \diamond \text{Env}_2 \subseteq \text{Env}_1$, *then* $\mathcal{IS}_2 \diamond \mathcal{IS}_0 \overset{\text{Env}_2}{\equiv} \mathcal{IS}_2 \diamond \mathcal{IS}_1$.

The same properties hold for $\overset{\text{perf}}{\equiv}$, $\overset{\text{stat}}{\equiv}$ *and* $\overset{\text{comp}}{\equiv}$.

Proof The first three properties are straight forward. To show that $\mathcal{IS}_2 \diamond \mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_2 \diamond \mathcal{IS}_1$, we have to show that $\mathcal{Z} \diamond (\mathcal{IS}_2 \diamond \mathcal{IS}_0) \overset{\text{stat}}{\equiv} \mathcal{Z} \diamond (\mathcal{IS}_2 \diamond \mathcal{IS}_1)$ for all $\mathcal{Z} \in \text{Env}$ which are environments for $\mathcal{IS}_2 \diamond \mathcal{IS}_0$ and $\mathcal{IS}_2 \diamond \mathcal{IS}_1$. Since $\mathcal{Z} \in \text{Env}$ and $\mathcal{IS}_2 \in \text{Env}^\diamond$, we have that $\mathcal{Z} \diamond \mathcal{IS}_2 \in \text{Env}$. From $\mathcal{Z} \diamond \mathcal{IS}_2 \in \text{Env}$ and $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_1$ it follows that $(\mathcal{Z} \diamond \mathcal{IS}_2) \diamond \mathcal{IS}_0 \overset{\text{stat}}{\equiv} (\mathcal{Z} \diamond \mathcal{IS}_2) \diamond \mathcal{IS}_1$. The claim then follows from $(\mathcal{Z} \diamond \mathcal{IS}_2) \diamond \mathcal{IS}_b = \mathcal{Z} \diamond (\mathcal{IS}_2 \diamond \mathcal{IS}_b)$ for $b = 0, 1$. The proof of the last property follows in the same manner. □

We are particularly interested in Property 4 in the above proposition. It basically says that if two interactive systems are hard to distinguish, then this also holds if we use them in some context $\mathcal{IS}_2$. I.e., as long as the context $\mathcal{IS}_2$ does not perform any computation that the class of distinguishers that we consider cannot perform on its own. This is very useful in comparing systems like $\mathcal{IS}_2 \diamond \mathcal{IS}_0$ and $\mathcal{IS}_2 \diamond \mathcal{IS}_1$, as we only have to compare their non-common parties, $\mathcal{IS}_0$ and $\mathcal{IS}_1$. This is sometimes called modularity of indistinguishability and sometimes called composability of indistinguishability.

In Property 4 we showed composability only for $\mathcal{IS}_2 \in \text{Env}^{\diamond}$. This is, in fact, the best general result we can hope for. If, e.g., $\mathcal{Z}$ is the class of poly-time environments and $\mathcal{IS}_2$ performs exponential time computations, then it might happen that $\mathcal{IS}_0 \overset{\text{Env}}{\equiv} \mathcal{IS}_1$ but $\mathcal{IS}_2 \diamond \mathcal{IS}_0 \overset{\text{Env}}{\not\equiv} \mathcal{IS}_2 \diamond \mathcal{IS}_1$: it might be that $\mathcal{IS}_0$ and $\mathcal{IS}_1$ are indistinguishable to poly-time environments due to the use of cryptography but that $\mathcal{IS}_2$ breaks this cryptography using its exponential time computing power. The reason why this does not contradict the proposition is that when Env is the class of poly-time environments and $\mathcal{IS}_2$ is an exponential time system, then $\mathcal{IS}_2 \notin \text{Env}^{\diamond}$.[2]

Example 2.4 Let A be an interactive agent with $\text{In}(A) = \{\mathtt{a}, \mathtt{d}\}$ and $\text{Out}(A) = \{\mathtt{b}, \mathtt{c}\}$ and the following behavior: On input $x \in \mathbb{Z}$ on $\mathtt{a}$ output $x+1$ on $\mathtt{c}$, and on input $x \in \mathbb{Z}$ on $\mathtt{d}$ output $x+1$ on $\mathtt{b}$. Let B be an interactive agent with $\text{In}(B) = \{\mathtt{c}\}$ and $\text{Out}(B) = \{\mathtt{d}\}$ and the following behavior: On input $x \in \mathbb{Z}$ on $\mathtt{c}$ output $2x$ on $\mathtt{d}$. Let C be an interactive agent with $\text{In}(C) = \{\mathtt{a}\}$ and $\text{Out}(C) = \{\mathtt{b}\}$ and the following behavior: On input $x \in \mathbb{Z}$ on $\mathtt{a}$ output $2x+3$ on $\mathtt{b}$. Let $\mathcal{IS}_0 = \{A, B\}$ and let $\mathcal{IS}_1 = \{C\}$. Then $\mathcal{IS}_0 \overset{\text{perf}}{\equiv} \mathcal{IS}_1$. △

Exercise 2.7 Prove Property 5 in Proposition 2.19.

2.5 Public-Key Cryptosystems

The following section precisely defines the security of public key cryptosystems, and we will use such systems later, but at the same time gives an example of how interactive systems and the notion of indistinguishability of interactive systems can be used to make precise definitions.

A public-key cryptosystem consists of three algorithms (G, E, D) called the key generator, the encryption algorithm and the decryption algorithm, respectively.

We define security of a public-key cryptosystem by comparing two interactive agents. For $b = 0, 1$, let A_b be the following interactive agent:

- $\text{In}(\mathsf{A}_b) = \{\mathtt{keygen}, \mathtt{msgs}, \mathtt{dec}\}$.
- $\text{Out}(\mathsf{A}_b) = \{\mathtt{pk}, \mathtt{target}, \mathtt{plaintext}\}$.
- It behaves as follows:
  - The first time a message is input on `keygen` it samples $(pk, sk) \leftarrow G(\kappa)$ and outputs pk on `pk`. It ignores all subsequent messages on `keygen`.

[2] Assume namely that $\mathcal{IS}_2 \in \text{Env}^{\diamond}$. By definition, this means that $\mathcal{IS}_2 \diamond \mathcal{IS} \in \text{Env}$ for all $\mathcal{IS} \in \text{Env}$. Since the empty interactive system $\mathcal{IS} = \emptyset$ is clear poly-time we have that $\emptyset \in \text{Env}$, which implies that $\mathcal{IS}_2 \diamond \emptyset = \mathcal{IS}_2 \in \text{Env}$. This contradicts the premises that Env is the poly-time systems and that $\mathcal{IS}_2$ is an exponential time system.

– On the first message of the form (m_0, m_1) with $|m_0| = |m_1|$ on `msgs` after (pk, sk) has been sampled, it samples a target ciphertext $c^* \leftarrow E_{pk}(m_b)$ and returns c^* on `target`. It ignores all subsequent messages on `msgs`.
– On each message c on `dec`, after (pk, sk) has been sampled, it returns $m \leftarrow D_{sk}(c)$ on `plaintext`. If c^* has been defined, then it ignores inputs with $c = c^*$.

The following definition captures the standard notions of indistinguishability under chosen ciphertext attack and indistinguishability under chosen plaintext attack.

Definition 2.20 *We say that (G, E, D) is IND-CCA secure if $\mathsf{A}_0 \overset{\text{comp}}{\equiv} \mathsf{A}_1$. We say that (G, E, D) is IND-CPA secure if $\mathsf{A}_0' \overset{\text{comp}}{\equiv} \mathsf{A}_1'$, where A_b' is A_b with the ports `dec` and `plaintext` removed.*

The notion of IND-CPA (which is sometimes called semantic security) says that it is impossible to distinguish encryptions of different messages in poly-time, even if you get to pick the messages yourself, and even if you get to pick the messages after seeing the public key. The notion of IND-CCA says that the task remains hard even if you get access to a decryption oracle, unless of course you decrypt the ciphertext you were given as challenge (which would make the task trivial).

It is possible to build cryptosystems which are IND-CPA secure and IND-CCA secure under many assumptions, e.g., the RSA assumption and the discrete logarithm assumption.

3

MPC Protocols with Passive Security

Contents

3.1 Introduction

In Chapter 1, we gave an introduction to the multiparty computation problem and explained what it intuitively means for a protocol to compute a given function securely, namely, we want a protocol that is correct and private. However, we only considered 3 players, we only argued that a single player does not learn more than he is supposed to, and finally we assumed that all players would follow the protocol.

In this chapter, we will consider a more general solution to multiparty computation, where we remove the first two restrictions: we will consider any number $n \geq 3$ of players, and we will be able to show that as long as at most $t < n/2$ of the players go together after the protocol is executed and pool all their information, they will learn nothing more than their own inputs and the outputs they were supposed to receive, even if their computing power is unbounded. We will still assume, however, that all players follow the protocol. This is known as *semi-honest* or *passive* security.

To argue security in this chapter, we will use a somewhat weak but very simple definition that only makes sense for semi-honest security. We then extend this in the next chapter to a fully general model of what protocols are, and what security means.

Throughout this chapter, we will assume that each pair of players can communicate using a perfectly secure channel, so if two players exchange data, a third player has no information at all about what is sent. Such channels might be available because of physical circumstances, or we can implement them relative to a computational assumption using cryptography. For more details on this, see Chapter 7.

We end the chapter by showing that the bound $t < n/2$ is optimal: if $t \geq n/2$, then some functions cannot be computed securely.

3.2 Secret Sharing

Our main tool to build the protocol will be so called secret-sharing schemes. The theory of secret sharing schemes is a large and interesting field in its own right with many applications to MPC, and we look at this in more detail in Chapter 11, where a formal definition of the notion as well as a general treatment of the theory of secret-sharing can be found.

Here we concentrate on a particular example scheme that will be sufficient for our purposes in this chapter, namely Shamir's secret sharing scheme. This scheme is based on polynomials over a finite field $\mathbb{F}$. The only necessary restriction on $\mathbb{F}$ is that $|\mathbb{F}| > n$, but we will assume for concreteness and simplicity that $\mathbb{F} = \mathbb{Z}_p$ for some prime $p > n$.

A value $s \in \mathbb{F}$ is *shared* by choosing a random polynomial $f_s(\mathtt{X}) \in \mathbb{F}[\mathtt{X}]$ of degree at most t such that $f_s(0) = s$. And then sending privately to player P_j the share $s_j = f_s(j)$. The basic facts about this method are that any set of t or fewer shares contain no information on s, whereas it can easily be reconstructed from any $t+1$ or more shares. Both of these facts are proved using Lagrange interpolation.

Lagrange Interpolation

If $h(\mathtt{X})$ is a polynomial over $\mathbb{F}$ of degree at most l and if C is a subset of $\mathbb{F}$ with $|C| = l + 1$, then

$$h(\mathtt{X}) = \sum_{i \in C} h(i)\delta_i(\mathtt{X}) \ ,$$

where $\delta_i(\mathtt{X})$ is the degree l polynomial such that, for all $i, j \in C$, $\delta_i(j) = 0$ if $i \neq j$ and $\delta_i(j) = 1$ if $i = j$. In other words,

$$\delta_i(\mathtt{X}) = \prod_{j \in C, j \neq i} \frac{\mathtt{X} - j}{i - j} \ .$$

We briefly recall why this holds. Since each $\delta_i(\mathtt{X})$ is a product of l monomials, it is a polynomial of degree at most l. Therefore the right hand side $\sum_{i \in C} h(i)\delta_i(\mathtt{X})$ is a polynomial of degree at most l that on input i evaluates to $h(i)$ for $i \in C$. Therefore, $h(\mathtt{X}) - \sum_{i \in C} h(i)\delta_i(\mathtt{X})$ is 0 on all points in C. Since $|C| > l$ and only the zero-polynomial has more zeroes than its degree (in a field), it follows that $h(\mathtt{X}) - \sum_{i \in C} h(i)\delta_i(\mathtt{X})$ is the zero-polynomial, from which it follows that $h(\mathtt{X}) = \sum_{i \in C} h(i)\delta_i(\mathtt{X})$.

Using the same argument, one easily sees that the Lagrange interpolation works, even if no polynomial is predefined. Given any set of values $\{y_i \in \mathbb{F} |\ i \in C\}$,

$|C| = l+1$, we can construct a polynomial h with $h(i) = y_i$ of degree at most l as

$$h(\mathtt{X}) = \sum_{i \in C} y_i \delta_i(\mathtt{X}) \ .$$

A consequence of Lagrange interpolation is that there exist easily computable values $\mathbf{r} = (r_1, ..., r_n)$, such that

$$h(0) = \sum_{i=1}^{n} r_i h(i) \tag{3.1}$$

for all polynomials $h(X)$ of degree at most $n-1$. Namely, $r_i = \delta_i(0)$. We call $(r_1, ..., r_n)$ the recombination vector. Note that $\delta_i(\mathtt{X})$ does not depend on $h(\mathtt{X})$, so neither does $\delta_i(0)$. Hence, the *same* recombination vector $\mathbf{r}$ works for all $h(\mathtt{X})$. It is a public piece of information that all players can compute.

A final consequence is that for all secrets $s \in \mathbb{F}$ and all $C \subset \mathbb{F}$ with $|C| = t$ and $0 \notin C$, if we sample a uniformly random f of degree $\leq t$ and with $f(0) = s$ then the distribution of the t shares

$$(f(i))_{i \in C}$$

is the uniform distribution on $\mathbb{F}^t$. Since the uniform distribution on $\mathbb{F}^t$ clearly is independent of s, it in particular follows that given only t shares one gets no information on the secret.

One way to see that any t shares are uniformly distributed is as follows: One way to sample a polynomial for sharing of a secret s is to sample a uniformly random $a = (a_1, \ldots, a_t) \in \mathbb{F}^t$ and let $f_a(\mathtt{X}) = s + \sum_{j=1}^{t} a_j \mathtt{X}^t$ (as clearly $f_a(0) = s$.) For a fixed s and fixed C as above this defines an evaluation map from $\mathbb{F}^t$ to $\mathbb{F}^t$ by mapping $a = (a_1, \ldots, a_t)$ to $(f_a(i))_{i \in C}$. This map is invertible. Namely, given any $(y_i)_{i \in C} \in \mathbb{F}^t$, we know that we seek $f_a(\mathtt{X})$ with $f_a(i) = y_i$ for $i \in C$. We furthermore know that $f_a(0) = s$. So, we know $f_a(\mathtt{X})$ on $t+1$ points, which allows to compute $f_a(\mathtt{X})$ and $a \in \mathbb{F}^t$, using Lagrange interpolation. So, the evaluation map is invertible. Any invertible map from $\mathbb{F}^t$ to $\mathbb{F}^t$ maps the uniform distribution on $\mathbb{F}^t$ to the uniform distribution on $\mathbb{F}^t$.

Example Computations

We look at an example. Assume that we have five parties $\mathsf{P}_1, \ldots, \mathsf{P}_5$ and that we want to tolerate $t = 2$ corrupted parties. Assume that we work in $\mathbb{F} = \mathbb{Z}_{11}$ and want to share $s = 7$. We pick $a_1, a_2 \in_{\mathrm{R}} \mathbb{F}$ uniformly at random, say they become $a_1 = 4$ and $a_2 = 1$, and then we define

$$h(\mathtt{X}) = s + a_1 \mathtt{X} + a_2 \mathtt{X}^2 = 7 + 4\mathtt{X} + \mathtt{X}^2 \ . \tag{3.2}$$

Then we compute $s_1 = h(1) = 7 + 4 + 1 \bmod 11 = 1$, $s_2 = h(2) = 19 \bmod 11 = 8$, $s_3 = h(3) = 6$, $s_4 = h(4) = 6$, $s_5 = h(5) = 8$. So, the sharing is

$$[s] = (1, 8, 6, 6, 8) \ .$$

We send s_i securely to P_i.

Assume now that someone is given just the shares s_3, s_4, s_5. Since $3 > 2$, she can use Lagrange interpolation to compute the secret.

We first compute

$$\delta_3(\mathtt{X}) = \prod_{j=4,5} \frac{\mathtt{X}-j}{3-j} = \frac{(\mathtt{X}-4)(\mathtt{X}-5)}{(3-4)(3-5)} = (\mathtt{X}^2-9\mathtt{X}+20)((3-4)(3-5))^{-1} \pmod{11} .$$

We have that $(3-4)(3-5) = 2$ and $2 \cdot 6 \bmod 11 = 1$, so $((3-4)(3-5))^{-1} \bmod 11 = 6$. So,

$$\delta_3(\mathtt{X}) = (\mathtt{X}^2-9\mathtt{X}+20)6 = (\mathtt{X}^2+2\mathtt{X}+9)6 = 6\mathtt{X}^2+12\mathtt{X}+54 = 6\mathtt{X}^2+\mathtt{X}+10 \pmod{11} .$$

We check that

$$\delta_3(3) = 6 \cdot 3^2 + 3 + 10 = 67 = 1 \pmod{11} ,$$

$$\delta_3(4) = 6 \cdot 4^2 + 4 + 10 = 110 = 0 \pmod{11} ,$$

$$\delta_3(5) = 6 \cdot 5^2 + 5 + 10 = 165 = 0 \pmod{11} ,$$

as it should be.

We then compute

$$\begin{aligned}\delta_4(\mathtt{X}) = \prod_{j=3,5} \frac{\mathtt{X}-j}{4-j} &= \frac{(\mathtt{X}-3)(\mathtt{X}-5)}{(4-3)(4-5)} \\ &= (\mathtt{X}^2-8\mathtt{X}+15)(-1)^{-1} \\ &= (\mathtt{X}^2+3\mathtt{X}+4)10 \\ &= 10\mathtt{X}^2+8\mathtt{X}+7 .\end{aligned}$$

We can check that $\delta_4(3) = 121 = 0 \pmod{11}$, $\delta_4(4) = 199 = 1 \pmod{11}$, and $\delta_4(5) = 297 = 0 \pmod{11}$.

We then compute

$$\begin{aligned}\delta_5(\mathtt{X}) = \prod_{j=3,4} \frac{\mathtt{X}-j}{5-j} &= \frac{(\mathtt{X}-3)(\mathtt{X}-4)}{(5-3)(5-4)} \\ &= (\mathtt{X}^2-7\mathtt{X}+12)(2)^{-1} \\ &= (\mathtt{X}^2+4\mathtt{X}+1)6 \\ &= 6\mathtt{X}^2+2\mathtt{X}+6 .\end{aligned}$$

We can check that $\delta_5(3) = 66 = 0 \pmod{11}$, $\delta_5(4) = 110 = 0 \pmod{11}$, and $\delta_5(5) = 166 = 1 \pmod{11}$.

It is now clear that if for any s_3, s_4, s_5 we let

$$h(\mathtt{X}) = s_3 \cdot \delta_3(\mathtt{X}) + s_4 \cdot \delta_4(\mathtt{X}) + s_5 \cdot \delta_5(\mathtt{X}) ,$$

then $h(3) = s_3 \cdot 1 + s_4 \cdot 0 + s_5 \cdot 0 = s_3$, $h(4) = s_3 \cdot 0 + s_4 \cdot 1 + s_5 \cdot 0 = s_4$ and $h(5) = s_3 \cdot 0 + s_4 \cdot 0 + s_5 \cdot 1 = s_5$, which implies that if $s_3 = f(3)$, $s_4 = f(4)$

and $s_5 = f(5)$ for some quadratic polynomial, then $h(\mathtt{X}) = f(\mathtt{X})$. This allows to compute $h(\mathtt{X})$ from the three shares.
More concretely, notice that

$$\begin{aligned} h(\mathtt{X}) &= s_3\delta_3(\mathtt{X}) + s_4\delta_4(\mathtt{X}) + s_5\delta_5(\mathtt{X}) \\ &= (6s_3 + 10s_4 + 6s_5)\mathtt{X}^2 + (s_3 + 8s_4 + 2s_5)\mathtt{X} + (10s_3 + 7s_4 + 6s_5) \ . \end{aligned}$$

Since we consider $h(\mathtt{X})$ of the form $h(\mathtt{X}) = s + a_1\mathtt{X} + a_2\mathtt{X}^2$, we have that

$$\begin{aligned} s &= 10s_3 + 7s_4 + 6s_5 \bmod 11 \\ a_1 &= s_3 + 8s_4 + 2s_5 \bmod 11 \\ a_2 &= 6s_3 + 10s_4 + 6s_5 \bmod 11 \ , \end{aligned}$$

which is then the general formula for computing $h(\mathtt{X})$ from the three shares $s_3 = h(3), s_4 = h(4), s_5 = h(5)$.

In our concrete example we had the shares $s_3 = 6$, $s_4 = 6$ and $s_5 = 8$. If we plug this in we get

$$\begin{aligned} s &= 10 \cdot 6 + 7 \cdot 6 + 6 \cdot 8 \bmod 11 = 150 \bmod 11 = 7 \\ a_1 &= 6 + 8 \cdot 6 + 2 \cdot 8 \bmod 11 = 70 \bmod 11 = 4 \\ a_2 &= 6 \cdot 6 + 10 \cdot 6 + 6 \cdot 8 \bmod 11 = 144 \bmod 11 = 1 \ , \end{aligned}$$

which gives exactly the polynomial in (3.2).

If we had only been interested in finding the secret s and not the entire polynomial we would only need the equation $s = 10s_3 + 7s_4 + 6s_5 \bmod 11$. We see that $r = (10, 7, 6)$ is the recombination vector for finding $h(0)$ from $h(3), h(4), h(5)$ when $h(\mathtt{X})$ is a polynomial of degree at most 2.

3.3 A Passively Secure Protocol

3.3.1 Arithmetic Circuits

We will present a protocol which can securely evaluate a function with inputs and outputs in a finite field $\mathbb{F}$. For notational convenience we construct the protocol for the case where each party has exactly one input and one output from $\mathbb{F}$. I.e., $f : \mathbb{F}^n \to \mathbb{F}^n, (x_1, \ldots, x_n) \to (y_1, \ldots, y_n)$. We assume that the mapping $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ is described using an arithmetic circuit.

More formally, such a circuit is an acyclic directed graph, where each node is called a gate, the edges are called wires. Each gate has at most 2 in-coming wires. There are n input gates with no in-coming and 1 out-coming wire, each such gate is labeled by i for the player P_i who is going to supply the secret input value x_i for that gate. Then there are a number of internal addition and multiplication gates, with 2 input wires and any number of output wires, these add or multiply their two inputs. We also have multiply-by-constant gates, they have 1 input wire and any number of output wires, each such gate is labeled by a constant $\alpha \in \mathbb{F}$ and does multiplication by α. Finally there is for each P_i exactly one output gate labeled by i, with 1 input wire and no output wires. The value eventually assigned to the input wire of this gate is going to be y_i.

Evaluating a circuit can be done as follows: we assume that the input values have been specified initially, and we think of this as assigning the value x_i to the wire(s) coming out of the input gate labeled i. We assume that the gates have been numbered in some (arbitrary) way. We then take the first gate for which values have been assigned to all its input wires, compute the output value and assign it to the output wire(s) of the gate. Repeat this until all wires have had values assigned. This then also defines the output values.

The order in which we visit the gates in this procedure will be called the computational ordering in the following.

Considering arithmetic circuits is without loss of generality: It is well known that any function that is feasible to compute at all can be specified as a polynomial-size Boolean circuit using *and* and *negation*. But any such circuit can be simulated by operations in $\mathbb{F}$: Boolean values `true` or `false` can be encoded as 1 resp. 0. Then the *negation* of bit b is $1 - b$, and the *and* of bits b and b' is $b \cdot b'$.

Exercise 3.1 Assume that you are given a protocol that securely computes any function $f : \mathbb{F}^n \to \mathbb{F}^n$ given by an arithmetic circuit. Show how it can be used to securely compute any function $g : \{0,1\}^n \to \{0,1\}^n$, where each party has a bit as input and g can be any poly-time computable function. Assume that you have a poly-sized Boolean circuit for g. We argued how to do that above, but the solution only works if players select inputs correctly. If parties may not do this, there is the problem when the Boolean circuit is coded as an arithmetic circuit: it is important that all parties input 0 or 1. If players may input any $x_i \in \mathbb{F}$ we may be in trouble: Consider a case with three parties, each with an input $x_i \in \{0,1\}$. Assume that $y_1 = (1 - x_1)x_2 + x_1x_3$. If $x_1 = 0$, then $y_1 = x_2$, and if $x_1 = 1$, then $y_1 = x_3$. I.e., P_1 can choose to learn either x_2 or x_3, but not both.

1. Argue that a cheating P_i which inputs $x_1 \notin \{0,1\}$ can learn both x_2 and x_3.
2. Give a general construction which prevents this type of attack. [Hint: Assume that $\mathbb{F}$ is small and try to map all possible inputs $x_i \in \mathbb{F}$ to an input $x'_i \in \{0,1\}$ and then do the actual computation on $(x'_1, \ldots, x'_n)$.]

An assumption about the structure of circuits

In order to simply the proofs of security in this book, we will assume throughout that a circuit that is to be computed securely contains a multiplication gate immediately before each output gate, and furthermore this multiplication gate has only one output wire. We can assume this without loss of generality – if the circuit does not satisfy the condition for output y_j we can, for instance, introduce an extra input x'_j from P_j and then change to a new circuit that multiplies y_j by x'_j just before the output gate. This will make the circuit at most a constant factor larger and by choosing $x'_j = 1$, P_j will still learn the same value.

We emphasize that the protocols we present are secure even without this assumption, but proofs become much more cumbersome. We will have more to say about this later, when we get to the concrete proofs.

3.3.2 The Protocol

We recall that we will give a protocol for the scenario where there are secure channels between all parties. We assume that some subset C of the players, of size at most t, will go together after the protocol and try to learn as much as they can from the data they have seen. We will say in the following that the players in C are corrupt, while players not in C are honest. Since the function we are to compute is specified as an arithmetic circuit over $\mathbb{F}$, our task is, loosely speaking, to compute a number of additions and multiplications in $\mathbb{F}$ of the input values (or intermediate results), while revealing nothing except for the final result(s).

We begin by defining some convenient notation

Definition 3.1 *We define $[a; f]_t$, where $a \in \mathbb{F}$ and f is a polynomial over f with $f(0) = a$ and degree at most t:*

$$[a; f]_t = (f(1), ..., f(n)) \ ,$$

i.e., the set of shares in secret a, computed using polynomial f. Depending on the context, we sometimes drop the degree or the polynomial from the notation, so we write $[a; f]$ or just $[a]$.

Notice that on one hand the notation $[a; f]$ describes an object, namely the set of shares $(f(1), \ldots, f(n))$. On the other hand, it is also a statement. *It states that $f(0) = a$. The notation $[a; f]_t$ describes the* same *object, $(f(1), \ldots, f(n))$, but further more states that* $\deg(f) \leq t$.

Using the standard notation for entrywise addition, multiplication by a scalar and the Schur product we have for $\alpha \in \mathbb{F}$:

$$[a; f]_t + [b; g]_t = (f(1) + g(1), ..., f(n) + g(n)) \ , \tag{3.3}$$

$$\alpha[a; f]_t = (\alpha f(1), ..., \alpha f(n)) \ , \tag{3.4}$$

$$[a; f]_t * [b; g]_t = (f(1)g(1), ..., f(n)g(n)) \ . \tag{3.5}$$

By the trivial observations that $f(i) + g(i) = (f + g)(i)$, $f(i)g(i) = (fg)(i)$, the following very important facts are easy to see:

Lemma 3.2 *The following holds for any $a, b, \alpha \in \mathbb{F}$ and any polynomials f, g over $\mathbb{F}$ of degree at most t with $f(0) = a$ and $g(0) = b$:*

$$[a; f]_t + [b; g]_t = [a + b; f + g]_t \ , \tag{3.6}$$

$$\alpha[a; f]_t = [\alpha a; \alpha f]_t \ , \tag{3.7}$$

$$[a; f]_t * [b, g]_t = [ab; fg]_{2t} \ . \tag{3.8}$$

Since our notation for secret sharing both describe objects and make statements, the equations in the lemma actually say more than might be obvious on a first reading. As an example, the equation $[a; f]_t * [b; g]_t = [ab; fg]_{2t}$ states that the object $(f(1), \ldots, f(n)) * (g(1), \ldots, g(n))$ is identical to $((fg)(1), \ldots, (fg)(n))$. It also states that $(fg)(0) = ab$ and $\deg(fg) \leq 2t$.

The importance of the lemma is that it shows that by computing only on shares, we can – implicitly – compute on the corresponding secrets. For instance, the first of the above three relations shows that, if secrets a, b have been shared, then if each player adds his shares of a and b, then we obtain shares in the secret $a + b$. It will be useful in the following to have language for this sort of thing, so we define:

DEFINITION 3.3 *When we say that a player P_i* distributes *$[a; f_a]_t$, this means that he chooses a random polynomial $f_a(\mathtt{X})$ of degree $\leq t$ with $f_a(0) = a$, and then sends the share $f_a(j)$ to P_j, for $j = 1, \ldots, n$. Whenever players have obtained shares of a value a, based on polynomial f_a, we say that* the players hold *$[a; f_a]_t$.*

Suppose the players hold $[a; f_a]_t$, $[b; f_b]_t$. Then, when we say that the players compute *$[a; f_a]_t + [b; f_b]_t$, this means that each player P_i computes $f_a(i) + f_b(i)$, and by Lemma 3.2, this means that the players now hold $[a + b; f_a + f_b]_t$. In a similar way we define what it means for the players to compute $[a; f]_t * [b, g]_t$ and $\alpha[a; f]_t$.*

Using the tools and terms defined so far, we can describe a protocol for securely evaluating an arithmetic circuit, see Protocol CEPS (CIRCUIT EVALUATION WITH PASSIVE SECURITY).

3.3.3 Analysis

We will now analyze the security of the CEPS protocol. We will require two conditions:

Perfect Correctness With probability 1, all players receive outputs that are correct based on the inputs supplied.

Perfect Privacy Any subset C of corrupt players, of size at most $t < n/2$, learns no information beyond $\{x_j, y_j\}_{\mathsf{P}_j \in C}$ from executing the protocol, regardless of their computing power.

To be more precise about privacy, we will use the simulation paradigm that we also alluded to in the introduction: to show that someone learns only information X, we show that *everything* he sees can be *efficiently* recreated, or simulated, given only X.

To express what this exactly means in our case, we define view_j to be all the values that P_j sees during the execution of the protocol. More precisely, view_j is a vector, where we first add x_j, and then each time P_j makes a random choice we add it to view_j, and each time P_j receives a message, we add the message to view_j. Notice that view_j is a random variable, as it depends on the random choices made by P_j and the other parties (via their messages to P_j).

Privacy now means that there exists an efficient probabilistic algorithm – a simulator S – which, given $\{x_j, y_j\}_{\mathsf{P}_j \in C}$, can produce an output whose distribution is exactly the same as the entire set of data seen by C during the protocol, i.e., we want that

$$\mathsf{S}(\{x_j, y_j\}_{\mathsf{P}_j \in C}) \overset{\text{perf}}{\equiv} \{\text{view}_j\}_{\mathsf{P}_j \in C} \ .$$

Protocol CEPS (Circuit Evaluation with Passive Security)

The protocol proceeds in three phases: the input sharing, computation and output reconstruction phases.

Input Sharing: Each player P_i holding input $x_i \in \mathbb{F}$ distributes $[x_i; f_{x_i}]_t$.

We then go through the circuit and process the gates one by one in the computational order defined above. Just after the input sharing phase, we consider all input gates as being processed. We will maintain the following:

Invariant: Recall that computing with the circuit on inputs $x_1, ..., x_n$ assigns a unique value to every wire. Consider an input or an output wire for any gate, and let $a \in \mathbb{F}$ be the value assigned to this wire. Then, if the gate has been processed, the players hold $[a; f_a]_t$ for a polynomial f_a.

We then continue with the last two phases of the protocol:

Computation Phase: Repeat the following until all gates have been processed (then go to the next phase): Consider the first gate in the computational order that has not been processed yet. According to the type of gate, do one of the following

Addition gate: The players hold $[a; f_a]_t, [b; f_b]_t$ for the two inputs a, b to the gate. The players compute $[a; f_a]_t + [b; f_b]_t = [a + b; f_a + f_b]_t$.

Multiply-by-constant gate: The players hold $[a; f_a]_t$ for the inputs a to the gate. The players compute $\alpha[a; f_a]_t = [\alpha a; \alpha f_a]_t$.

Multiplication gate: The players hold $[a; f_a]_t, [b; f_b]_t$ for the two inputs a, b to the gate.

1. The players compute $[a; f_a]_t * [b; f_b]_t = [ab; f_a f_b]_{2t}$.
2. Define $h \stackrel{\text{def}}{=} f_a f_b$. Then $h(0) = f_a(0) f_b(0) = ab$ and the parties hold $[ab; h]_{2t}$, i.e., P_i holds $h(i)$. Each P_i distributes $[h(i); f_i]_t$.
3. Note that $\deg(h) = 2t \le n - 1$. Let $\mathbf{r}$ be the recombination vector defined in section 3.2, that is, the vector $\mathbf{r} = (r_1, \ldots, r_n)$ for which it holds that $h(0) = \sum_{i=1}^{n} r_i h(i)$ for any polynomial h of degree $\le n - 1$. The players compute

$$\begin{aligned}\sum_i r_i [h(i); f_i]_t &= [\sum_i r_i h(i); \sum_i r_i f_i]_t \\ &= [h(0); \sum_i r_i f_i]_t = [ab; \sum_i r_i f_i]_t\ .\end{aligned}$$

Output Reconstruction: At this point all gates, including the output gates have been processed. So do the following for each output gate (labeled i): The players hold $[y; f_y]_t$ where y is the value assigned to the output gate. Each P_j securely sends $f_y(j)$ to P_i, who uses Lagrange interpolation to compute $y = f_y(0)$ from $f_y(1), \ldots, f_y(t + 1)$, or any other $t + 1$ points.

The requirement that the distributions be exactly equal is the reason we talk about *perfect* privacy here.

We warn the reader already now that the general security definition we give in the next chapter looks quite different, although it too is based on the simulation paradigm. This is necessary, as correctness and privacy are simply not sufficient to define security if corrupt players may deviate from the protocol.

Correctness

If the invariant defined in the CEPS protocol is maintained, it is clear that once all gates are processed, then in particular for every wire going into an output gate, players hold $[y; f_y]_t$ where y is the correct value for that wire. So the player who is to receive that value will indeed get y.

To check that the invariant holds, note that this is trivially true for the input gates after the first phase. In the computation phase we easily see that the protocol for each type of gate maintains the invariant, by Lemma 3.2, and (for multiplication) by definition of the recombination vector.

Privacy

We now argue that the values seen by t corrupted parties gives them no information extra to their inputs and outputs.

Note that the corrupted parties only receive two types of messages from the honest parties.

Type 1: In Input Sharing and Multiplication gate, they receive shares in the random sharings $[x_i; f_{x_i}]_t$ respectively $[h(i); f_i]_t$ made by honest parties.

Type 2: In Output Reconstruction they receive all shares in $[y; f_y]_t$ for each output y which is for a corrupted party.

The privacy of the protocol now follows from the following two observations:

Observation 1: All values sent by honest parties to corrupted parties in Input Sharing and Multiplication are shared using random polynomials, f_{x_i} respectively f_i, of degree at most t. Since there are at most t corrupted parties, the set of corrupted parties receive at most t shares of these polynomials, and t shares of random polynomials of degree at most t are just uniformly random values.

Observation 2: The values sent by honest parties to corrupted parties in Output Reconstruction to reconstruct $[y; f_y]_t$ could have been computed by the corrupted parties themselves given the values they know prior to Output Reconstruction plus the output y. The reason is that prior to Output Reconstruction the corrupted parties already known $f_y(i)$ for each of the t corrupted parties P_i. This gives them t point on $f_y(\mathtt{X})$. If we give them the result y, they additionally know that $f_y(0) = y$. So, with their view prior to Output Reconstruction and the result y, they know $t + 1$ points on $f_y(\mathtt{X})$. Since $f_y(\mathtt{X})$ has degree at most t, this allows the corrupted parties to efficiently compute $f_y(\mathtt{X})$ using Lagrange interpolation. From $f_y(\mathtt{X})$ they can compute $f_y(i)$ for all honest parties P_i, and $f_y(i)$ is exactly the share in y sent by P_i in Output Reconstruction.

Let us first see why these two observations intuitively should make us consider the protocol secure. Recall that the goal is that the corrupted parties learn nothing extra to their own inputs and outputs. But clearly they don't! In Input Sharing and Multiplication they just receive uniformly random values. This gives them nothing, as they could have sampled such values themselves. In Output Reconstruction they then receive values which they could have computed themselves from their outputs y.

The observant reader will notice that this intuition is in fact also close to being a formal proof. Recall that our formal requirement to call a protocol perfectly private is that there exists a simulator S which given the inputs and outputs of the corrupted parties outputs a value with the same distribution as the view of the corrupted parties in the protocol. From the above intuition it is clear how S would proceed:

1. It runs the corrupted parties on their own inputs according to the protocol. It can do so, as it is given the inputs of the corrupted parties.
2. During Input Sharing and Multiplication it simulates the shares sent to the corrupted parties from the honest parties by simply sampling uniformly random values, which produces the right distribution by Observation 1.
3. In Output Reconstruction it will for each output y to a corrupted party compute the honest parties' shares in $[y; f_y]$ to be those consistent with the t simulated shares of the corrupted parties and $f_y(0) = y$. It can do so, as y is the output of a corrupted party and hence was given as input to S. This gives the right distribution by Observation 2.

We will now give the proof in more detail. It follows the above proof very closely. In the rest of the book, when we prove protocols secure, we will give proofs at the level of detail above. The purpose of the detailed proof below is to enable the reader interested in rigorous proofs to flesh out such proofs along the lines used below.

Formalizing Observation 1

We start by formalizing Observation 1. For this purpose, we define a function $\text{Strip}(\text{view}_j)$ which takes as input the view of a party $\mathsf{P}_j \in C$ and removes from view_j the $n - t$ shares $f_{y_j}(i)$ sent to P_j in Output Reconstruction from parties $\mathsf{P}_i \notin C$. The function outputs this stripped view. Notice that $\text{Strip}(\text{view}_j)$ still contains x_j, the random choices made by P_j, the shares sent to P_j in Input Sharing and Multiplication, and the shares sent to P_j from *corrupted* parties in Output Reconstruction.

We can now formalize Observation 1 as follows.

LEMMA 3.4 *Let $C \subset \{P_1, \ldots, P_n\}$ be a set of corrupted parties, and assume for simplicity that $|C| = t$. Consider two global input vectors $\mathbf{x}^{(0)} = (x_1^{(0)}, \ldots, x_n^{(0)})$ and $\mathbf{x}^{(1)} = (x_1^{(1)}, \ldots, x_n^{(1)})$, where $x_j^{(0)} = x_j^{(1)}$ for $\mathsf{P}_j \in C$, i.e., the corrupted parties have the same inputs in both vectors. For $b = 0, 1$, let view_j^b be the view of P_j after running Protocol* CEPS (CIRCUIT EVALUATION WITH PASSIVE SECURITY) *on*

$\mathbf{x}^b$. *Then it holds that*

$$\{\text{Strip}(\text{view}_j^{(0)})\}_{P_j \in C} \overset{\text{perf}}{\equiv} \{\text{Strip}(\text{view}_j^{(1)})\}_{P_j \in C} \ . \tag{3.9}$$

Proof For each part of the protocol we check that C sees the same distribution in case (0) and in case (1):

Input Sharing: Each $P_i \in C$ distributes $[x_i^{(b)}]_t$. Since $x_i^{(0)} = x_i^{(1)}$, this clearly leads to the same distribution on the shares in the two cases.

Each $P_j \notin C$ distributes $[x_i^{(b)}]_t$. It might be the case that $x_j^{(0)} \neq x_j^{(1)}$, but parties in C see at most t shares, and these are uniformly random and independent of $x_j^{(b)}$; In particular, the distribution is the same when $b = 0$ and $b = 1$.

Addition: Here no party sends or receives anything, so there is nothing to show.

Multiplication: Follows as for Input Sharing: We can assume that all values held by parties in C before the current multiplication have the same distribution in case (0) and in case (1). This is trivially the case for the first multiplication and can be assumed by induction for the following ones. In particular, the values they are supposed to locally multiply have the same distribution. Therefore all values generated and sent by parties in C have the same distribution. The honest parties only send shares of random sharings, and C only sees t shares of each sharing. These are just uniformly random values.

Output Reconstruction: Here all shares in $[y_j^{(b)}; f_{y_j^{(b)}}]$ are securely sent to P_j, for $j = 1, \ldots, n$. If P_j is honest, parties in C see nothing. If $P_j \in C$ is corrupt, then parties in C see *all* shares in $[y_j^{(b)}; f_{y_j^{(b)}}]$. But Strip exactly deletes the shares sent from honest parties to corrupted parties. The shares sent from corrupted parties to corrupted parties are computed from their view prior to Output Reconstruction, which we, by induction, can assume are identically distributed when $b = 0$ and $b = 1$.

□

Formalizing Observation 2

We then formalize Observation 2. For this purpose we need another function, Dress.

For any value $\mathbf{y}_C = \{y_j\}_{\mathsf{P}_j \in C} \in \mathbb{F}^t$, we define a function $\text{Dress}_{\mathbf{y}_C}$ which takes an input $\{\text{view}'_j\}_{\mathsf{P}_j \in C}$, where each view'_j is a stripped view of P_j, i.e., $\text{view}'_j = \text{Strip}(\text{view}_j)$, where view_j is a possible view of P_j. The output is of the form $\{\text{view}''_j\}_{\mathsf{P}_j \in C}$, where each view''_j is of the form of a full view of P_j. The purpose of $\text{Dress}_{\mathbf{y}_C}$ is to try to fill in those values that Strip removed. It attempts this as follows: For each $\mathsf{P}_j \in C$, it takes the t shares that the corrupted parties hold in the output of P_j, then it assumes that the output of P_j is the value y_j given in $\mathbf{y}_C$, computes which shares the honest parties would hold in y_j in that case, and extends view'_j with these shares, as if they were sent to P_j. In more detail, first $\text{Dress}_{\mathbf{y}_C}$ inspects each view'_j, $\mathsf{P}_j \in C$, and reads off the shares of the output of P_j that was sent to P_j from corrupted parties $\mathsf{P}_i \in C$ — recall that we did not delete these shares, only the shares sent by honest parties. So, for $\mathsf{P}_j \in C$, it now knows the shares of the t corrupted parties in the output of P_j — call these $f_{y_j}(i)$ for $\mathsf{P}_i \in C$. Then it takes the value y_j from $\mathbf{y}_C$ and defines $f_{y_j}(0) \overset{\text{def}}{=} y_j$. After this, $\text{Dress}_{\mathbf{y}_C}$ has defined $f_{y_j}(\mathtt{X})$ in $t+1$ points and can hence compute the unique degree-t polynomial $f(\mathtt{X})$ consistent with these points. Then for $\mathsf{P}_i \notin C$ it adds the share $f_{y_j}(i)$ to view_j as if P_i sent $f_{y_j}(i)$ to P_j. Call the result view''_j, and let $\text{Dress}_{\mathbf{y}_C}(\{\text{view}'_j\}_{\mathsf{P}_j \in C}) = \{\text{view}''_j\}_{\mathsf{P}_j \in C}$.

We can now formalize Observation 2 as follows.

LEMMA 3.5 *Let $C \subset \{P_1, \ldots, P_n\}$ be a set of corrupted parties, and assume for simplicity that $|C| = t$. Let $\mathbf{x}$ be any input vector, let $(y_1, \ldots, y_n) = f(\mathbf{x})$, let $\mathbf{y}_C = \{y_j\}_{P_j \in C}$, and let* view_j *be the view of P_j after running Protocol* CEPS (CIRCUIT EVALUATION WITH PASSIVE SECURITY) *on $\mathbf{x}$. Then*

$$\text{Dress}_{\mathbf{y}_C}(\{\text{Strip}(\text{view}_j)\}_{P_j \in C}) = \{\text{view}_j\}_{P_j \in C} \ .$$

PROOF The view view_j contains n shares in the sharing $[y'_j, f_{y'_j}]_t$ of the output y'_j that P_j computes in the protocol. By perfect correctness of Protocol CEPS (CIRCUIT EVALUATION WITH PASSIVE SECURITY), we know that $y'_j = y_j$ for the y_j given by $(y_1, \ldots, y_n) = f(\mathbf{x})$. So, view_j contains n shares in a sharing $[y_j; f_{y_j}]_t$ of y_j. Then Strip removes the shares of the honest parties. The function $\text{Dress}_{\mathbf{y}_C}$, however, recomputes the same shares, as it computes them from the same correct y_j (the y_j in $\mathbf{y}_C$ was taken from $(y_1, \ldots, y_n)$) and the t shares of the corrupted parties, which were not deleted from view_j. By Lagrange interpolation this gives exactly the shares back that Strip deleted. See Exercise 3.2 for the case where less than t parties are corrupted. □

Formalizing the Simulator

We can also formalize the simulator S via the two functions defined above. It proceeds as follows:

1. The input to S is $\{x_j, y_j\}_{j \in C}$.
2. It defines a global input vector $\mathbf{x}^{(0)} = (x_1^{(0)}, \ldots, x_n^{(0)})$, where $x_j^{(0)} = x_j$ for $\mathsf{P}_j \in C$ and $x_j^{(0)} = 0$ for $\mathsf{P}_j \notin C$. Here x_j for $\mathsf{P}_j \in C$ are the values it received as input.
3. It runs Protocol CEPS (CIRCUIT EVALUATION WITH PASSIVE SECURITY) on $\mathbf{x}^{(0)}$ and lets $\text{view}_j^{(0)}$ be the view of P_j in this execution of the protocol.
4. It lets $\mathbf{y}_C = \{y_j\}_{\mathsf{P}_j \in C}$, where y_j is the value it received as input.
5. It outputs $\text{Dress}_{\mathbf{y}_C}(\{\text{Strip}(\text{view}_j^{(0)})\}_{\mathsf{P}_j \in C})$.

Notice that running the protocol on wrong inputs for the honest parties results in sending t shares of wrong values during Input Sharing and Computation Phase. Since the corrupted parties only receive t shares, this is the same as just sending them uniformly random values. Then Strip and $\text{Dress}_{\mathbf{y}_C}$ removes the shares sent by the honest parties in Output Reconstruction and replaces them with shares consistent with the shares of the corrupted parties and the correct output. So, the above way to specify the simulator is equivalent to the one given earlier.

Formally Analyzing the Simulator

We now prove that the simulator produces the right distribution.

THEOREM 3.6 *Let* $\mathbf{x} = (x_1, \ldots, x_n)$ *be a global input vector and let* $(y_1, \ldots, y_n) = f(\mathbf{x})$ *and let* $C \subset \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ *with* $|C| = t$*. For* $\mathsf{P}_j \in C$*, let* view_j *be the view of* P_j *when running Protocol* CEPS (CIRCUIT EVALUATION WITH PASSIVE SECURITY) *on the input* $\mathbf{x}$*. Then*

$$\mathsf{S}(\{x_j, y_j\}_{\mathsf{P}_j \in C}) \overset{\text{perf}}{\equiv} \{\text{view}_j\}_{\mathsf{P}_j \in C} \ .$$

PROOF Let $\mathbf{y}_C = \{y_j\}_{\mathsf{P}_j \in C}$ for the values y_j defined in the premise of the theorem. Define $\mathbf{x}^{(0)} = (x_1^{(0)}, \ldots, x_n^{(0)})$ where $x_j^{(0)} = x_j$ for the x_j given in the premise of the theorem for $\mathsf{P}_j \in C$ and $x_j^{(0)} = 0$ for $\mathsf{P}_j \notin C$. Let $\text{view}_j^{(0)}$ be the view of P_j generated by S by running Protocol CEPS (CIRCUIT EVALUATION WITH PASSIVE SECURITY) on the input $\mathbf{x}^{(0)}$.

By construction of the simulator we have that

$$\mathsf{S}(\{x_j, y_j\}_{\mathsf{P}_j \in C}) \overset{\text{perf}}{\equiv} \text{Dress}_{\mathbf{y}_C}(\{\text{Strip}(\text{view}_j^{(0)})\}_{\mathsf{P}_j \in C}) \ . \tag{3.10}$$

Since S generates $\text{view}_j^{(0)}$ by running Protocol CEPS (CIRCUIT EVALUATION WITH PASSIVE SECURITY) on the input $\mathbf{x}^{(0)}$, and because $x_j = x_j^{(0)}$ for $\mathsf{P}_j \in C$ and $|C| = t$, it follows from Lemma 3.4 that

$$\{\text{Strip}(\text{view}_j^{(0)})\}_{\mathsf{P}_j \in C} \overset{\text{perf}}{\equiv} \{\text{Strip}(\text{view}_j)\}_{\mathsf{P}_j \in C} \ . \tag{3.11}$$

We then apply Proposition 2.8, Property 3 to Eq. 3.11. This gives us that

$$A(\{\text{Strip}(\text{view}_j^{(0)})\}_{\mathsf{P}_j \in C}) \overset{\text{perf}}{\equiv} A(\{\text{Strip}(\text{view}_j)\}_{\mathsf{P}_j \in C}) \tag{3.12}$$

for all functions A. Let $A = \text{Dress}_{\mathbf{y}_C}$. Plugging this A into Eq. 3.12 we get that

$$\text{Dress}_{\mathbf{y}_C}(\{\text{Strip}(\text{view}_j^{(0)})\}_{\mathsf{P}_j \in C}) \overset{\text{perf}}{\equiv} \text{Dress}_{\mathbf{y}_C}(\{\text{Strip}(\text{view}_j)\}_{\mathsf{P}_j \in C}) \ . \tag{3.13}$$

By Lemma 3.5 we have that

$$\text{Dress}_{\mathbf{y}_C}(\{\text{Strip}(\text{view}_j)\}_{\mathsf{P}_j \in C}) \overset{\text{perf}}{\equiv} \{\text{view}_j\}_{\mathsf{P}_j \in C} \ . \tag{3.14}$$

Using Eq. 3.10, Eq. 3.13 and Eq. 3.14 and transitivity of $\overset{\text{perf}}{\equiv}$ (Proposition 2.8, Property 2) we get that

$$\mathsf{S}(\{x_j, y_j\}_{\mathsf{P}_j \in C}) \overset{\text{perf}}{\equiv} \{\text{view}_j\}_{\mathsf{P}_j \in C} \ ,$$

as desired. □

To formally prove that Protocol CEPS (Circuit Evaluation with Passive Security) is perfectly private we have to prove Theorem 3.6 for all C with $|C| \leq t$, not just $|C| = t$. There is a technical issue here that we need to take care of: suppose the circuit computes $[a; f_a]_t$, $[b; f_b]_t$ and $[a+b, f_a+f_b]_t$ as intermediate values, and the corrupt players have output gates that let them learn a, b and $a + b$. If we would just open these directly, the corrupt players would see three polynomials where the sum of the first two equals the third, and so the simulator would have to ensure that this also happens in the simulation. This would require special care in the case where $|C| < t$, since there the output polynomials are not determined from the shares of corrupt players and the outputs, the simulator would have to make some random choices that would ensure the right relations between polynomials.

However, recall that we have assumed that a multiplication always happens immediately before we open an output. While doing the multiplication, players generate fresh random polynomials and the sharing of the multiplication result is done using a linear combination of these polynomials. As a result, we can assume that whenever we open an output, the polynomial that is revealed is uniformly random with the only constraint that it determines the correct output value. This allows the simulator to handle each output gate independently. Using this observation, the reader is invited to prove the general case:

Exercise 3.2 Give a rigorous proof for Theorem 3.6 for the case $|C| < t$. [Hint: Dress is going to be a probabilistic algorithm.]

3.3.4 Example Computations and Proofs by Example

The above argument for the output reconstruction shows that it does not harm to give all shares of an output to the corrupted parties. This, in particular, shows that the shares do not carry information about how the result was computed: If $c = a + b$ is reconstructed and the result is 6, then the n shares of c will be consistent with both $a = 2, b = 4$ and $a = 1, b = 5$ — otherwise the protocol could not be secure. We will, however, look at two exercises to exemplify this phenomenon.

Consider a setting where variables a and b have been computed, and where then a variable $c = a + b$ is computed and output to P_1. Assume that $n = 3$ and $t = 1$. I.e., we have three parties $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$ and one can be corrupted. For sake of example, say it is P_1. Since $t = 1$ we are using polynomials of the form $f(\mathtt{X}) = \alpha_0 + \alpha_1\mathtt{X}$, a.k.a. lines.

Assume that $a = 2$ and that a is shared using the polynomial $a(\mathtt{X}) = 2 + 2X$, and assume that $b = 4$ and that b is shared using the polynomial $a(\mathtt{X}) = 4 + X$. This gives the following computation:

Variable	Value	P_1	P_2	P_3
a	2	**4**	6	8
b	4	**5**	6	7
$c = a + b$	6	**9**	**12**	**15**

We show the shares $a(1) = 4$, $a(2) = 6$, $a(3) = 8$ and the shares $b(1) = 5$, $b(2) = 6$, $b(3) = 7$ in the rows to the right of the variables and their values. When the parties compute the variable $c = a + b$, they simply add locally and compute the shares 9, 12 respectively 15. In the table all shares that P_1 would see in this case are put in bold.

We want that P_1 only learns that $c = 6$, and nothing about a and b except that $a + b = 6$. We demonstrate that this is the case by an example. We let the party P_1 make the hypothesis that $a = 1$ and $b = 5$. Hopefully it cannot exclude this hypothesis. For starters, P_1 has the following view (not knowing the shares of P_1 and P_2):

Variable	Value	P_1	P_2	P_3
a	1	**4**	?	?
b	5	**5**	?	?
$c = a + b$	6	**9**	**12**	**15**

If $a(0) = 1$ and $a(1) = 4$, then it must be the case that $a(\mathtt{X}) = 1 + 3\mathtt{X}$, which would imply that $a(2) = 7$ and $a(3) = 10$. Furthermore, if $b(0) = 5$ and $b(1) = 5$, then it must be the case that $b(\mathtt{X}) = 5 + 0\mathtt{X}$, which would imply that $b(2) = 5$ and $b(3) = 5$. If P_1 fills these values into the table it concludes that the network must be configured as follows for its hypothesis to hold:

Variable	Value	P_1	P_2	P_3
a	1	**4**	7	10
b	5	**5**	5	5
$c = a + b$	6	**9**	**12**	**15**

Note that this hypothesis is consistent with the protocol and what P_1 have seen, as $7 + 5 = 12$ and $10 + 5 = 15$. Therefore $a = 1$ and $b = 5$ is as possible as $a = 2$ and $b = 4$.

EXERCISE 3.3 In the above example, P_1 could also have made the hypothesis that $a = 0$ and $b = 6$. Show that P_1 cannot exclude this example, by filling in the below table and noting that it is consistent.

Variable	Value	P_1	P_2	P_3
a	0	**4**	?	?
b	6	**5**	?	?
$c = a + b$	6	**9**	**12**	**15**

We now consider an example of a multiplication of variables $a = 2$ and $b = 3$. The polynomials used to share them are $a(\mathtt{X}) = 2 + \mathtt{X}$ and $b(\mathtt{X}) = 3 - \mathtt{X}$:

Variable	Value	P_1	P_2	P_3
a	2	**3**	4	5
b	3	**2**	1	0
$d = ab$	6	**6**	4	0
d_1	**6**	**4**	**2**	**0**
d_2	4	**6**	8	10
d_3	0	**0**	0	0
$c = 3d_1 - 3d_2 + d_3$	6	**−6**	**−18**	**−30**

We explain the lower part of the table soon, but first note that the shares of $d = ab = 6$ are not on a line, as all the other shares are. The reason is that $d(\mathtt{X}) = a(\mathtt{X})b(\mathtt{X})$ is not a line, but a quadratic polynomial. In fact, $d(\mathtt{X}) = (2 + \mathtt{X})(3 - \mathtt{X}) = 6 + \mathtt{X} - \mathtt{X}^2$, which is consistent with $d(1) = 6$, $d(2) = 4$ and $d(3) = 0$.

After having computed the local products d_i, the next step in the multiplication algorithm uses the Lagrange formula for computing d from d_1, d_2, d_3, so we derive that one. Since $2t = 2$ we are looking at quadratic polynomials $y(\mathtt{X}) = \alpha_0 + \alpha_1\mathtt{X} + \alpha_2\mathtt{X}^2$, where α_0 is the secret. Therefore the shares are $y_1 = y(1) = \alpha_0 + \alpha_1 + \alpha_2$, $y_2 = y(2) = \alpha_0 + 2\alpha_1 + 4\alpha_2$ and $y_3 = y(3) = \alpha_0 + 3\alpha_1 + 9\alpha_2$. It follows that α_0 can always be computed from the shares as $\alpha_0 = 3y_1 - 3y_2 + y_3$. This formula was found using simple Gaussian elimination, but is also given by the Lagrange interpolation formula. I.e., in our case the recombination vector is $r = (3, -3, 1)$.

In our example we have $d_1 = 6$, $d_2 = 4$ and $d_3 = 0$, and indeed $3d_1 - 3d_2 + d_3 = 18 - 12 = 6 = ab$, as it should be. Each party now shares its value d_i. In the table P_1 used the polynomial $d_1(\mathtt{X}) = 6 - 2\mathtt{X}$, P_2 used the polynomial $d_2(\mathtt{X}) = 4 + 2\mathtt{X}$ and P_3 used the polynomial $d_3(\mathtt{X}) = 0 + 0\mathtt{X}$. The parties then locally combine their shares by an inner product with the recombination vector $(3, -3, 1)$, leading to the shares in the table: as an example, P_1 computed $-6 = 3 \cdot 4 + (-3) \cdot 6 + 1 \cdot 0$.

Again, an example will reveal that any other hypothesis, like $a = 1$ and $b = 6$ would have given the exact same view to P_1. The reader is encourage to do that, by solving the following exercise.

EXERCISE 3.4 Show that the values seen by P_1 are consistent with the hypothesis $a = 1$ and $b = 6$ by filling in the following table and noting that it is consistent.

Variable	Value	P_1	P_2	P_3
a	1	**3**	?	?
b	6	**2**	?	?
$d = ab$	6	**6**	?	?
d_1	**6**	**4**	**2**	**0**
d_2	?	**6**	?	?
d_3	?	**0**	?	?
$c = 3d_1 - 3d_2 + d_3$	6	$\mathbf{-6}$	$\mathbf{-18}$	$\mathbf{-30}$

[To check solution: It must say -42 and 84 somewhere.]

3.4 Optimality of the Corruption Bound

What happens if the number t of corrupt players is larger than what we assumed so far, i.e., what if $t \geq n/2$? Then the CEPS protocol no longer works, the multiplication subprotocol breaks down. You may want to take a moment to see why this is the case. We will now show that this is no coincidence. In fact there are functions that cannot be computed securely if $t \geq n/2$.

Towards a contradiction, suppose there is a protocol π, with *perfect privacy* and *perfect correctness* (as defined earlier) for two players $\mathsf{P}_1, \mathsf{P}_2$ to securely evaluate the logical AND of their respective private input bits b_1, b_2, i.e., $b_1 \wedge b_2$. The set C of corrupt players from the definition may in this case be $C = \{\mathsf{P}_1\}$ or $C = \{\mathsf{P}_2\}$, i.e., $n = 2, t = 1$.

We assume as usual that the players communicate using a perfect *error-free communication channel.*

Without loss of generality, we may assume the protocol is of the following form.

1. Each player P_i has a private input bit b_i. Before the protocol starts, they select private random strings $r_i \in \{0,1\}^*$ of appropriate length.

   Their actions in the forthcoming protocol are now uniquely determined by these initial choices.
2. P_1 sends the first message m_{11}, followed by P_2's message m_{21}.

   This continues until P_2 has sent sufficient information for P_1 to compute $y = b_1 \wedge b_2$. Finally, P_1 sends y (and some halting symbol) to P_2.

   The transcript of an s-round conversation is

$$\mathcal{T} = (m_{11}, m_{21}, \ldots, m_{1s}, m_{2s}, y).$$

   For $i = 1, 2$, the view of P_i is

$$\text{view}_i = (b_i, r_i, \mathcal{T}) .$$

To be concrete about the assumptions, perfect correctness means here that the protocols always halts (in a number of rounds t that may perhaps depend on the inputs and the random coins) and that the correct result is always computed.

Perfect privacy means that the distribution of P_i's view of the protocol depends only on his input bit b_i and the result $r = b_1 \wedge b_2$.

Note that these conditions imply that if one of the players has input bit equal to 1, then he learns the other player's input bit with certainty, whereas if his input bit equals 0, he should have no information at all about the other player's input bit.

Let $\mathcal{T}(0,0)$ denote the set of transcripts $\mathcal{T}$, which can arise when $b_1 = 0$ and $b_2 = 0$ and let $\mathcal{T}(0,1)$ denote the set of transcripts $\mathcal{T}$, which can arise when $b_1 = 0$ and $b_2 = 1$.

Given a transcript $\mathcal{T} = (m_{11}, m_{21}, \ldots, m_{1s}, m_{2s}, y)$ we say that it is *consistent with input* $b_1 = 1$ if there exists r_1 such that running P_1 with input $b_1 = 1$ and randomness r_1 might result in $\mathcal{T}$ being generated. Namely, if we assume that P_1 receives the message m_{2i} in round i for $i = 1, \ldots, s$, then with $b_1 = 1$ and randomness r_1, P_1 would send exactly the message m_{1i+1} in round $i+1$. Let $\mathcal{C}_{b_1=1}$ denote the transcripts consistent with $b_1 = 1$.

It follows from perfect privacy that

$$\mathcal{T}(0,0) \subset \mathcal{C}_{b_1=1} \ .$$

Assume namely that P_1 has input $b_1 = 0$ and P_2 has input $b_2 = 0$, and that P_2 sees a transcript $\mathcal{T}$ which is not from $\mathcal{C}_{b_1=1}$. Then P_2 can conclude that $b_1 = 0$, contradicting privacy.

From the perfect correctness we can conclude that

$$\mathcal{C}_{b_1=1} \cap \mathcal{T}(0,1) = \emptyset \ .$$

Consider namely $\mathcal{T} \in \mathcal{T}(0,1)$. By perfect correctness it follows that $y = 0$ in $\mathcal{T}$. Therefore $\mathcal{T}$ is clearly not consistent with input $b_1 = 1$, as that would mean that $\mathcal{T}$ could be produced with $b_1 = 1$ and $b_2 = 1$, which would give result $y = 1$ (again by perfect correctness).

From these two observations we see that $\mathcal{T}(0,0) \cap \mathcal{T}(0,1) = \emptyset$.

Now consider a case where P_1 executes the protocol with $b_1 = 0$. Then he sees a transcript $\mathcal{T}$ from either $\mathcal{T}(0,0)$ or $\mathcal{T}(0,1)$. Since they are disjoint, P_1 can determine the input of P_2 simply by checking whether $\mathcal{T} \in \mathcal{T}(0,0)$ or $\mathcal{T} \in \mathcal{T}(0,1)$. This contradicts perfect privacy.

One concrete algorithm P_1 could use to check whether $\mathcal{T} \in \mathcal{T}(0,0)$ or $\mathcal{T} \in \mathcal{T}(0,1)$ is to do the equivalent test of whether $\mathcal{T} \in \mathcal{C}_{b_1=1}$. This can be done by a brute force search of all possibilities for his own randomness. Such an algorithm is of course not efficient, but this is a not a problem as we consider corrupt players that may have unlimited computing power.

The argument can be generalized to show impossibility of 2-party protocols where privacy or correctness may fail with small but negligible probability, or even with small constant probability.

To show a similar impossibility result for $n > 2$ players, assume that there exist an n-player protocol π where each P_i has an input bit b_i, and the protocol computes $b_1 \wedge \cdots \wedge b_n$ as output for everyone, with perfect security against $t \geq n/2$ corrupted players. Such a protocol would imply the existence of a protocol for two players A, B computing the AND, which we have just seen is impossible. We construct this protocol by letting A and B emulate π, such that A runs $n/2$ players in π "in his head" and B runs the others. A and B exchange message when a message in π is sent between the two sets of $n/2$ players. We can make this protocol compute the AND of two inputs bits from A and B, and security follows from security of π. The details are straightforward and are left to the reader.

Exercise 3.5 Show that there is no 2-party perfectly secure and perfectly correct protocol for the OR function $(b_1, b_2) \mapsto b_1 \vee b_2$.

Exercise 3.6 Show that the following protocol is a perfectly secure (in the sense of poly-time simulation) and perfectly correct protocol for the XOR function $(b_1, b_2) \mapsto b_1 \oplus b_2$. Party P_1 sends b_1 to P_2 and P_2 sends b_2 to P_1. Then they both output $b_1 \oplus b_2$.

Exercise 3.7 Any binary Boolean function $B : \{0,1\} \times \{0,1\} \to \{0,1\}$ can be given by a vector $(o_{00}, o_{01}, o_{10}, o_{11}) \in \{0,1\}^4$, by letting $B(b_1, b_2) = o_{b_1 b_2}$. The AND function is given by $(0,0,0,1)$, the OR function is given by $(0,1,1,1)$, the XOR function is given by $(0,1,1,0)$, and the NAND function is given by $(1,1,1,0)$. Show that all functions specified by a vector with an even number of 1's can be securely computed as defined above and that none of the other functions can.

Computational Security

The assumptions about the players' unbounded computational resources and the communication channel are essential for the impossibility results.

It can be shown that any of the following conditions is sufficient for the existence of a secure two-party protocol for the AND function (as well as OR).

1. Existence of trapdoor one-way permutations.
2. Both players are memory bounded.
3. The communication channel is noisy.

We sketch a secure AND protocol based on the assumption that there exists a public-key cryptosystem where the public keys can be sampled in two different ways: There is the usual key generation which gives an encryption key ek and the corresponding decryption key dk. The other method only generates ek and even the party having generated ek cannot decrypt ciphertexts under ek. We assume that these two key generators give encryption keys with the same distribution

If $b_1 = 0$, then P_1 samples ek uniformly at random without learning the decryption key. If $b_1 = 1$, then P_1 samples ek in such a way that it learns the decryption key dk. It sends ek to P_2. Then P_2 sends $C = E_{pk}(b_2)$ to P_1. If $b_1 = 0$, then P_1

outputs 0 and sends $y = 0$ to P_2 which outputs y. If $b_1 = 1$, then P_1 decrypts C to learn b_2, outputs b_2 and sends $y = b_2$ to P_2 which outputs y.

Since the two ways to sample the public key gives the same distribution, the protocol is perfectly secure for P_1. The security of P_2 depends on the encryption hiding b_2 when $b_1 = 0$ and is therefore computational. In particular, a computationally unbounded P_1 could just use brute force to decrypt C and learn b_2 even when $b_1 = 0$.

This protocol can be in principle be made secure against deviations by letting the parties use generic zero-knowledge proofs to show that they followed the protocol. This in principle leads to secure two-party protocols for any function.

Exercise 3.8

1. Use the special cryptosystem from above to give a secure protocol for the OR function.
2. Try to generalize to any two-party function, where one of the parties has a constant number of input bits and the other party might have an arbitrary number of inputs. [Hint: The party with a constant number of inputs will have perfect security and will not send just a single encryption key.]

4

Models

Contents

4.1 Introduction

The protocol we described in Chapter 3 is perfectly correct and perfectly private, but cannot tolerate that any of the parties deviate from the protocol. In later chapters we will present protocols which are correct and private even if some parties do not follow the protocol. Such protocols are called robust. Before we can prove that these protocols are robust, we need a good definition of what we mean by that.

In this section we will describe a security model for cryptographic protocols known as the UC model. Here *UC* stands for *universally composable*. The name was adopted because a protocol proven secure in this model remains secure regardless of the context in which it is used. In other words, it can be "universally" composed with any set of other protocols. Our formulation of the UC model differs in several respects from the way it was originally formalized in the literature. We will hint at these differences as we go, some additional comments can be found in the notes section of this chapter.

Before we look at the formal details, let us start by discussing the basic ideas of how to define privacy and robustness of a protocol and last, but not least, why and how these two concepts should be combined.

4.1.1 Defining Privacy

It is convenient to first look back on how we defined privacy in Chapter 3. Informally we defined a protocol to be private (against corruptions of size t) as follows. First pick an input $(x_1, \ldots, x_n)$ for the protocol and make a run of the protocol on this input. Then pick some $C \subset \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ with $|C| \le t$ and consider the values $\{\text{view}_j\}_{\mathsf{P}_j \in C}$, where view_j is the view of party P_j in the execution. The values $\{\text{view}_j\}_{\mathsf{P}_j \in C}$ constitute exactly the information leaked to the corrupted parties, C, during an execution. In the following we therefore call the values $\{\text{view}_j\}_{\mathsf{P}_j \in C}$ the leaked values.

We then want to say that the leaked values do not allow the corrupted parties to learn anything that they should not learn. It is clear that the corrupted parties necessarily must learn their own inputs and outputs, in fact this is the entire purpose of the protocol. We therefore call the values $\{x_j, y_j\}_{\mathsf{P}_j \in C}$ the allowed values.

In these new terms we can give the following informal definition of what it means for a protocol to be private.

A protocol is private if it always holds that the leaked values contain no more information than the allowed values.

This leaves us with the problem of defining what it means for one value V_1 to contain no more information than some other value V_2. One very convenient way to define this is to note that if V_1 can be computed from (only) V_2, then clearly V_1 cannot contain more information than V_2. However, this definition would be a bit too liberal for our use, in particular if we consider security which is guaranteed using cryptography. Suppose, for instance, that we make $V_2 = n$ public where $n = pq$ is an integer that is the product of two large primes p, q. And let us define $V_1 = (p, q)$. Now, since the prime factors of integers are unique, it is certainly *possible* to compute $V_1 = (p, q)$ from $V_2 = n$. Nevertheless, if factoring large integers is a hard problem (as we hope if we use the RSA public-key cryptosystem), we could make n public and expect that, in practice, p, q will remain secret.

From this point of view, one would claim that $V_2 = (p, q)$ contains more than $V_1 = n$, because given V_2 we can efficiently compute much more than if we were given only V_1. We therefore say that a value V_1 contains no more knowledge than V_2 if V_1 can be computed *efficiently* from V_2.

Following this line of thought, we get the following more developed definition of privacy:

A protocol is private if it always holds that the leaked values can be computed efficiently from the allowed values.

To prove that a protocol is private one therefore has to show that there exists an efficient algorithm which takes the allowed values as input and outputs the leaked values. This program, which demonstrates that it is possible to efficiently compute the leaked values from the allowed values, is what we called the simulator. By our latest attempt at a definition, the simulator would be required to efficiently compute the leaked values from the allowed values. Note, however, that the traces view_i are generated by the parties in the protocol, and they are allowed to make random choices. Therefore $\{\text{view}_j\}_{\mathsf{P}_j \in C}$ is not a fixed value, but actually a random variable. Therefore the simulator will also have to output a random variable, i.e., use internal random choices. This is why the actual definition of privacy says that

There exists an efficient simulator S such that the simulated values $\mathsf{S}(\{x_j, y_j\}_{P_j \in C})$ and the leaked values $\{\text{view}_j\}_{P_j \in C}$ have the same distribution.

4.1.2 Defining Robustness

We now sketch how we define robustness. Robustness has to do with how an attack influences the protocol and achieves some effect, in that the outputs may be different. In other words, an attack on robustness tries to achieve influence rather than knowledge. For the definition, we will therefore play exactly the same game as we did with privacy, just using influence as the basic currency instead of knowledge.

When we consider robustness, we assume that the corrupted parties are what we call Byzantine. This just means that we make absolutely no assumptions on how they behave. Concretely, we assume that an adversary has taken control over the corrupted parties, and every time the protocol instructs them to send some message, which is supposed to be computed in a particular way according to the protocol, the adversary may instruct them send any other message. Our job, as protocol designer, is then to design the protocol to work correctly regardless of how the adversary behaves.

Before discussing influence in more general term it is instructive to see two examples of what constitutes influence.

Example 4.1 Consider Protocol CEPS (Circuit Evaluation with Passive Security), and let us run it with three parties and $t = 1$. Let us compute the function $f(x_1, x_2, x_3) = (x_1x_2x_3, x_2, x_3)$. That is, P_1 is to learn $x_1x_2x_3$ and the other parties are only to learn their own input. Let us now assume that P_1 is corrupted and show an example of how P_1 can influence the protocol. Suppose P_1 replaces its input x_1 by $x_1' = 1$, i.e., in the Input sharing phase it does a secret sharing of 1. Then it follows the rest of the protocol honestly. As a result P_1 will learn $y_1 = x_1'x_2x_3 = 1x_2x_3 = x_2x_3$. △

The above example shows that Protocol CEPS (Circuit Evaluation with Passive Security) allows P_1 to influence the protocol in such a way that he always learns x_2x_3. The reader should take a moment to contemplate whether we should consider this a security problem? and if so, why?

Example 4.2 Consider again Protocol CEPS (Circuit Evaluation with Passive Security) with three parties, $t = 1$ and $f(x_1, x_2, x_3) = (x_1x_2x_3, x_2, x_3)$. Say we order that gates in the corresponding circuit such that we first compute $[u]_t = [x_1x_2]_t$ and then $[y_1]_t = [ux_3]_y$. Finally, in the output phase, players send all their shares in $[y_1]_t$ to P_1.[1] Let us again assume that P_1 is corrupt. Now, however, we assume that P_1 follows the protocol during the Input sharing phase using the input $x_1' = 0$. Then during the Multiplication protocol where $[u]_t = [x_1'x_2]_t$ is computed, players locally compute products that define a polynomial h with $h(0) = x_1'x_2$. However, P_1 does not distribute $[h(1)]_t$ as he is supposed to, but instead he distributes $[r_1^{-1} + h(1)]$, where we recall that r_1 is the first entry in the reconstruction vector. This means that the result becomes

$$\begin{aligned}
[u]_t &= r_1[r_1^{-1} + h(1)]_t + \sum_{i=2}^{n} r_i[h(i)]_t \\
&= [r_1r_1^{-1} + r_1h(1) + \sum_{i=2}^{n} r_ih(i)]_t \\
&= [1 + \sum_{i=1}^{n} r_ih(i)]_t \\
&= [1 + x_1'x_2]_t \\
&= [1]_t \ ,
\end{aligned}$$

where the last equality follows because $x_1' = 0$. Then P_1 follows the protocol honestly in the multiplication protocol for $[y_1]_t = [ux_3]_t$ and in the Reconstruction phase. As a result P_1 will learn $y_1 = ux_3 = 1x_3 = x_3$. △

The above example shows that Protocol CEPS (Circuit Evaluation with Passive Security) allows P_1 to influence the protocol in such a way that P_1 always learns x_3. Should we consider this a security problem? and if so, why?

[1] Note that the other parties can just output their input, though the generic protocol would secret share their inputs and the reconstruct them towards the parties.

It seems natural to say that a protocol is robust if the adversary cannot gain anything from influencing the protocol. Note, however, that if the adversary is able to take over a party P_i, then there is some influence which is inevitable, namely input substitution. If, e.g., the adversary is controlling the party P_1, then it can of course force P_1 to use some value x_1' as the input to the protocol instead of x_1. As a result, the protocol would compute $f(x_1', x_2, \ldots, x_n)$ instead of $f(x_1, x_2, \ldots, x_n)$. There is no way to combat this, as it is identical to the situation where an honest P_1 is running with input x_1', which is of course a perfectly allowable situation. We therefore have a certain allowed influence which we must accept. In the secure function evaluation protocol, the only allowed influence is input substitution.

In general we decide on what the allowed influence is by specifying an ideally secure way to solve the problem at hand. Then we say that any influence which is possible in this ideal world is allowed influence. To specify an ideal world for secure function evaluation, we could imagine a perfectly trusted hardware box which allows each P_i to securely input x_i to the box without the other parties learning anything about x_i. Then the box computes $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and outputs y_i to P_i in such a way that the other parties learn nothing about y_i. After this the box blows up such that its internal state is lost forever. Except for the explosion, it is hard to imagine a more secure way to do function evaluation. And, it is easy to see that the only power a set of corrupted parties have in this ideal world is to pool their inputs and then compute some alternative inputs to the box, i.e., they can only do input substitution.

The influence that the attacker has on the actual protocol execution is called the actual influence. For a secure function evaluation protocol that runs on a network with secure channels, the actual influence of the adversary is that he can send arbitrary values to the honest parties on behalf of one or more of the corrupted parties.

We then say that a protocol is robust if no matter how the adversary uses his actual influence, the effect will always correspond to an effect of an allowed influence. This would, namely, show that whatever effect the actual influence has, the attacker could have obtained the same using an allowed influence, so the effect is by definition allowed. This leads to the following first attempt at defining robustness.

A protocol is robust if it always holds that the effect of an actual influence can also be obtained using an allowed influence.

To make this more precise, recall that when we considered privacy we said that a protocol was private if the actual values leaked by the protocol could be efficiently simulated given the allowed values. We will reuse this approach. So, we will require that for every adversary attacking the protocol one can efficiently compute an allowed way to influence the protocol which has the same effect. The efficient algorithm computing this allowed influence is called a simulator.

A protocol is robust if there exists an efficient simulator S, such that for every *adversary attacking the protocol, S can efficiently compute an allowed influence with the same effect.*

For the secure function evaluation protocol, this definition just means that for each attack on the protocol resulting in outputs $(y'_1, \ldots, y'_n)$, one should be able to efficiently compute an input substitution for the corrupted parties which makes the ideal box for function evaluation give the same outputs $(y'_1, \ldots, y'_n)$.

Let us then return to the above two examples of possible attacks. In the first example P_1 influences the protocol as to always learn x_2x_3. This is not a problem, as P_1 can obtain this by simply using $x_1 = 1$ as input, which is an allowed influence. So, this is not an attack on the robustness. In the second example P_1 influences the protocol as to always learn x_3. This *is* an attack on the robustness, as there is no input substitution which allows P_1 to always learn x_3. To see this, suppose we are in a case where P_2 has input 0. This means, of course, that no matter how the other inputs are chosen, the result will be 0. In particular, no matter how P_1 chooses his input, he will learn nothing about x_3. However, in the actual attack, he learns x_3 with certainty.

4.1.3 Combining Privacy and Robustness

The last attack we just saw looks at first like an attack on the correctness only: P_1 makes the result be different from what it should be. However, it actually turns out to be an attack on the privacy as well: the incorrectness makes the protocol leak the input of P_3 in cases where it should be perfectly hidden from P_1. The fact that correctness and privacy can be entangled in this way makes it impossible to consider them in isolation. They are really two sides of the same thing.

What this concretely means for our full formal definition is that we must require existence of *one single simulator* that simultaneously demonstrates both privacy and robustness.

What this means is that the simulator receives information on how the adversary tries to influence the real protocol, and it must translate this efficiently into an allowed influence. In the "opposite" direction, it receives the allowed values and must efficiently simulate the leaked values. If this can be done, it essentially shows that anything that an adversary could obtain in an attack could also be obtained using an attack that is by definition harmless.

4.1.4 A Sketch of UC Security

In this subsection, we will sketch how our intuitive ideas on privacy and correctness materialize when we move towards a formal definition by considering players as computational entities. In Section 4.2 we then give the full formal model and definition of security.

The UC model was originally phrased in terms of interactive Turing machines.

We will take a slightly more abstract approach and model protocols using so-called interactive systems, which in turn consists of a number of so-called interactive agents, which can communicate by sending messages on so-called ports. Interactive systems are defined in Chapter 2. Here we recall briefly the main terminology.

The ports of an agent are named and are divided into inports and outports. This is a simple mechanism for describing how the agents are connected. If some agent A_1 outputs a message m on an outport named P and some agent A_2 has an inport also named P, then m will be input to A_2 on the port P.

An interactive agent A can be thought of as a computer which has some open connections, or ports, on which it can receive and send messages. The main technical difference is that an agent only changes state when it is explicitly activated, and that the activation happens on an inport. Let AP be the name of an inport of A. An activation starts by inputting AP to A to tell it which port it was activated on — the port AP is called the activation port. Now A might read some of the messages waiting on its inports, possibly change state and possibly sends messages on some of its outports. This is called an activation of the agent. At the end of an activation, the agent also outputs the name RP of one of its outports — the port RP is called the return port. In a larger system, the return port specifies the next activation port. We think of this as some activation token @ being passed around between the agents.

An interactive system $\mathcal{IS}$ is just a set of agents, where no two agents have inports with the same name and no two agents have outports with the same name — this is to ensure that it is uniquely given how the ports should be connected.

If some agent A has an outport named P and some agent B has an inport named P, then we say that A and B are connected by P. Technically, when the interactive system is executed, a message queue will be associated to P. When A sends a message on P it is entered to the end of the queue. When B reads from P it pops the front element of the queue or receives EOQ if the queue is empty.

If some agent A in an interactive system $\mathcal{IS}$ has a inport named P and no agent in $\mathcal{IS}$ has an outport named P, then we call P an open inport of the system $\mathcal{IS}$. In the same way, if some agent A in $\mathcal{IS}$ has an outport named P and no agent in $\mathcal{IS}$ has an inport named P, then we call P an open outport of the system $\mathcal{IS}$.

An interactive system $\mathcal{IS}$ can receive messages from its environment on its open inports — these are just entered into the message queues for the open inports. An interactive system with open inports can also be activated. This happens by specifying an open inport AP of the system. The system is activated as follows: first the agent who has AP as inport is activated with activation port AP. As a result of this it reads messages on some of its inports, changes state and sends messages on some of its outports. It also specifies a return port RP. The agent with an inport named RP is then activated next, with the activation port begin RP, and so on. When at some point an agent specifies a return port RP which is an open outport of the system (such that there is no agent in the system with an inport named RP), the activation of the system stops. We say that $\mathcal{IS}$ returned with return port RP. As a result of this the system might have read some messages on its open inports, might have changed state, might have output messages on some of its open outports, and will have specified a return port RP. Hence an interactive system is much like an interactive agent — it just has more internal structure.

Behavioral Equivalence

Seen from outside the system (not having access to the internal structure of $\mathcal{IS}$) the only observable events in an activation of $\mathcal{IS}$ is that some values were input on some open inports and that later some values appeared on some open outports. We are often not interested in the internal structure of a system (as it will correspond to how a protocol is implemented) but only this externally observable input/output behavior (as it will correspond to the inputs and outputs of the protocol, the allowed influence on the protocol and the leakage from the protocol). We therefore have a notion of two systems being indistinguishable if they give the same outputs on the same open outports whenever they get the same inputs on the same open inports. Because two indistinguishable systems need not have the same internal structure, as long as they behave in the same way, we sometimes call them behaviorally equivalent.

Security by Behavioral Equivalence.

In the UC model, the security of a protocol is based on the notion of behaviorally equivalent systems.

The first step in the formalization is to model the protocol using an interactive system π. The system π will contain an agent P_i for each party in the protocol and some agent $\mathcal{R}$ modeling the communication resource that the parties have access to, like an agent securely moving messages between the parties. The party P_i will have an inport $\texttt{in}_i$ on which it receives inputs for the protocol and an outport $\texttt{out}_i$ on which it delivers outputs from the protocol. It also has ports connecting it to the resource $\mathcal{R}$. The resource $\mathcal{R}$ takes care of moving messages between parties. In addition to moving messages, $\mathcal{R}$ also models the leakage of the communication network and the possible influence that an attacker might have on the communication network.

Example 4.3 If we want $\mathcal{R}$ to model authenticated communication without order preservation of the messages, it could have an outport R.leak on which it would output all messages sent via $\mathcal{R}$, to model that authenticated channels not necessarily hide the messages sent, and it could have an inport R.infl on which it takes instruction about which order to deliver the messages in, to model that it does not preserve the ordering of messages. This is exactly the allowed leakage and the allowed influence on an authenticated channel without order preservation. △

To specify how a protocol is *supposed* to work, a potentially much simpler system F is formulated. The system F is formulated such that it always has the intended input/output behavior, such that it only leaks the allowed values, and such that it only allows the allowed influence. We sometimes call F the intended functionality and sometimes we call F the ideal functionality. The ideal functionality F is often without internal structure (just a single agent) as its only job is to have the correct input/output behavior, such that we can compare the protocol π to F.

EXAMPLE 4.4 For the secure function evaluation problem, F could simply take one input x_i from each party, on some inport $\texttt{in}_i$ designated for that party, then it internally computes $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and outputs each y_i on an outport $\texttt{out}_i$ designated for the party who is to learn y_i. Its only leakage would be that it reveals *when* a party provides input. The only influence would be that it allows the attacker to specify when the outputs are to be delivered.[2] To model this leakage and the influence, we give F an outport `F.leak` and an inport `F.infl`. When it receives x_i on $\texttt{in}_i$, it outputs "`party` i `gave input`" on `F.leak`. After computing $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, if it receives an input "`deliver to party` i" on `F.infl`, it outputs y_i on $\texttt{out}_i$, unless it already gave output to party number i. △

Notice that if there existed such a thing as F in the real world and party number i was magically and locally connected to $\texttt{in}_i$ and $\texttt{out}_i$ on F, then secure function evaluation would be easy: each party inputs x_i to F and then at some point gets back y_i. The adversary would learn when and if F was used and determine when it delivers messages, but the parties would always be guaranteed that the results are correct and that their inputs are kept secret by F. It does not get better than this, which is why we call F the *ideal* functionality.

A protocol π is then said to be secure if the system π behaves "in the same way" as the ideal functionality F. This means that the security of a protocol π is always relative to another system F. We can therefore not speak about a protocol π being secure. We can only say that π is at least as secure as F. This is natural, as a statement of the form π *is secure* is actually absurd; It claims that the protocol π is secure without even stating what problem the protocol π is supposed to solve. The statement π *is at least as secure as F* can however be made precise. The job of F in this statement is then to specify the task that π is supposed to solve, by having the required input/output behavior, by allowing only the allowed influence and leaking only the allowed leakage. When π is as secure as the intended functionality F, then we also say that π securely implements F.

The Simulator

When comparing a protocol π and an ideal functionality F, we are left with the following problem. The ideal functionality F has a leakage port `F.leak` and an influence port `F.infl`, and it typically allows very limited leakage and influence. The

[2] Creating protocols where it is not visible when a party gives input and where an attacker with some control over the communication network cannot influence when the parties are ready to give outputs is hard. Hence we can turn this into allowed leakage and influence by adding it to F, to make F easier to realize.

protocol π will consist of n parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, and these will be connected using an agent $\mathcal{R}$ modeling the communication network used by the protocol. All these agents will have their own leakage ports and influence ports, say $\texttt{R.leak}, \texttt{R.infl}$ for $\mathcal{R}$.

Hence we cannot expect π and F to be behaviorally equivalent, the sets of available ports are completely different. To make matters worse, the ports of the players and of $\mathcal{R}$ will typically allow much more leakage and influence than does F.

We therefore introduce the simulator whose job it is to fix this problem. The simulator is another interactive agent $\mathcal{S}$ meant to be connected to F. It connects to the leakage port $\texttt{F.leak}$ of F and the influence port $\texttt{F.infl}$ of F, i.e., it sees the leakage of F and gets to influence F. At the same time it has all the leakage and influence ports of the players in π and of $\mathcal{R}$. So, if we connect $\mathcal{S}$ and F, then $\mathsf{F} \diamond \mathcal{S}$ has the same set of open ports π. Now it makes sense to ask for π and $\mathsf{F} \diamond \mathcal{S}$ to be behaviorally equivalent.

Here we used the notion $\mathcal{IS}_0 \diamond \mathcal{IS}_1$ for composing interactive systems. This notation is defined in detail i Chapter 2, here we just remind the reader that the two composed systems must have matching port names in order for the composition to be well defined.

Notice that whether π and $\mathsf{F} \diamond \mathcal{S}$ are behaviorally equivalent depends heavily on the simulator $\mathcal{S}$. The job of $\mathcal{S}$ is to make the systems look the same to any distinguisher. I.e., it translates commands on the influence ports corresponding to the protocol into commands on the influence port of the ideal functionality, and it sees the values output on the leakage port of the ideal functionality and must then output values on the leakage ports corresponding to the protocol. It must do so in such a way that $\mathsf{F} \diamond \mathcal{S}$ behaves like π. In other words, the simulator simulates the leakage of the protocol using the the leakage from the ideal functionality and it simulates influence on the protocol using its influence on the ideal functionality. This is exactly according to our informal definitions of privacy and robustness from above. Note that we have not yet talked about how we model corruption of players, we will return to this shortly when we give the complete definition.

Universal Composition

A primary reason for the popularity of the UC model is its so-called universal composition Theorem (UC Theorem). The UC Theorem says that if π is a secure protocol for some task (specified by an intended functionality F), then it is safe to use the protocol π as a sub-protocol in any context where one needs to solve the task at hand. More precisely, if some protocol is secure when it uses F as an auxiliary resource, then that protocol is also secure if F is replaced by the protocol π. This allows to analyze a complex protocol $\pi_{\texttt{CMPLX}}$ in an easier way by abstracting away some sub-protocol π: we replace calls to π by calls to an ideal functionality F with the indented input/output behavior of π. We then prove two claims which are reduced in complexity, namely that 1) $\pi_{\texttt{CMPLX}}$ is secure when it uses the ideal sub system F as resource and we prove that 2) the protocol π

securely implements F. From this we get for free, using the UC Theorem, that $\pi_{\texttt{CMPLX}}$ is also secure when it uses π as sub protocol instead of calling F.

4.2 The UC Model

We now proceed to give the formal details of our version of the UC framework.

4.2.1 Clock-Driven Execution

One minor deviation we do from the way the UC framework was originally formulated is that we use a clock-driven execution, where we introduce a very abstract notion of local clocks in most entities. The original formulation of the UC framework was more message driven. The reason for choosing clock-driven activation is that we want our framework to handle both synchronous protocols and asynchronous protocols. We can do this by either letting the local clocks be synchronized or by allowing them to drift apart. The original formulation of the UC framework was targeted against asynchronous protocols only, and hence does not capture synchronous protocols in a convenient way.

By a clocked entity in our framework we mean an interactive agent with some extra properties as specified below. Recall that in Chapter 2, we defined interactive systems that are composed of interactive agents. When executing such a system, an agent passes control to another by handing it an activation token. An interactive system of clocked entities will simply be a special case, where the token is passed according to certain rules that we will specify in a moment.

All clocked entities in our framework are going to have one or more inports with a name ending in `infl` or $\texttt{infl}_i$. This could, e.g., be an inport named `N.infl` or $\texttt{N.infl}_i$. If a clocked entity has an inport named `N.infl`, then it must have a matching outport named `N.leak`, called a leakage port. And vice versa, if a clocked entity has an outport named `N.infl`, then it must have a matching inport named `N.leak`. Among many other things these ports will be used to drive the execution of the interactive systems we consider.

A clocked entity receives a clocking signal (we say it is clocked) if it receives the activation token on an inport with a name of form `N.infl` (or $\texttt{N.infl}_i$). When it is clocked it must eventually return the activation on the matching outport `N.leak` (respectively $\texttt{N.leak}_i$). We introduce this rule, and other rules for passing the activation around, for technical reasons, which we discuss later. Before returning the activation token on `N.leak` or $\texttt{N.leak}_i$, a clocked entity C is allowed to do recursive calls to other clocked entities. Concretely, if C has an outport with a name of form `R.infl` or $\texttt{R.infl}_i$, it is allowed to send the activation token on that port. When it later receives the activation token back on `R.leak` respectively $\texttt{R.leak}_i$, it must either return the activation token on `N.leak` (respectively $\texttt{N.leak}_i$) or do another recursive call to a clocked entity. Summarizing, a clocked entity must obey the following list of rules for activation.

initialization A clocked entity holds as part of its state a bit active $\in \{0, 1\}$,

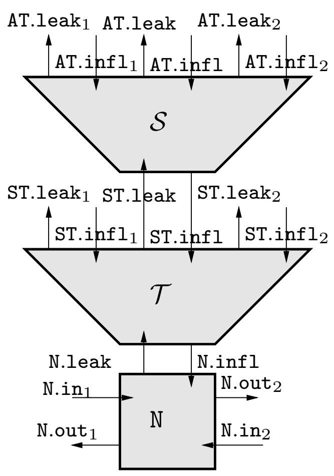


Figure 4.1 Diagram used for explaining the rules for activation.

which is initially set to 0. When active = 0 the clocked entity is said to be inactive. When active = 1 the clocked entity is said to be active. It also holds a bit calling $\in \{0, 1\}$, which is initially set to 0. When calling = 1 the clocked entity is said to be calling.

activation bounce during inactivity If an inactive clocked entity receives the activation token on any port with a name not of the form `N.infl` or `N.infl`$_i$, then it returns the activation token on the matching outport without doing anything else, i.e., it does not read messages, it does not change state and it does not send messages.

clocking If a inactive clocked entity receives the activation token on an open inport with a name of the form `N.infl` or `N.infl`$_i$, then it sets active $\leftarrow 1$ and stores the name `CP` of the inport on which it was activated. We say that it was called on `CP`.

return or clock An active clocked entity is only allowed to send the activation token on an outport matching the inport `CP` on which it was called or an open outport with a name of the form `R.infl` or `R.infl`$_i$.

return the call If an active clocked entity sends the activation token on the outport matching the inport `CP` on which it was called, then it sets active $\leftarrow 0$. In that case we say that it returned the call.

recursive call If an active clocked entity sends the activation token on an open outport named `R.infl` or `R.infl`$_i$, it first sets calling $\leftarrow 1$ and stores the name `RP` of the port on which it did the recursive call. We say that it did a recursive call on `RP`.

activation bounce during calls If a *calling* clocked entity receives the activation token any inport `P` which does not match the outport `RP` on which it did the recursive call, then it returns the activation token on the outport matching `P` without doing anything else.

result If a *calling* clocked entity receives the activation token on the inport matching the outport `RP` on which it did the recursive call, then it sets calling $\leftarrow 0$.

Together, these rules ensure that all clocked entities pass the activation token in a simple recursive manner over matching influence and leakage ports, and that they only do work when they are called on their influence ports or after having being returned a recursive call.

As an example, see Fig. 4.1. If we call $\mathcal{S}$ on $\texttt{AT.leak}_1$, then it must return the activation on $\texttt{AT.leak}_1$. But before it does so, it is allowed to make a number of calls to $\mathcal{T}$ on `ST.infl`.[3] Whenever $\mathcal{S}$ does a recursive call to $\mathcal{T}$, the clocked entity $\mathcal{T}$ must return the activation on `ST.leak`, but before it does so, it is allowed to make a number of calls to the agent named `N` on `N.infl`.[4] Whenever $\mathcal{T}$ does a recursive call to `N`, the clocked entity `N` must return the activation on `N.leak`. And, since `N` has no outport ending in `infl` or $\texttt{infl}_i$, it is not allowed to do recursive calls before returning the activation. The net result of this is that when $\mathcal{S} \diamond \mathcal{T} \diamond \texttt{N}$ is called on $\texttt{AT.infl}_1$, then it may send messages on some of its outports, and it eventually returns the activation on $\texttt{AT.leak}_1$.

As another example, assume that we call $\mathcal{T}$ on $\texttt{ST.infl}_1$. In that case $\mathcal{T}$ must eventually return the call on $\texttt{ST.leak}_1$. Before doing so, it is allowed to do recursive calls to `N`. It is, however, not allowed to do recursive calls to $\mathcal{S}$, as it is connected to $\mathcal{S}$ by an outport that is a leakage and not an influence port, so this would be against the rule *return or clock*.

We can extend the definition of a clocked entity to an interactive system $\mathcal{IS}$ composed of several clocked entities. A system $\mathcal{IS}$ composed of clocked entities is defined to be active if and only if at least one of its constituents is active. It is calling if one its constituents is calling and has called an agent that it not part of $\mathcal{IS}$. It is now trivial to check that if $\mathcal{IS}$ consists of clocked entities, then $\mathcal{IS}$ itself also obeys all the above rules for clocked entities. That is, we have

LEMMA 4.1 *A composition of clocked entities is a clocked entity.*

Another advantage of only having simple recursive calls is that we can define a notion of polynomial time which is maintained under composition.

DEFINITION 4.2 *We say that a clocked entity is recursive poly-time if all its internal computation is expected polynomial time (in the security parameter κ) and it does at most an expected polynomial (in κ) number of recursive calls before it returns it own call.*

[3] Note that $\mathcal{S}$ is *not* allowed to send the activation on any other outport, as it is only allowed to do recursive calls on ports ending in `infl` or $\texttt{infl}_i$.

[4] Note that $\mathcal{T}$ is *not* allowed to send the activation on any other outport.

Lemma 4.3 *A composition of recursive poly-time clocked entities is a recursive poly-time clocked entity.*

Proof Consider a system $\mathcal{IS}$ composed of recursive poly-time agents. It follows from Lemma 4.1 that $\mathcal{IS}$ is a clocked entity. It is also clear that $\mathcal{IS}$ is responsive, i.e., an activation is eventually returned. Now let us see that the total number of internal activations in any execution of $\mathcal{IS}$ is expected polynomial. This follows by induction on the number of agents. If $\mathcal{IS}$ has only 1 agent, the claim is trivial. So assume we have i agents and consider an activation of $\mathcal{IS}$ and say agent `A` is the first one activated. Note that by the clocking rules the activation of $\mathcal{IS}$ ends when `A` returns its call. The expected number of calls `A` makes is bounded by a polynomial $p(\kappa)$ by definition. By the clocking rules, each such call is actually an activation of a composed clocked entity with $i - 1$ agents. By induction hypothesis the expected number of internal calls in the smaller system is bounded by a polynomial $q(\kappa)$. This means that the total expected number of calls in $\mathcal{IS}$ is at most $p(\kappa)q(\kappa)$, which is polynomial. The fact that $\mathcal{IS}$ is responsive and has expected polynomial number of internal activations exactly means that $\mathcal{IS}$ is a poly-time interactive system in the sense we defined in Chapter 2. Hence, by Proposition 2.17, the expected running spent in any activation of $\mathcal{IS}$ is polynomial. $\square$

In the following, we describe various types of agents we need in our framework, namely players in protocols, ideal functionalities and simulators. These will all be clocked entities and are therefore assumed to follow the rules for clocked entities. A typical behavior for such an agent when it is activated is to empty all its message queues and then return the call on the appropriate leakage port. When we describe concrete examples, we will not always say explicitly that calls are returned this way, as this is required behavior for any clocked entity.

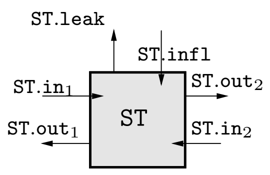
4.2.2 Ideal Functionalities

Formally an ideal functionality will be an interactive agent, as defined in Section 2.4. To name ports uniquely throughout the framework, each ideal functionality has a name `F`. The interface of F is as follows: F has n inports named $\texttt{F.in}_1, \ldots, \texttt{F.in}_n$ and n outports named $\texttt{F.out}_1, \ldots, \texttt{F.out}_n$. These $2n$ ports are called the protocol ports. In addition to the protocol ports, F has two special ports, an inport `F.infl` called the influence port and an outport `F.leak` called the leakage port.

Example 4.5 As an example, we specify an ideal functionality F_{ST} for secure transfer as in Agent F_{ST}. Only the port structure for two parties is shown. The code is general enough to handle any number of parties. Each time the ideal functionality is clocked, we apply the rules until all messages queues are empty and then return on `ST.leak`. The commands on the influence port is used to determine in which order the messages are delivered. The leakage of $(mid, i, j, |m|)$ specifies that also an implementation of F_{ST} is allowed to leak the message identifier and

Agent $\mathrm{F}_{\mathtt{ST}}$

The ideal functionality for secure transfer between n parties.



- On input (mid, j, m) on $\mathtt{ST.in}_i$ (where $j \in \{1, \ldots, n\}$), output $(mid, i, j, |m|)$ on $\mathtt{ST.leak}$ and store (mid, i, j, m). Ignore any later input of the form $(mid, j, \cdot)$ on $\mathtt{ST.in}_i$. Here mid is a message identifier used to distinguish different messages from i to j.
- On input $(\mathtt{deliver}, mid, i, j)$ on $\mathtt{ST.infl}$, where some (mid, i, j, m) is stored, delete (mid, i, j, m) and output (mid, i, m) on $\mathtt{ST.out}_j$.

the length of m. Leaking $|m|$ is important, as no cryptosystem can fully hide the size of the information being encrypted. Leaking mid allows implementations to do the same, which might make implementation easier. $\triangle$

Modeling Corruption

The special ports of an ideal functionality are also used to model which information is allowed to leak when a party is corrupted and which control an adversary gets over a party when that party is corrupted. There are many choices, but we will assume that all ideal functionalities have the following standard corruption behavior.

- On input $(\mathtt{passive\ corrupt}, i)$ on $\mathtt{F.infl}$, the ideal functionality F outputs $(\mathtt{state}, i, \sigma)$ on $\mathtt{F.leak}$, where σ is called the ideal internal state of party i and is defined to be all previous inputs on $\mathtt{F.in}_i$ plus those on the message queue of $\mathtt{F.in}_i$, along with all previous outputs on $\mathtt{F.out}_i$.[5]
- On input $(\mathtt{active\ corrupt}, i)$ on $\mathtt{F.infl}$, the ideal functionality F records that the party i is corrupted and outputs the ideal internal state of party i on $\mathtt{F.leak}$. Then it starts ignoring all inputs on $\mathtt{F.in}_i$ and stops giving outputs on $\mathtt{F.out}_i$. Instead, whenever it gets an input $(\mathtt{input}, i, x)$ on $\mathtt{F.infl}$, it behaves exactly as if x had arrived on $\mathtt{F.in}_i$, and whenever F is about to output some value y on $\mathtt{F.out}_i$, it instead outputs $(\mathtt{output}, i, y)$ on $\mathtt{F.leak}$.

[5] Formally we do not require that the ideal functionality empties its input queues and outputs all messages on the leakage tape, as this would not be poly-time behaviour, and in some settings we need that ideal functionalities are poly-time. We can solve this technical problem by instead saying that it outputs only the next message from $\mathtt{F.in}_i$. The adversary can then just input $(\mathtt{passive\ corrupt}, i)$ several times in a row to empty the input queue, if so desired. This is considered poly-time, as the ideal functionality responds in poly-time on each activation. Similar tricks are used later, but we stop mentioning them explicitly.

One way to think of a passive corruption is that if a party is corrupted, the adversary only learns the inputs and outputs of that party.[6]

One way to think about active corruption is that after a party has become actively corrupted, all its inputs are chosen by the adversary (via $\texttt{F.infl}$) and all its outputs are leaked to the adversary (via $\texttt{F.leak}$). Since it is impossible to protect against input substitution (even an otherwise honest party could do this) and it is inevitable that outputs intended for some party is seen by an adversary controlling that party, the standard corruption behavior models an ideal situation where an adversary gets only these inevitable powers.

4.2.3 Protocols

A simple protocol π consists of just n parties, $\mathsf{P}_1, \dots, \mathsf{P}_n$, where each P_i is an agent, i.e., the protocol is the interactive system $\pi = \{\mathsf{P}_1, \dots, \mathsf{P}_n\}$. A simple protocol has a protocol name $\texttt{F}$. For reasons that will become obvious, we let a protocol π and the ideal functionality F that π is supposed to implement have the same name. A simple protocol also has a resource name $\texttt{R}$. This is the name of the resource that π uses for communication. The agent P_i is called a simple party, and it has six ports. The port structure of P_i is derived from the names $\texttt{F}$ and $\texttt{R}$. It has an inport $\texttt{F.in}_i$ and an outport $\texttt{F.out}_i$, exactly as the ideal functionality F that π will be compared to later. Those two ports are called the protocol ports. In addition it has an outport named $\texttt{R.in}_i$ and an inport named $\texttt{R.out}_i$. Those two ports are called the resource ports. Finally P_i has an inport name $\texttt{R.infl}_i$ and an outport named $\texttt{R.leak}_i$. These are called the special ports, and are used to model corruption of P_i (and to clock P_i). This is detailed below, but first we discuss how the parties in a protocol communicate.

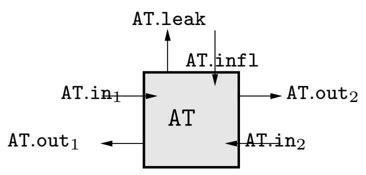
Let $\pi = \{\mathsf{P}_1, \dots, \mathsf{P}_n\}$ be a protocol with resource name $\texttt{R}$ and let $\mathcal{R}$ be an ideal functionality with name $\texttt{R}$. Then $\mathcal{R}$ has an inport named $\texttt{R.in}_i$ and P_i has an outport named $\texttt{R.in}_i$, and $\mathcal{R}$ has an outport named $\texttt{R.out}_i$ and P_i has an inport named $\texttt{R.out}_i$. This means that in the system $\pi \diamond \mathcal{R} = \{\mathsf{P}_1, \dots \mathsf{P}_n, \mathcal{R}\}$, the resource ports of the parties are connected to the protocol ports of $\mathcal{R}$. Hence the only open ports in $\pi \diamond \mathcal{R}$ are the protocol ports of π and the special ports of π and $\mathcal{R}$. We call $\pi \diamond \mathcal{R}$ a protocol using resource $\mathcal{R}$. Notice that this way ideal functionalities can play two roles, they can be specification of intended behavior, but they can also play the role as network resources, i.e., the means by which parties communicate. As we will see, this duality is central in formulating the UC Theorem.

EXAMPLE 4.6 We continue with the secure transfer example. We want to implement F_{ST} using an authenticated channel and a public-key encryption scheme. For starters, let us consider a sender P_1 and a receiver P_2. These will communicate

[6] A non-standard corruption behavior could be to only leak the last input or output. This would model an even more ideal situation, where an adversary cannot learn previous inputs and outputs when it breaks into a party. This is known as forward security and is a desirable property in some cases, but not a concern we will have in this book.

Agent $\mathsf{F}_{\mathtt{AT}}$

The ideal functionality for authenticated transfer between n parties.



- On receiving a value (mid, j, m) on $\mathtt{AT.in}_i$ (where $j \in \{1, \ldots, n\}$), output (mid, i, j, m) on $\mathtt{AT.leak}$ and store (mid, i, j, m). Ignore any later input of the form $(mid, j, \cdot)$ on $\mathtt{AT.in}_i$.
- On an input $(\mathtt{deliver}, mid, i, j)$ on $\mathtt{AT.infl}$, where some (mid, i, j, m) is stored, delete (mid, i, j, m) and output (mid, i, m) on $\mathtt{AT.out}_j$.

Agent P_1

1. On input $(mid, 2, m)$ on $\mathtt{ST.in}_1$, output $(mid, 2, \mathtt{hello})$ on $\mathtt{AT.in}_1$.
2. On a later input $(mid, 2, ek)$ on $\mathtt{AT.out}_1$, sample a random encryption $c \leftarrow E_{ek}(m)$ and output $(mid, 2, c)$ on $\mathtt{AT.in}_1$.

Agent P_2

1. On input $(mid, 1, \mathtt{hello})$ on $\mathtt{AT.out}_2$, sample a random key pair (ek, dk) for an IND-CPA secure public-key cryptosystem and output $(mid, 1, ek)$ on $\mathtt{AT.in}_2$.
2. On a later input $(mid, 1, c)$ on $\mathtt{AT.out}_2$, output $(mid, 1, D_{dk}(c))$ on $\mathtt{ST.out}_2$.

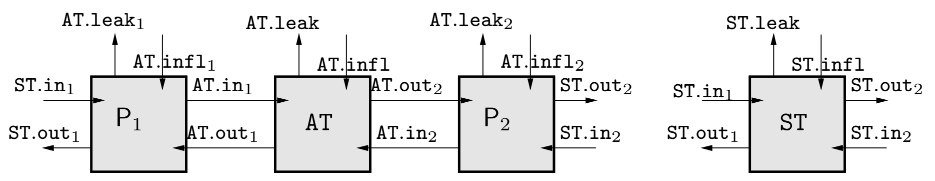


Figure 4.2 The protocol $\pi_{\mathtt{ST}}$ for secure transfer and the ideal functionality $\mathsf{F}_{\mathtt{ST}}$ for secure transfer

using an authenticated channel. To do a secure transfer from P_1 to P_2 one can use the following protocol:

1. First P_1 announces over the authenticated channel to P_2 that it wants to send a message securely.

2. Then P_2 samples a key pair (ek, dk) and sends the encryption key ek to P_1 over the authenticated channel.
3. Then P_1 encrypts the message, $c \leftarrow E_{ek}(m)$, and returns c over the authenticated channel.
4. Then P_2 outputs $m = D_{dk}(c)$.

We want to formally model this protocol within the UC model.

To model the above protocol for secure transfer we need an ideal functionality $\mathsf{F}_{\mathtt{AT}}$ for authenticated transfer, to use as resource for the protocol, see Agent $\mathrm{F}_{\mathtt{AT}}$. The only difference from $\mathsf{F}_{\mathtt{ST}}$ is that m is leaked, and not just $|m|$. This models that m is allowed to leak in an authenticated channel. The protocol $\pi_{\mathtt{ST}}$ is given by Agent P_1 and Agent P_2. Similar code is included for the direction from P_2 to P_1, and similarly between each other pair of parties. In Fig. 4.2 we show the protocol $\pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$ next to the ideal functionality $\mathsf{F}_{\mathtt{ST}}$ that it tries to implement. $\triangle$

Modeling Corruption

Back to how corruption is modeled. Again there are many choices, but we assume that all parties have the following standard corruption behavior:

- If a party P_i receives a special symbol (`passive corrupt`) on $\mathtt{R.infl}_i$, then P_i returns its internal state σ on $\mathtt{R.leak}_i$. The internal state σ consists of all randomness used by the party so far along with all inputs sent and received on its ports and the messages in the message queues of its inports.
- If a party P_i receives a special symbol (`active corrupt`) on $\mathtt{R.infl}_i$, then P_i outputs it current state on $\mathtt{R.leak}_i$ and starts executing the following rules, and only these rules:
  - On input $(\mathtt{read}, \mathtt{P})$ on $\mathtt{R.infl}_i$, where $\mathtt{P}$ is an inport of P_i, it reads the next message m on $\mathtt{P}$ and returns m on $\mathtt{R.leak}_i$.
  - On input $(\mathtt{send}, \mathtt{P}, m)$ on $\mathtt{R.infl}_i$, where $\mathtt{P}$ is an outport of P_i, it sends m on $\mathtt{P}$.

By passive corrupting a party in a protocol $\pi \diamond \mathcal{R}$ using resource $\mathcal{R}$, we mean that (`passive corrupt`) is input on $\mathtt{R.infl}_i$ and then $(\mathtt{passive\ corrupt}, i)$ is input on `R.infl` on the communication device $\mathcal{R}$. By active corrupting a party in $\pi \diamond \mathcal{R}$ we mean that (`active corrupt`) is input on $\mathtt{R.infl}_i$ and then $(\mathtt{active\ corrupt}, i)$ is input on `R.infl` on $\mathcal{R}$. After that P_i can be controlled using read and send commands.

4.2.4 The Simulator

If we inspect Fig. 4.2 we see that we have the problem discussed in the introduction to this chapter that a protocol, like $\pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$, and the ideal functionality specifying its intended behavior, like $\mathsf{F}_{\mathtt{ST}}$, have different open port structures and hence cannot be behaviorally equivalent. We also discussed that we solve this be introducing a simulator $\mathcal{S}$ which gives $\pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$ and $\mathsf{F}_{\mathtt{ST}} \diamond \mathcal{S}$ the same open port structure. An example is shown in Fig. 4.3.

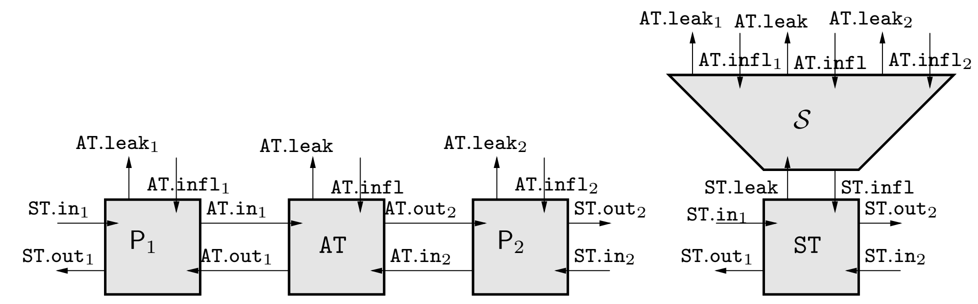


Figure 4.3 Port structure of the protocol $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$ for secure transfer and the ideal functionality $\mathsf{F}_{\texttt{ST}}$ for secure transfer composed with a simulator $\mathcal{S}$ for $\pi_{\texttt{ST}}$.

In general, a simulator $\mathcal{S}$ for a simple protocol π with protocol name `F` and resource name `R` is defined as follows:

It is a poly-time[7] interactive system with an open inport named `F.leak` and an open outport named `F.infl`. These two ports allows it to connect to an ideal functionality F with name `F`, as such an F has an open outport named `F.leak` and an open inport named `F.infl`. In addition to these two ports $\mathcal{S}$ has ports corresponding to the special ports of π, i.e., for $i = 1, \ldots, n$ it has inports named $\texttt{R.infl}_i$ and outports named $\texttt{R.leak}_i$, like the parties of π; finally $\mathcal{S}$ has an inport named `R.infl` and an outport named `R.leak`, like a resource $\mathcal{R}$ used by π. As a consequence, $\mathsf{F} \diamond \mathcal{S}$ and $\pi \diamond \mathcal{R}$ has the same open ports.

We call the open ports of $\mathcal{S}$ which connects to F the ideal functionality ports of $\mathcal{S}$. We call the other open ports of $\mathcal{S}$ the simulation ports of $\mathcal{S}$.

We additionally require that $\mathcal{S}$ is corruption preserving in the sense that it does not corrupt a party on F unless that party was corrupted in the simulation. I.e., $\mathcal{S}$ does not output $(\texttt{passive corrupt}, i)$ on `F.infl` unless it at some point received (`passive corrupt`) on $\texttt{R.infl}_i$, and $\mathcal{S}$ does not output $(\texttt{active corrupt}, i)$ on `F.infl` unless it at some point received (`active corrupt`) on $\texttt{R.infl}_i$. We also require that if $\mathcal{S}$ receives (`passive corrupt`) on $\texttt{R.infl}_i$, then its next action is to output $(\texttt{passive corrupt}, i)$ on `F.infl`, and similarly for active corruptions. This last requirement is a technicality needed when we later define so-called static security.

Finally we require that $\mathcal{S}$ is clock preserving. This is a technicality needed when we later define synchronous security. What we need is that $\mathcal{S}$ outputs $(\texttt{clockin}, i)$ on `F.infl` if and only if it just received $(\texttt{clockin}, i)$ on $\texttt{R.infl}_i$, and $\mathcal{S}$ outputs $(\texttt{clockout}, i)$ on `F.infl` if and only if it just received $(\texttt{clockout}, i)$ on $\texttt{R.infl}_i$. We shall return to the use of these commands later.

[7] A poly-time interactive system is a system which returns on an open outport within expected poly-time when it is activated on an open inport. For the interested reader, a formal definition is given in Definition 2.16.

4.2.5 The Environment

Having introduced the simulator we can now compare protocol π using resource $\mathcal{R}$, to an ideal functionality F with an attached simulator $\mathcal{S}$ for the protocol. We will that by requiring that $\pi \diamond \mathcal{R}$ is "hard" to distinguish from from $\mathsf{F} \diamond \mathcal{S}$. Indistinguishability of interactive systems was defined in Definition 2.18, in Chapter 2. The technical details can be found there, here we remind the reader of the main ideas, this will be sufficient to understand our discussion in this chapter.

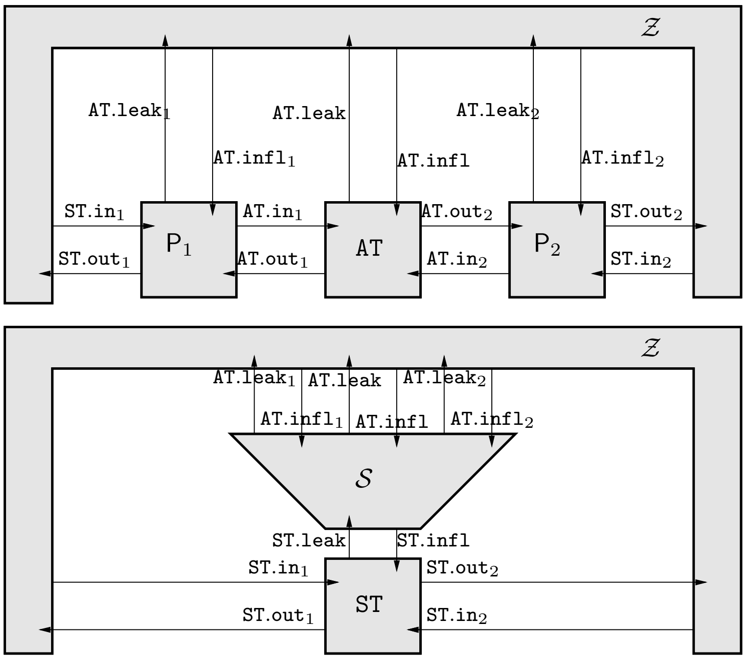


Figure 4.4 Example of two systems closed using the same environment $\mathcal{Z}$

The notion of indistinguishability requires that we consider an interactive system $\mathcal{Z}$ that must be an environment for $\pi \diamond \mathcal{R}$ and $\mathsf{F} \diamond \mathcal{S}$. This notion was defined for general interactive systems in Definition 2.18, and we now flesh out what it means in our concrete context of protocols and resources:

An environment $\mathcal{Z}$ for a simple protocol π with protocol name F and resource name R is an interactive system $\mathcal{Z}$ which have the dual open ports of $\pi \diamond \mathcal{R}$ where $\mathcal{R}$ is a resource with name R. Concretely, for each open inport P of $\pi \diamond \mathcal{R}$, the system $\mathcal{Z}$ has an open outport named P and for each open outport P of $\pi \diamond \mathcal{R}$, the system $\mathcal{Z}$ has an open inport named P. As a consequence $\pi \diamond \mathcal{R} \diamond \mathcal{Z}$ is a closed systems, and so is $\mathsf{F} \diamond \mathcal{S} \diamond \mathcal{Z}$. A closed system is just one with no open ports. See Fig. 4.4 for an example. Note that $\mathcal{S}$, being a simulator must be such that the port structure of $\pi \diamond \mathcal{R}$ and $\mathsf{F} \diamond \mathcal{S}$ are the same and therefore $\mathcal{Z}$ is an environment for both systems.

We call the open ports of $\mathcal{Z}$ which connects to the protocol ports of π the protocol ports of $\mathcal{Z}$. We call the other open ports of $\mathcal{Z}$ the special ports of $\mathcal{Z}$.

Two systems are indistinguishable to an environment $\mathcal{Z}$ if $\mathcal{Z}$ cannot tell them apart (except with negligible advantage) by sending and receiving messages on the open ports of the systems. Two systems such as $\pi \diamond \mathcal{R}$ and $\mathsf{F} \diamond \mathcal{S}$ are called indistinguishable to a class Z of environments if they are indistinguishable to all $\mathcal{Z} \in Z$. This is written $\pi \diamond \mathcal{R} \overset{Z}{\equiv} \mathsf{F} \diamond \mathcal{S}$. If, for instance, Z is the class of polynomial time environments, we speak about computational indistinguishability and write $\pi \diamond \mathcal{R} \overset{\text{comp}}{\equiv} \mathsf{F} \diamond \mathcal{S}$. If Z contains all environments (that activate the system only a polynomial number of times) we speak about statistical indistinguishability (written $\pi \diamond \mathcal{R} \overset{\text{stat}}{\equiv} \mathsf{F} \diamond \mathcal{S}$). Finally, if all environments have zero advantage in distinguishing, we speak about perfect indistinguishability (written $\pi \diamond \mathcal{R} \overset{\text{perf}}{\equiv} \mathsf{F} \diamond \mathcal{S}$).

4.2.6 Comparing Protocols to the Ideal Functionalities

We are now finally ready to define what it means that a protocol π using, say, resource F_{AT} does "the same" as an ideal functionality, say F_{ST}. The definition is general, but we reuse the names from the example for concreteness.

Definition 4.4 (Security for simple protocols) *Let F_{ST} be any ideal functionality with name ST, let π_{ST} be any simple protocol with protocol name ST and resource name AT, and let F_{AT} be any ideal functionality with name AT. We say that $\pi_{\mathsf{ST}} \diamond \mathsf{F}_{\mathsf{AT}}$ securely implements F_{ST} in environments Z if there exists a simulator $\mathcal{S}$ for π_{ST} such that $\pi_{\mathsf{ST}} \diamond \mathsf{F}_{\mathsf{AT}} \overset{Z}{\equiv} \mathsf{F}_{\mathsf{ST}} \diamond \mathcal{S}$.*

We will also write this as $\pi_{\mathsf{ST}} \diamond \mathsf{F}_{\mathsf{AT}} \overset{Z}{\geq} \mathsf{F}_{\mathsf{ST}}$. We will sometimes say that $\pi_{\mathsf{ST}} \diamond \mathsf{F}_{\mathsf{AT}}$ is at least as secure as F_{ST} for environments in Z.

By defining Z appropriately, we can use the definiiton to talk talk about perfect, statistical, computational security of π_{ST}.

For simplicity, this definition only considers protocols using a single resource. It is trivial to extend to several resources by allowing multiple resource names as long the port structure of simulators and environments are required to match this.

Example 4.7 As an example, we prove that $\pi_{\mathsf{ST}} \diamond \mathsf{F}_{\mathsf{AT}}$ securely implements F_{ST} in polynomial time environments. As a first step, we first consider a poly-time environment $\mathcal{Z}$ which does not corrupt any of the parties.

As we describe the simulator $\mathcal{S}$, it is instructive to look at Fig. 4.4. Recall that we try to construct some $\mathcal{S}$ for that setting such that $\mathcal{Z}$ cannot see which of the two systems it is playing with. The simulator is described in Agent $\mathcal{S}$.

The reason why $\mathcal{S}$ uses m' and not the real m is that F_{ST} only outputs $l = |m|$ to $\mathcal{S}$. Giving m to $\mathcal{S}$ would make the simulation trivial, but remember that the

Agent $\mathcal{S}$

A simulator for $\pi_{\mathtt{ST}}$. This is how to simulate a secure transfer from P_1 to P_2. All other directions are handled similarly.

1. On input $(mid, 1, 2, l)$ on ST.leak (from $\mathsf{F}_{\mathtt{ST}}$, and with $l = |m|$) it outputs $(mid, 1, 2, \mathtt{hello})$ on AT.leak.
2. On a later input $(\mathtt{deliver}, mid, 1, 2)$ on AT.infl it samples a key pair (ek, dk) and outputs $(mid, 2, 1, ek)$ on AT.leak.
3. On a later input $(\mathtt{deliver}, mid, 2, 1)$ on AT.infl it samples a random encryption $c \leftarrow E_{ek}(m')$ and outputs $(mid, 1, 2, C)$ on AT.leak. Here $m' = 0^l$ is an all-zero message of length l.
4. On a later input $(\mathtt{deliver}, mid, 1, 2)$ on AT.infl it outputs $(\mathtt{deliver}, mid, 1, 2)$ on ST.infl, which makes $\mathsf{F}_{\mathtt{ST}}$ output $(mid, 1, m)$ on $\mathtt{ST.out}_2$.

whole idea of $\mathcal{S}$ is to demonstrate that the real leakage can be simulated given only the leakage allowed by the ideal functionality, and m is not allowed to leak in a secure transfer.

Consider now some distinguisher $\mathcal{Z}$ which gets to play with either $\pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$ or $\mathsf{F}_{\mathtt{ST}} \diamond \mathcal{S}$. For now we assume that $\mathcal{Z}$ does not use the special ports of the parties — i.e., it makes no corruptions. Furthermore, for simplicity, we only allow $\mathcal{Z}$ to do one secure transfer and to do it only from P_1 to P_2. Then $\mathcal{Z}$ works as follows:

1. It picks some message m and inputs $(mid, 2, m)$ on $\mathtt{ST.in}_1$.
2. Then it sees $(mid, 1, 2, \mathtt{hello})$ on AT.leak and inputs $(\mathtt{deliver}, mid, 1, 2)$ on AT.infl.
3. Then it sees some $(mid, 2, 1, ek)$ on AT.leak and inputs $(\mathtt{deliver}, mid, 2, 1)$ on AT.infl.
4. Then it sees some $(mid, 1, 2, c'')$ on AT.leak and inputs $(\mathtt{deliver}, mid, 1, 2)$ on AT.infl.
5. In response to this it sees some $(mid, 1, m'')$ output on $\mathtt{ST.out}_2$.

It could, of course, refuse some of the deliveries. This would, however, only have $\mathcal{Z}$ see less messages and thus make the distinguishing of the systems harder.

Note that by design of the simulator $\mathcal{S}$, the distinguisher $\mathcal{Z}$ will see both system behave as specified above. The only difference between the two systems is that

- If $\mathcal{Z}$ is playing with $\mathsf{F}_{\mathtt{ST}} \diamond \mathcal{S}$, then $c'' \leftarrow E_{ek}(0^{|m|})$ and m'' is the message $m'' = m$ output by $\mathsf{F}_{\mathtt{ST}}$.
- If it is playing with $\pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$, then $c'' \leftarrow E_{ek}(m)$ and $m'' = D_{dk}(c'')$ is the message output by P_2.

If the encryption scheme has perfect correctness, then $D_{dk}(E_{ek}(m)) = m$, so $\pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$ and $\mathsf{F}_{\mathtt{ST}} \diamond \mathcal{S}$ always output the same m''. So, the only difference between the two systems is that $c'' \leftarrow E_{ek}(m)$ or $c'' \leftarrow E_{ek}(0^{|m|})$. So, a distinguisher $\mathcal{Z}$ essentially has the following job: Pick m and receive $(ek, E_{ek}(m'))$, where ek is random and $m' = m$ or $m' = 0^{|m|}$. Then try to distinguish which m' was used.

The definition of IND-CPA security more or less exactly says that no poly-time system can distinguish $(ek, E_{ek}(m))$ and $(ek, E_{ek}(0^{|m|}))$. Hence it follows that $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}} \diamond \mathcal{Z} \overset{\text{stat}}{\equiv} \mathsf{F}_{\texttt{ST}} \diamond \mathcal{S} \diamond \mathcal{Z}$ for all poly-time environments $\mathcal{Z}$ which do not corrupt parties and which do only one transfer, from P_1 to P_2. More formally, we can turn a poly-time system $\mathcal{Z}$ which distinguishes the two systems $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$ and $\mathsf{F}_{\texttt{ST}} \diamond \mathcal{S}$ into a poly-time system $\mathcal{Z}'$ which wins in the IND-CPA game in Section 2.5:

1. The system $\mathcal{Z}'$ will play with A_0 or A_1 as defined in Section 2.5.
2. First $\mathcal{Z}'$ receives ek from A_b (here $b = 0$ or $b = 1$).
3. Then $\mathcal{Z}'$ runs $\mathcal{Z}$ to see which message $(mid, 2, m)$ it outputs on $\texttt{ST.in}_1$.
4. Then $\mathcal{Z}'$ inputs $(m, 0^{|m|})$ to A_b and gets back an encryption c^*, where c^* is an encryption of m if $b = 0$ and c^* is an encryption of $0^{|m|}$ if $b = 1$.
5. Then $\mathcal{Z}'$ runs $\mathcal{Z}$ and shows it the messages $(mid, 1, 2, \texttt{hello})$, $(mid, 2, 1, ek)$, $(mid, 1, 2, c^*)$ on $\texttt{AT.leak}$ and $(mid, 1, m)$ on $\texttt{ST.out}_2$.
6. Then $\mathcal{Z}'$ runs $\mathcal{Z}$ until it outputs its guess c, and then $\mathcal{Z}'$ outputs c.

If $b = 0$, then c^* is a random encryption of m, and if $b = 1$, then c^* is a random encryption of $0^{|m|}$. So, if $b = 0$, then $\mathcal{Z}$ see exactly the interaction it would see when interacting with $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$, and if $b = 1$, then $\mathcal{Z}$ sees exactly the interaction it would see when interacting with $\mathsf{F}_{\texttt{ST}} \diamond \mathcal{S}$. So, if $\mathcal{Z}$ can distinguish $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$ and $\mathsf{F}_{\texttt{ST}} \diamond \mathcal{S}$, then $c = b$ with probability significantly better than $\frac{1}{2}$. But this exactly means that $\mathcal{Z}'$, who also outputs c, will guess b with probability significantly better than $\frac{1}{2}$. If the encryption scheme is IND-CPA secure, then this is a contradiction, as $\mathcal{Z}'$ is poly-time when $\mathcal{Z}$ is poly-time and IND-CPA security means that no poly-time system can distinguish A_0 and A_1.

It is fairly easy to extend the argument to poly-time environments $\mathcal{Z}$ which do not corrupt parties but are allowed to do any number of transfers. Each transfer is simulated as above, and the security follows from the fact that IND-CPA security is maintained even if you get so see many encryptions. Later we will extend the analysis also to poly-time environments which are allowed to corrupt parties. $\triangle$

4.2.7 Composed Protocols

An important property of the UC framework is that when $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$ implements $\mathsf{F}_{\texttt{ST}}$ with computational security, then $\mathsf{F}_{\texttt{ST}}$ can securely be replaced by $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$ in any poly-time protocol.

Consider some third ideal functionality $\mathsf{F}_{\texttt{N}}$ doing some interesting task ideally secure. Assume that we can design a protocol $\pi_{\texttt{N}}$ with protocol name $\texttt{N}$ and resource name $\texttt{ST}$, which securely implements $\mathsf{F}_{\texttt{N}}$ when it uses $\mathsf{F}_{\texttt{ST}}$ as communication resource. Designing a secure implementation of $\mathsf{F}_{\texttt{N}}$ using secure transfer as communication resource is potentially much easier than designing a protocol using only authenticated transfer as communication resource. The structure of such a protocol is shown in the top row of Fig. 4.5, along with $\mathsf{F}_{\texttt{N}}$.

To get an implementation using only authenticated transfer, we can replace the use of $\mathsf{F}_{\texttt{ST}}$ in $\pi_{\texttt{N}} \diamond \mathsf{F}_{\texttt{ST}}$ by the use of the protocol $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$, resulting in the protocol

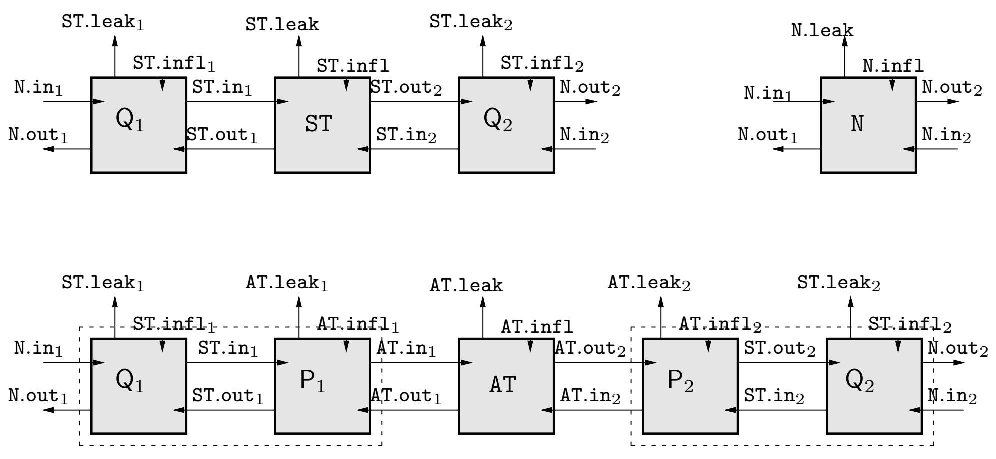


Figure 4.5 The protocol $\pi_{\mathtt{N}} \diamond \mathsf{F}_{\mathtt{ST}}$, the ideal functionality $\mathsf{F}_{\mathtt{N}}$, and the protocol $\pi_{\mathtt{N}} \diamond (\pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}})$

$\pi_{\mathtt{N}} \diamond (\pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}})$. This is possible as $\mathsf{F}_{\mathtt{ST}}$ and $\pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$ have the same structure of protocol ports. The result is shown in the bottom row in Fig. 4.5.

We have that $\pi_{\mathtt{N}} \diamond (\pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}) = (\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}}) \diamond \mathsf{F}_{\mathtt{AT}}$, as interactive systems are just sets of agents and $\diamond$ just the union operator on sets. We call $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}}$ a composed protocol. In general a composed protocol is just an interactive system composed of simple protocols. We use the term protocol to cover both simple protocols and composed protocols.

The protocol name of a composed protocol is the protocol name of the outer simple protocol, as it is this outer protocol which provides the open protocol ports. As an example, the protocol name of $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}}$ is `N`.

The resource name of a composed protocol is the protocol name of the inner simple protocol, as it is this inner protocol which provides the open resource ports. As an example, the resource name of $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}}$ is `AT`.

We can also compose protocols which are already composed, as long as the resource name of the outer composed protocol matches the protocol name of the inner composed protocol.

Composed Parties

We consider Q_1 and P_1 as one party and consider P_2 and Q_2 as one party. This leads to a notion of a composed party, which are just interactive systems consisting of simple parties. In these words, a composed protocol π simply consist of n composed parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, i.e., $\pi = \mathcal{P}_1 \diamond \cdots \diamond \mathcal{P}_n$. We use the term party to cover both simple parties and composed parties.

As an example, $\mathcal{P}_1 = \{\mathsf{Q}_1, \mathsf{P}_1\}$ is just an interactive system with open ports $\mathtt{AT.in}_1$ and $\mathtt{AT.out}_1$ connecting it to the protocol's communication device $\mathsf{F}_{\mathtt{AT}}$ and with open protocol ports $\mathtt{N.in}_1$ and $\mathtt{N.out}_1$ named as the ideal functionality $\mathsf{F}_{\mathtt{N}}$ that the protocol is trying to implement. The ports inside $\mathcal{P}$ are just particularities of how the party is implemented.

In addition $\mathcal{P}_1$ has some special ports which allow to corrupt it. We could insist on somehow joining these ports, but allowing to corrupt the components of $\mathcal{P}_1$ separately just gives more power to the attacker. A passive corruption of $\mathcal{P}_1$ is done by inputting (`passive corrupt`) on both $\mathtt{ST.infl}_1$ and $\mathtt{AT.infl}_1$, and inputting (`passive corrupt`, 1) on `AT.infl`, in response to which one receives the internal state of both components of the party plus the internal state of the party on the communication device. An active corruption of $\mathcal{P}_1$ is done by inputting (`active corrupt`) on both $\mathtt{ST.infl}_1$ and $\mathtt{AT.infl}_1$, and inputting (`active corrupt`, 1) on `AT.infl`.

Security of Composed Protocols

We define security of composed protocols as we did for simple protocols. Consider the protocol $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$ in the bottom row of Fig. 4.5. If we want to ask if it is a secure implementation of $\mathsf{F}_{\mathtt{N}}$ in the upper right corner of Fig. 4.5, we have the usual problem that the port structures are different. We therefore introduce a simulator $\mathcal{U}$. As usual, this simulator has an open outport `N.infl` and an open inport `N.leak`, so it can connect to $\mathsf{F}_{\mathtt{N}}$ to exploit the allowed leakage and influence of this ideal functionality. Its job is then to simulate the leakage and influence of the protocol $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$. Therefore $\mathcal{U}$ must have an open port for each of the special ports in $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$, see Fig. 4.6.

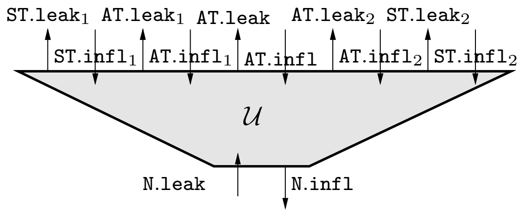


Figure 4.6 Open port structure of a simulator for the composed protocol $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$.

In general a simulator for a composed protocol π with protocol name `F` and resource name `R` is just an interactive system $\mathcal{S}$ with an open outport `F.infl` and an open inport `F.leak`, so it can connect to the special ports of $\mathsf{F}_{\mathtt{F}}$. It must further have open ports that match the special ports of all simple parties and resources used in the composed protocol. With this extension of the simulator, we can reuse Definition 4.4 to say what it means for a composed protocol to implement a functionality.

4.2.8 The UC Theorem

We are now ready to phrase and prove the UC Theorem. Before doing so, it is, however, instructive to give a pictorial proof using our secure transfer example.

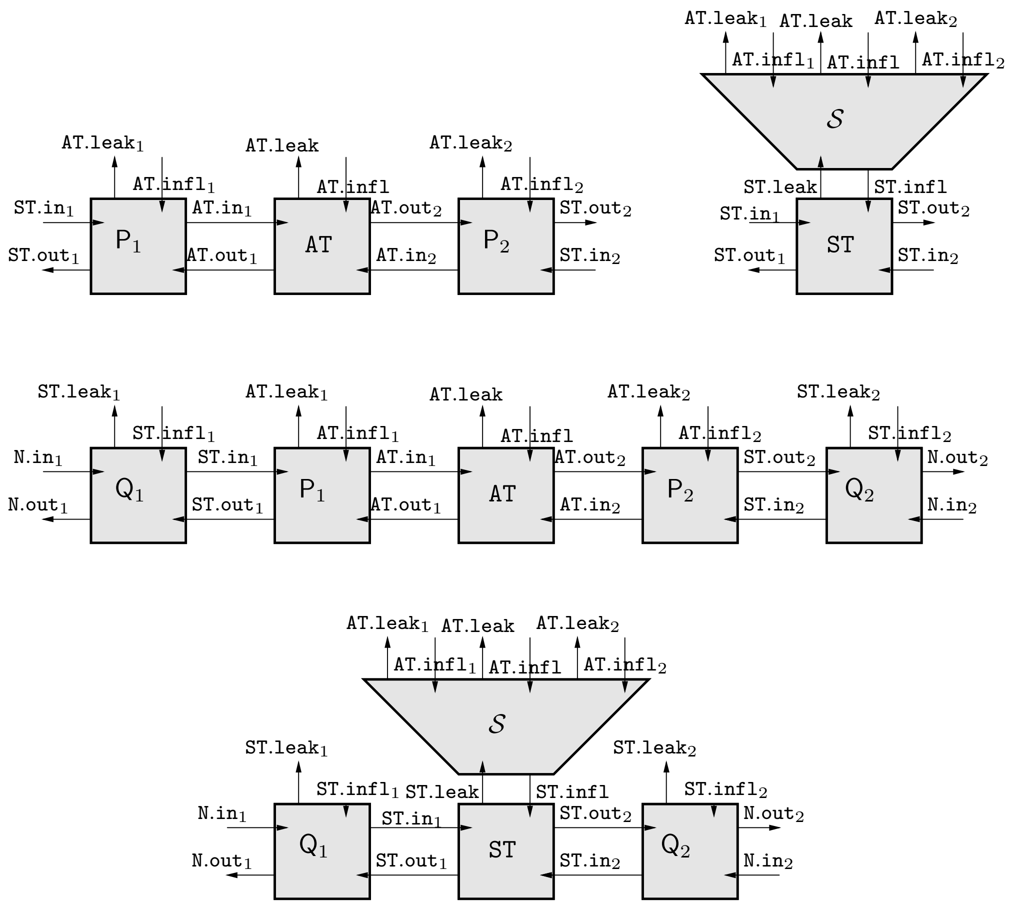


Figure 4.7 Extending $\pi_{\texttt{ST}}$ and $\mathsf{F}_{\texttt{ST}} \diamond \mathcal{S}$ with Q_1 and Q_2

Example 4.8 Let $\mathcal{Z}$ be the set of poly-time environments. We assumed that $\pi_{\texttt{N}} \diamond \mathsf{F}_{\texttt{ST}} \overset{\text{Env}}{\geqslant} \mathsf{F}_{\texttt{N}}$ and we argued that $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}} \overset{\text{Env}}{\geqslant} \mathsf{F}_{\texttt{ST}}$, though we still did not consider all cases in the analysis. Those two security guarantees allows us to conclude that $\pi_{\texttt{N}} \diamond \pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}} \overset{\text{Env}}{\geqslant} \mathsf{F}_{\texttt{N}}$. Notice that if all the systems where real numbers and $\diamond$ was addition and $\overset{\text{Env}}{\geqslant}$ was $\geq$, then our assumptions would be that $\pi_{\texttt{N}} + \mathsf{F}_{\texttt{ST}} \geq \mathsf{F}_{\texttt{N}}$ and $\pi_{\texttt{ST}} + \mathsf{F}_{\texttt{AT}} \geq \mathsf{F}_{\texttt{ST}}$. From this we could conclude as follows: $\pi_{\texttt{N}} + \pi_{\texttt{ST}} + \mathsf{F}_{\texttt{AT}} = \pi_{\texttt{N}} + (\pi_{\texttt{ST}} + \mathsf{F}_{\texttt{AT}}) \geq \pi_{\texttt{N}} + \mathsf{F}_{\texttt{ST}} \geq \mathsf{F}_{\texttt{N}}$. Then it follows from the transitivity of $\geq$ that

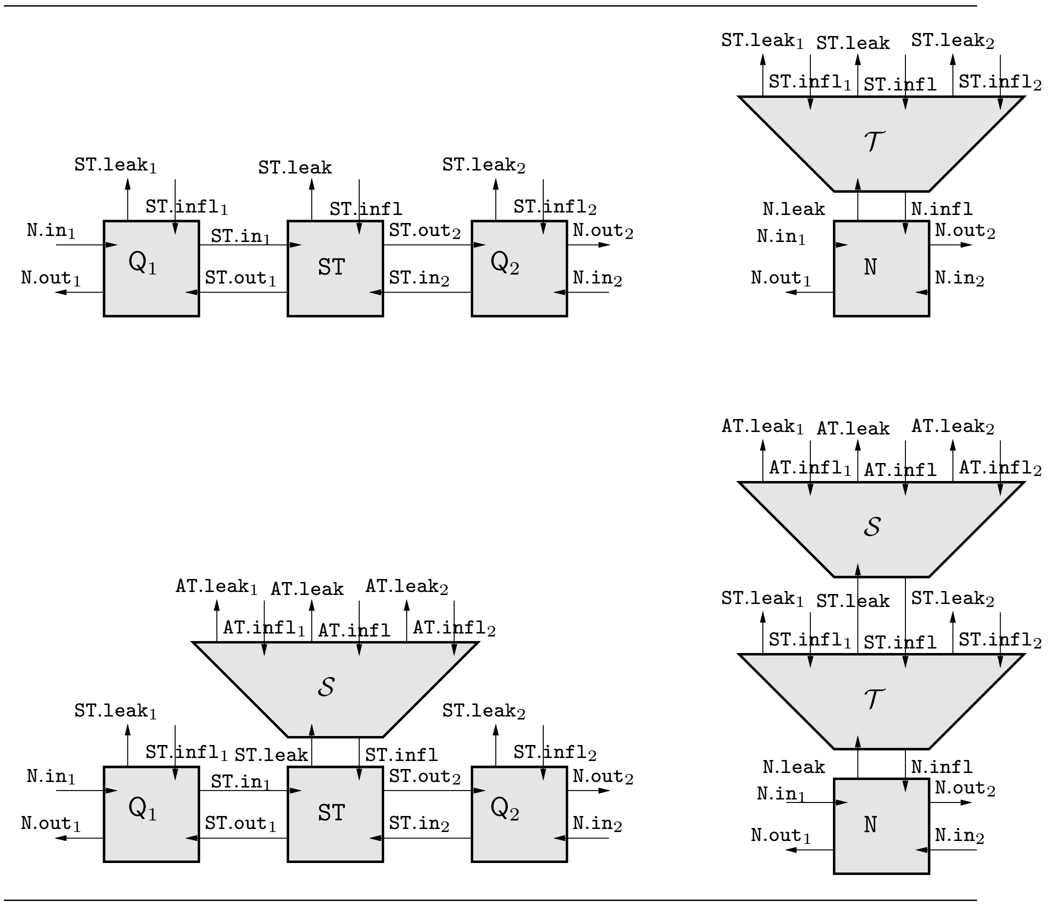


Figure 4.8 Extending $\pi_{\mathtt{N}}$ and $\mathsf{F}_{\mathtt{N}} \diamond \mathcal{T}$ with $\mathcal{S}$

$\pi_{\mathtt{N}} + \pi_{\mathtt{ST}} + \mathsf{F}_{\mathtt{AT}} \geq \mathsf{F}_{\mathtt{N}}$, which would correspond to $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}} \overset{\text{Env}}{\geqslant} \mathsf{F}_{\mathtt{N}}$. The proof of the UC Theorem follows this line of arguing. The proof goes as follows.

From $\pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}} \overset{\text{Env}}{\geqslant} \mathsf{F}_{\mathtt{ST}}$ we know that there exists a simulator $\mathcal{S}$ such that the two systems in the top row of Fig. 4.7 cannot be told apart by playing with their open ports. So, if we connect $\pi_{\mathtt{N}}$ to both systems, where $\pi_{\mathtt{N}}$ is represented by Q_1 and Q_2 in the figure, we know that the two bottom systems in Fig. 4.7 cannot be told apart. Namely, if some environment $\mathcal{Z}$ plays with the open ports of one of the bottom systems, it is really just playing with one of the system in the top row, via $\pi_{\mathtt{N}}$. I.e., $\mathcal{Z} \diamond \pi_{\mathtt{N}}$ is playing with one of the system in the top row. So, if $\mathcal{Z}$ distinguishes the bottom systems, then $\mathcal{Z} \diamond \pi_{\mathtt{N}}$ distinguishes the top systems. Since these systems cannot be told apart by any environment, it follows that no $\mathcal{Z}$ can distinguish the bottom systems.

From $\pi_{\mathtt{N}} \diamond \mathsf{F}_{\mathtt{ST}} \overset{\text{Env}}{\geqslant} \mathsf{F}_{\mathtt{N}}$ we know that there exists a simulator $\mathcal{T}$ such that the two systems in the top row of Fig. 4.8 cannot be told apart by playing with their open ports. So, if we connect $\mathcal{S}$ to both systems, we know that the two bottom systems cannot be told apart by playing with their open ports, using the same logic as above. Here we turn a successful distinguisher $\mathcal{Z}$ for the bottom systems into a successful distinguisher $\mathcal{Z} \diamond \mathcal{S}$ for the top systems and reach a contradiction.

Now, if $\mathcal{IS}_0$ cannot be told apart from $\mathcal{IS}_1$ and $\mathcal{IS}_1$ cannot be told apart from $\mathcal{IS}_2$, it follows that $\mathcal{IS}_0$ cannot be told apart from $\mathcal{IS}_2$. This follows from transitivity of indistinguishability as shown in Chapter 2. We use this transitivity to finish the proof. The protocol $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$ is the middle system in Fig. 4.7. We concluded that this system cannot be told apart from the system in the bottom row of Fig. 4.7. The same system is found as the bottom, left system in Fig. 4.8, and we concluded that this system cannot be told apart from the bottom right system in Fig. 4.8. The bottom, right system in Fig. 4.8 is $\mathsf{F}_{\mathtt{N}} \diamond \mathcal{T} \diamond \mathcal{S}$. So, we can conclude that $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$ cannot be told apart from $\mathsf{F}_{\mathtt{N}} \diamond \mathcal{T} \diamond \mathcal{S}$. If we let $\mathcal{U} = \mathcal{T} \diamond \mathcal{S}$, then it follows that $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$ cannot be told apart from $\mathsf{F}_{\mathtt{N}} \diamond \mathcal{U}$. But that means that $\mathcal{U}$ successfully simulates $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$ given $\mathsf{F}_{\mathtt{N}}$,[8] so $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}} \diamond \mathsf{F}_{\mathtt{AT}}$ is a secure implementation of $\mathsf{F}_{\mathtt{N}}$. $\triangle$

We are now almost ready to phrase and prove the UC Theorem. The last definition we need is the notion of a class of environments. The purpose of this notion is clear from the above example. Recall that we said that if some environment $\mathcal{Z}$ can tell apart the two bottom systems Fig. 4.7, then $\pi_{\mathtt{N}} \diamond \mathcal{Z}$ can tell apart the two systems in the top row of Fig. 4.7. We said that this leads to a contradiction as no environment can tell apart the two systems in the top row of Fig. 4.7 by assumption. Notice, however, that this requires that $\pi_{\mathtt{N}} \diamond \mathcal{Z}$ *is an environment.* We assume that *no environment* can tell apart the two systems in the top row of Fig. 4.7. So, if $\pi_{\mathtt{N}} \diamond \mathcal{Z}$ is not an environment, we do not get a contradiction. So, we need to require that if we take a protocol π and an environment $\mathcal{Z}$ for π, then $\mathcal{Z} \diamond \pi$ is again an environment. For similar reasons we need to assume that if we take an environment $\mathcal{Z}$ and a simulator $\mathcal{S}$, then $\mathcal{Z} \diamond \mathcal{S}$ is again an environment. As an example, if we work with the class of environments which are poly-time, then we need that if we take such an environment $\mathcal{Z}$ and a protocol π, then $\mathcal{Z} \diamond \pi$ is again poly-time. If we define poly-time environments in a proper way and only consider poly-time protocols, then this will be the case, as we demonstrate below.

DEFINITION 4.5 *We let* Pro *be the set of simple and composed protocols where all simple parties follows the rules for clocked entities and are recursive poly-time.*

It is straight-forward to verify the following lemma.

LEMMA 4.6 *If* $\pi_{\mathtt{F}} \in \mathrm{Pro}$ *is a protocol with protocol name* $\mathtt{F}$ *and resource name* $\mathtt{R}$ *and* $\pi_{\mathtt{R}} \in \mathrm{Pro}$ *is a protocol with protocol name* $\mathtt{R}$ *and* $\pi_{\mathtt{F}} \diamond \pi_{\mathtt{R}} \neq \bot$*, then* $\pi_{\mathtt{F}} \diamond \mathtt{R} \in \mathrm{Pro}$.

DEFINITION 4.7 *We let* Sim *be the set of interactive systems* $\mathcal{S}$ *which are a sim-*

[8] Notice, in particular, that $\mathcal{T} \diamond \mathcal{S}$ is a system which can connect to the special ports of $\mathsf{F}_{\mathtt{N}}$ and which has open special ports corresponding to $\pi_{\mathtt{N}} \diamond \pi_{\mathtt{ST}}$.

ulator for some protocol. We remind the reader that this means that for $\mathcal{S} \in \mathrm{Sim}$ *it holds that*

1. *There exists a protocol* π_F *(composed or simple) with protocol name* $\mathtt{F}$ *and an ideal functionality* F_F *with name* $\mathtt{F}$ *such that* π_F *and* $\mathsf{F}_F \diamond \mathcal{S}$ *have the same special ports.*
2. $\mathcal{S}$ *follows the rules for clocked entities and is recursive poly-time.*
3. $\mathcal{S}$ *is corruption preserving.*
4. $\mathcal{S}$ *is clock preserving.*

It is straight-forward to verify the following lemma.

Lemma 4.8 *If* $\mathcal{S} \in \mathrm{Sim}$ *and* $\mathcal{T} \in \mathrm{Sim}$ *and* $\mathcal{T}$ *has two open special ports which connects it to the ideal functionality ports of* $\mathcal{S}$ *and* $\mathcal{S} \diamond \mathcal{T} \neq \perp$*, then* $\mathcal{S} \diamond \mathcal{T} \in \mathrm{Sim}$.

Definition 4.9 *We say that* Env *is an* environment class *if the following holds.*

1. *Each* $\mathcal{Z} \in \mathrm{Env}$ *have the open port structure of an environment for some simple or composed protocol.*
2. *For all* $\mathcal{Z} \in \mathrm{Env}$ *and all* $\pi \in \mathrm{Pro}$ *where* $\pi \diamond \mathcal{Z} \neq \perp$ *and the protocol ports of* π *connect to the protocol ports of* $\mathcal{Z}$ *in* $\pi \diamond \mathcal{Z}$ *it holds that* $\pi \diamond \mathcal{Z} \in \mathrm{Env}$.
3. *For all* $\mathcal{Z} \in \mathrm{Env}$ *and all* $\mathcal{S} \in \mathrm{Sim}$ *where* $\mathcal{S} \diamond \mathcal{Z} \neq \perp$ *and the simulation ports of* $\mathcal{S}$ *connect to the special ports of* $\mathcal{Z}$*, it holds that* $\mathcal{S} \diamond \mathcal{Z} \in \mathrm{Env}$.

In the following we call an environment $\mathcal{Z}$ recursive poly-time if it is poly-time and it makes at most an expected polynomial number of calls before it makes its guess.

Proposition 4.10 *Let* Env^{poly} *be the set of all recursive poly-time systems which have an open port structure of an environment for some simple or composed protocol. Then* Env^{poly}*is an environment class.*

Proof 1) The first property of an environment class is fulfilled by definition.

2) If $\mathcal{Z} \in \mathrm{Env}^{\mathsf{poly}}$ and $\pi \in \mathrm{Pro}$ and the protocol ports of π connect to the protocol ports of $\mathcal{Z}$ in $\pi \diamond \mathcal{Z}$, then $\pi \diamond \mathcal{Z}$ has an open port structure of an environment for some simple or composed protocol. Note that this is true because we compose $\mathcal{Z}$ with only π, we do not include the resource used by π. Hence, if π has resource name $\mathtt{R}$, $\pi \diamond \mathcal{Z}$ has an open port structure matching a protocol with protocol name $\mathtt{R}$. So, to show that $\pi \diamond \mathcal{Z} \in \mathrm{Env}^{\mathsf{poly}}$ we just have to show that $\pi \diamond \mathcal{Z}$ is recursive poly-time. But since $\mathcal{Z} \in \mathrm{Env}^{\mathsf{poly}}$ it is recursive poly-time. Moreover $\pi \in \mathrm{Pro}$ is also recursive poly-time by definition, so $\pi \diamond \mathcal{Z}$ is recursive poly-time, as desired.

3) If $\mathcal{Z} \in \mathrm{Env}^{\mathsf{poly}}$ and $\mathcal{S} \in \mathrm{Sim}$ and the simulation ports of $\mathcal{S}$ connect to the special ports of $\mathcal{Z}$, it holds that $\mathcal{S} \diamond \mathcal{Z}$ has an open port structure of an environment for some simple or composed protocol. So, to show that $\mathcal{S} \diamond \mathcal{Z} \in \mathrm{Env}^{\mathsf{poly}}$ we just have to show that $\mathcal{S} \diamond \mathcal{Z}$ is recursive poly-time. This follows as for $\pi \diamond \mathcal{Z}$ as $\mathcal{S} \in \mathrm{Sim}$ implies that $\mathcal{S}$ is recursive poly-time. □

EXERCISE 4.1 Show that if Env_1 and Env_2 are environment classes, then $\text{Env}_1 \cap \text{Env}_2$ is an environment class and $\text{Env}_1 \cup \text{Env}_2$ is an environment class.

The following definition is very similar to Definition 4.4 but we now consider also composed protocols, and we make all the demands on environments that are necessary to prove the UC Theorem.

DEFINITION 4.11 (UC SECURITY FOR PROTOCOLS) *Let $\mathsf{F}_\mathtt{F}$ be an ideal functionality with name* F*, let $\pi_\mathtt{F}$ be a protocol with protocol name* F *and resource name* R*, and let $\mathsf{F}_\mathtt{R}$ be an ideal functionality with name* R*. Let* Env *be an environment class. We say that $\pi_\mathtt{F} \diamond \mathsf{F}_\mathtt{R}$* securely implements *$\mathsf{F}_\mathtt{F}$ in environments* Env *if there exists a simulator $\mathcal{S} \in \text{Sim}$ for $\pi_\mathtt{F}$ such that $\pi_\mathtt{F} \diamond \mathsf{F}_\mathtt{R} \overset{\text{Env}}{\equiv} \mathsf{F}_\mathtt{F} \diamond \mathcal{S}$.*

We will also write this as $\pi_\mathtt{F} \diamond \mathsf{F}_\mathtt{R} \overset{\text{Env}}{\geqslant} \mathsf{F}_\mathtt{F}$, and we will sometimes say that $\pi_\mathtt{F} \diamond \mathsf{F}_\mathtt{R}$ is at least as secure as $\mathsf{F}_\mathtt{ST}$ *for environments in Z.*

THEOREM 4.12 (THE UC THEOREM) *Let* Env *be an environment class. Let $\pi_\mathtt{F} \in$* Pro *be a protocol with protocol name* F *and resource name* G*. Let $\pi_\mathtt{G}$ be a protocol with protocol name* G *and resource name* H *and for which $\pi_\mathtt{F} \diamond \pi_\mathtt{G} \neq \bot$. Let $\mathsf{F}_\mathtt{F}$, $\mathsf{F}_\mathtt{G}$ and $\mathsf{F}_\mathtt{H}$ be ideal functionalities with names* F, G *respectively* H*. If $\pi_\mathtt{F} \diamond \mathsf{F}_\mathtt{G} \overset{\text{Env}}{\geqslant} \mathsf{F}_\mathtt{F}$ and $\pi_\mathtt{G} \diamond \mathsf{F}_\mathtt{H} \overset{\text{Env}}{\geqslant} \mathsf{F}_\mathtt{G}$, then $(\pi_\mathtt{F} \diamond \pi_\mathtt{G}) \diamond \mathsf{F}_\mathtt{H} \overset{\text{Env}}{\geqslant} \mathsf{F}_\mathtt{F}$.*

PROOF We start by writing out the premises of the theorem, to make them easier to use. We assume that $\pi_\mathtt{F} \diamond \mathsf{F}_\mathtt{G} \overset{\text{Env}}{\geqslant} \mathsf{F}_\mathtt{F}$. This means that there exists a simulator $\mathcal{S} \in \text{Sim}$ for $\pi_\mathtt{F}$ such that

$$\pi_\mathtt{F} \diamond \mathsf{F}_\mathtt{G} \diamond \mathcal{Z}_1 \overset{\text{stat}}{\equiv} \mathsf{F}_\mathtt{F} \diamond \mathcal{S} \diamond \mathcal{Z}_1 \tag{4.1}$$

for all $\mathcal{Z}_1 \in \text{Env}$ for which the compositions make sense. Note that this is just a convenient (and equivalent) way to say that $\pi_\mathtt{F} \diamond \mathsf{F}_\mathtt{G} \overset{\text{Env}}{\equiv} \mathsf{F}_\mathtt{F} \diamond \mathcal{S}$

We also assume that $\pi_\mathtt{G} \diamond \mathsf{F}_\mathtt{H} \overset{\text{Env}}{\geqslant} \mathsf{F}_\mathtt{G}$. This means that there exists a simulator $\mathcal{T} \in \text{Sim}$ for $\pi_\mathtt{G}$ such that

$$\pi_\mathtt{G} \diamond \mathsf{F}_\mathtt{H} \diamond \mathcal{Z}_2 \overset{\text{stat}}{\equiv} \mathsf{F}_\mathtt{G} \diamond \mathcal{T} \diamond \mathcal{Z}_2 \tag{4.2}$$

for all $\mathcal{Z}_2 \in \text{Env}$ for which the compositions make sense.

We then write out the conclusion, to better see what we have to prove. We want to prove that $\pi_\mathtt{F} \diamond \pi_\mathtt{G} \diamond \mathsf{F}_\mathtt{H} \overset{\text{Env}}{\geqslant} \mathsf{F}_\mathtt{H}$. This means that there should exist a simulator $\mathcal{U} \in \text{Sim}$ for $\pi_\mathtt{F} \diamond \pi_\mathtt{G}$ such that

$$\pi_\mathtt{F} \diamond \pi_\mathtt{G} \diamond \mathsf{F}_\mathtt{H} \diamond \mathcal{Z} \overset{\text{stat}}{\equiv} \mathsf{F}_\mathtt{F} \diamond \mathcal{U} \diamond \mathcal{Z} \tag{4.3}$$

for all $\mathcal{Z} \in \text{Env}$ for which the compositions make sense.

The flow of the proof is as follows. We first show that

$$\pi_\mathtt{F} \diamond \pi_\mathtt{G} \diamond \mathsf{F}_\mathtt{H} \diamond \mathcal{Z} \overset{\text{stat}}{\equiv} \pi_\mathtt{F} \diamond \mathsf{F}_\mathtt{G} \diamond \mathcal{T} \diamond \mathcal{Z} \ . \tag{4.4}$$

Then we show that

$$\pi_\mathtt{F} \diamond \mathsf{F}_\mathtt{G} \diamond \mathcal{T} \diamond \mathcal{Z} \overset{\text{stat}}{\equiv} \mathsf{F}_\mathtt{F} \diamond \mathcal{S} \diamond \mathcal{T} \diamond \mathcal{Z} \ . \tag{4.5}$$

Then we use transitivity of $\overset{\text{stat}}{\equiv}$ on Eq. 4.4 and Eq. 4.5 to conclude that

$$\pi_{\mathtt{F}} \diamond \pi_{\mathtt{G}} \diamond \mathsf{F}_{\mathtt{H}} \diamond \mathcal{Z} \overset{\text{stat}}{\equiv} \mathsf{F}_{\mathtt{F}} \diamond \mathcal{S} \diamond \mathcal{T} \diamond \mathcal{Z} \ . \tag{4.6}$$

Then we observe that if we $\mathcal{U} \overset{\text{def}}{=} \mathcal{S} \diamond \mathcal{T}$, then $\mathcal{U} \in$ Sim. Plugging this into Eq. 4.6, we get exactly Eq. 4.3. What remains is therefore just to show Eq. 4.4 and Eq. 4.5.

Consider first Eq. 4.4 and let $\mathcal{Z}_2 \overset{\text{def}}{=} \pi_{\mathtt{F}} \diamond \mathcal{Z}$. Then

$$\begin{aligned}\pi_{\mathtt{G}} \diamond \mathsf{F}_{\mathtt{H}} \diamond \mathcal{Z}_2 &= \pi_{\mathtt{G}} \diamond \mathsf{F}_{\mathtt{H}} \diamond \pi_{\mathtt{F}} \diamond \mathcal{Z} = \pi_{\mathtt{F}} \diamond \pi_{\mathtt{G}} \diamond \mathsf{F}_{\mathtt{H}} \diamond \mathcal{Z} \\ \mathsf{F}_{\mathtt{G}} \diamond \mathcal{T} \diamond \mathcal{Z}_2 &= \mathsf{F}_{\mathtt{G}} \diamond \mathcal{T} \diamond \pi_{\mathtt{F}} \diamond \mathcal{Z} = \pi_{\mathtt{F}} \diamond \mathsf{F}_{\mathtt{G}} \diamond \mathcal{T} \diamond \mathcal{Z}\end{aligned}$$

So, to prove Eq. 4.4 we just have to prove that

$$\pi_{\mathtt{G}} \diamond \mathsf{F}_{\mathtt{H}} \diamond \mathcal{Z}_2 \overset{\text{stat}}{\equiv} \mathsf{F}_{\mathtt{G}} \diamond \mathcal{T} \diamond \mathcal{Z}_2 \ . \tag{4.7}$$

This, however, follows directly from Eq. 4.2, as $\mathcal{Z}_2 \in$ Env when $\pi_{\mathtt{F}} \in$ Pro and $\mathcal{Z} \in$ Env.

Consider then Eq. 4.5 and let $\mathcal{Z}_2 \overset{\text{def}}{=} \mathcal{T} \diamond \mathcal{Z}$. Then to prove Eq. 4.4 we just have to prove that

$$\pi_{\mathtt{F}} \diamond \mathsf{F}_{\mathtt{G}} \diamond \mathcal{Z}_1 \overset{\text{stat}}{\equiv} \mathsf{F}_{\mathtt{F}} \diamond \mathcal{S} \diamond \mathcal{Z}_1 \ . \tag{4.8}$$

This, however, follows directly from Eq. 4.1, as $\mathcal{Z}_1 \in$ Env when $\mathcal{S} \in$ Sim and $\mathcal{Z} \in$ Env. □

4.2.9 Extensions

It is possible to extend the UC Theorem in several ways. As an example, we say (with the same notation as in the UC theorem) that $\pi_{\mathtt{F}}$ is a perfect secure implementation of $\mathsf{F}_{\mathtt{F}}$ in environments Env if there exist a simulator $\mathcal{S} \in$ Sim for $\pi_{\mathtt{F}}$ such that

$$\pi_{\mathtt{F}} \diamond \mathsf{F}_{\mathtt{R}} \diamond \mathcal{Z} \overset{\text{perf}}{\equiv} \mathsf{F}_{\mathtt{F}} \diamond \mathcal{S} \diamond \mathcal{Z}$$

for all $\mathcal{Z} \in$ Env instead of just

$$\pi_{\mathtt{F}} \diamond \mathsf{F}_{\mathtt{R}} \diamond \mathcal{Z} \overset{\text{stat}}{\equiv} \mathsf{F}_{\mathtt{F}} \diamond \mathcal{S} \diamond \mathcal{Z} \ .$$

The following theorem follows directly from the proof of Theorem 4.12.

THEOREM 4.13 *Let* Env *be an environment class. Let $\pi_{\mathtt{F}} \in$* Pro *be a protocol with protocol name* F *and resource name* G*. Let $\pi_{\mathtt{G}}$ be a protocol with protocol name* G *and resource name* H *and for which $\pi_{\mathtt{F}} \diamond \pi_{\mathtt{G}} \neq \perp$. Let $\mathsf{F}_{\mathtt{F}}$, $\mathsf{F}_{\mathtt{G}}$ and $\mathsf{F}_{\mathtt{H}}$ be ideal functionalities with names* F*,* G *respectively* H*. If $\pi_{\mathtt{F}} \diamond \mathsf{F}_{\mathtt{G}}$ is a perfect secure implementation of $\mathsf{F}_{\mathtt{F}}$ in the environments* Env *and $\pi_{\mathtt{G}} \diamond \mathsf{F}_{\mathtt{H}}$ is a perfect secure implementation of $\mathsf{F}_{\mathtt{G}}$ in the environments* Env*, then $(\pi_{\mathtt{F}} \diamond \pi_{\mathtt{G}}) \diamond \mathsf{F}_{\mathtt{H}}$ is a perfect secure implementation of $\mathsf{F}_{\mathtt{F}}$ in the environments* Env*.*

It is also possible to show the following theorem.

THEOREM 4.14 *Let* Env_1 *and* Env_2 *be environment classes. Let* $\pi_F \in \mathrm{Pro}$ *be a protocol with protocol name* F *and resource name* G*. Let* π_G *be a protocol with protocol name* G *and resource name* H*. Let* F_F*,* F_G *and* F_H *be ideal functionalities with names* F*,* G *respectively* H*. Assume that if* $\mathcal{Z} \in \mathrm{Env}_1$*, then* $\mathcal{Z} \diamond \pi_F \in \mathrm{Env}_2$*. If* $\pi_F \diamond \mathsf{F}_G \overset{\mathrm{Env}_1}{\geqslant} \mathsf{F}_F$ *and* $\pi_G \diamond \mathsf{F}_H \overset{\mathrm{Env}_2}{\geqslant} \mathsf{F}_G$*, then* $(\pi_F \diamond \pi_G) \diamond \mathsf{F}_H \overset{\mathrm{Env}_1}{\geqslant} \mathsf{F}_F$*.*

Intuitively $\pi_{\mathtt{G}} \diamond \mathsf{F}_{\mathtt{H}} \overset{\mathrm{Env}_2}{\geqslant} \mathsf{F}_{\mathtt{G}}$ say that it is secure to use $\pi_{\mathtt{G}} \diamond \mathsf{F}_{\mathtt{H}}$ instead of $\mathsf{F}_{\mathtt{G}}$ in environments from the class Env_2. And the assumption that if $\mathcal{Z} \in \mathrm{Env}_1$, then $\mathcal{Z} \diamond \pi_{\mathtt{F}} \in \mathrm{Env}_2$ says that if $\pi_{\mathtt{F}}$ is run in an environment from the class Env_1, then it provides and environment from the class Env_2 for its resource. So, replacing $\mathsf{F}_{\mathtt{G}}$ by $\pi_{\mathtt{G}} \diamond \mathsf{F}_{\mathtt{H}}$ inside $\pi_{\mathtt{F}}$ is secure as long as $\pi_{\mathtt{F}}$ is run in an environment from the class Env_1.

EXERCISE 4.2 Give a formal proof of Theorem 4.14 along the lines of the proof of Theorem 4.12.

Another, important, generalization is that we can consider protocols π which use more than one communication resource. In that case π has one set of protocol ports, named like the ideal functionality F it tries to implement, but it has a separate set of resource ports for each of the ideal functionalities $\mathsf{F}_{\mathtt{R,1}}, \dots, \mathsf{F}_{\mathtt{R,N}}$ which it uses as communication resources. It is still secure to replace some resource by a secure implementation of the resource. I.e., $\pi \diamond \mathsf{F}_{\mathtt{R,1}} \diamond \cdots \diamond \mathsf{F}_{\mathtt{R,N}} \overset{\mathrm{Env}}{\geqslant} \mathsf{F}$ and $\pi_{\mathtt{R,I}} \overset{\mathrm{Env}}{\geqslant} \mathsf{F}_{\mathtt{R,I}}$ implies that $\pi \diamond \mathsf{F}_{\mathtt{R,1}} \diamond \cdots \diamond \mathsf{F}_{\mathtt{R,I-1}} \diamond \pi_{\mathtt{R,I}} \diamond \mathsf{F}_{\mathtt{R,I+1}} \diamond \cdots \diamond \mathsf{F}_{\mathtt{R,N}} \overset{\mathrm{Env}}{\geqslant} \mathsf{F}$. Using this N times, we can replace all the resources by secure implementations.

For the above to hold some changes has to be made to the definition of Env, as $\mathcal{Z}_2$ in the proof will be $\pi \diamond \mathsf{F}_{\mathtt{R,1}} \diamond \cdots \diamond \mathsf{F}_{\mathtt{R,I-1}} \diamond \mathsf{F}_{\mathtt{R,I+1}} \diamond \cdots \diamond \mathsf{F}_{\mathtt{R,N}} \diamond \mathcal{Z}$. We therefore need that this $\mathcal{Z}_2$ is in Env. If Env is the set of all recursive poly-time environments and each $\mathsf{F}_{\mathtt{R,J}}$ for $J = 1, \dots, n$, $J \neq I$ are recursive poly-time, then it can be seen that Env is an environment class under this definition.

EXERCISE 4.3 Prove the UC Theorem for protocols using several resources, and make the needed modifications to the definitions involved if you have to. Argue that all the changes are needed.

4.3 Adversaries and Their Powers

In secure multiparty computation it is often impossible to prove security in all environments. We therefore work with a large number of restricted classes of environments. Below we list some of these restrictions, discuss why they have been considered, and model them using an environment class. In doing that we always let Env be the environment class consisting of all interactive systems, and then discuss how to restrict it.

4.3.1 Threshold Security

Our protocol Protocol CEPS (Circuit Evaluation with Passive Security) from Chapter 3 is only secure against $t < n/2$ corrupted parties, as $n/2$ parties have enough shares in all secret sharings to allow them to reconstruct. This is called threshold security, and in this case the threshold is $\lfloor n/2 - 1 \rfloor$. For any t we let Env^t be the set of $\mathcal{Z} \in \text{Env}$ which corrupts at most t parties. To prove that this is an environment class, the crucial observation is that if $\mathcal{Z} \in \text{Env}^t$ and $\mathcal{S} \in \text{Sim}$, then also $\mathcal{Z} \diamond \mathcal{S}$ corrupts at most t parties, as $\mathcal{S}$ is corruption preserving.

4.3.2 Adaptive Security versus Static Security

Environments in Env are allowed to corrupt parties when they desire. This is called adaptive corruption and the environment is called an adaptive adversary, as it can adapt its corruption pattern to the communication that it observes in the protocol. Protocols which can be proven secure against adaptive adversaries are called adaptively secure.

Sometimes adaptive corruption makes security proofs hard or impossible. We therefore also work with the notion of a static adversary which must specify which parties it is going to corrupt *before the protocol execution starts.* This is called static corruption and a protocol which is only proven secure against static adversaries are called statically secure.

Recall that when we proved Protocol CEPS (Circuit Evaluation with Passive Security) secure we gave the set of corrupted parties as input to the simulator S, as we gave it $\{x_i, y_i\}_{\mathsf{P}_i \in C}$. So what we did in Chapter 3 was actually a static simulation. It turns out that Protocol CEPS (Circuit Evaluation with Passive Security) is also adaptively secure, we just did not want to go into the details of this in Chapter 3. However, as we will see in a later example, there exist protocols which are statically but not adaptively secure. For now we concentrate on modeling the notion of a static adversary.

Technically, what we want is that the simulator knows which parties are going to be corrupted before the simulation starts. We ensure this by restricting Env to the set of $\mathcal{Z} \in \text{Env}$ which behave as follows. Before $\mathcal{Z}$ sends any other messages, it does a corruption of some subset C of the parties, either passively or actively. This is called the preamble. In the simulation, $\mathcal{S}$ sees the initial corruptions done by $\mathcal{Z}$, and can therefore learn the set C before it has to simulate any leakage or influence. We use $\text{Env}^{\mathsf{static}}$ to denote this set of environments. To prove that this is an environment class, the crucial observation is that if $\mathcal{Z} \in \text{Env}^{\mathsf{static}}$ and $\mathcal{S} \in \text{Sim}$, then also $\mathcal{Z} \diamond \mathcal{S}$ does static corruptions, as $\mathcal{S}$ is corruption preserving.

Example 4.9 To demonstrate the difference between adaptive corruption and static corruption we return to our secure transfer example, Example 4.7, and consider how we simulate corruption. Assume first that we allow one static, active corruption. This means that at most one party gets corrupted, and that the simulator $\mathcal{S}$ is told which party it is before the protocol is run.

Assume first that it is P_1 who is corrupted. Technically, this means that before it does anything else, the environment $\mathcal{Z}$ outputs ($\texttt{active corrupt}$) on $\texttt{AT.infl}_1$ and then outputs $(\texttt{active corrupt}, 1)$ on $\texttt{AT.infl}$. Now, since $\mathcal{S}$ is corruption preserving it will then output $(\texttt{active corrupt}, 1)$ on $\texttt{ST.infl}$. More importantly, $\mathcal{S}$ is now allowed to send $(\texttt{send}, 1, (mid, 2, m'))$ on $\texttt{ST.infl}$ at any point, in response to which $\mathsf{F}_{\texttt{ST}}$ stores $(mid, 1, 2, m')$ as if $(mid, 2, m')$ had arrived on $\texttt{ST.in}_1$. Then $\mathcal{S}$ can send $(\texttt{deliver}, mid, 1, 2)$ on $\texttt{ST.infl}$, in response to which $\mathsf{F}_{\texttt{ST}}$ outputs $(mid, 1, m')$ on $\texttt{ST.out}_2$. So, all in all, $\mathcal{S}$ can make $\mathsf{F}_{\texttt{ST}}$ output any message of the form $(mid, 1, m')$ on $\texttt{ST.out}_i$. The simulator uses this to simulate as follows: It runs a copy of $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$ internally, where it corrupts party 1. It connects the special ports of $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$ to $\mathcal{Z}$, such that $\mathcal{Z}$ is interacting with a copy of $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$ exactly as in the real world. Whenever the party P_2 in the internal copy of $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$ outputs a message of the form $(mid, 1, m')$ on $\texttt{ST.out}_2$, then $\mathcal{S}$ makes $\mathsf{F}_{\texttt{ST}}$ output $(mid, 1, m')$ on $\texttt{ST.out}_2$. As a result $\mathcal{Z}$ will receive $(mid, 1, m')$ on $\texttt{ST.out}_2$ exactly as in the real world where it interacts with $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$. It is easy to see that this gives a perfect simulation.[9]

If it is P_2 who is corrupted, then $\mathcal{S}$ simulates as follows: Whenever $\mathcal{S}$ is activated it checks if it received some $(mid, 1, 2, |m|)$ on $\texttt{ST.leak}$. If so, it inputs $(\texttt{deliver}, mid, 1, 2)$ on $\texttt{ST.infl}$. In response to this $\mathsf{F}_{\texttt{ST}}$ outputs $(mid, 1, m)$ on $\texttt{ST.infl}$, as outputs of corrupted parties are redirected to the simulator. This means that as soon as $\mathcal{Z}$ sends $(mid, 2, m)$ to $\mathsf{F}_{\texttt{ST}}$ on $\texttt{ST.in}_1$, the simulator can learn m. This again makes the simulation trivial, as $\mathcal{S}$ now knows the input, so it can just run the protocol on m and connect the special ports of the protocol to $\mathcal{Z}$. △

The above example shows that $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$ securely implements $\mathsf{F}_{\texttt{ST}}$ in poly-time environments doing one active corruption. It is trivial to give a simulator for environments that do a static corruption of both P_1 and P_2, as the simulator then sees all inputs of P_1 and P_2 and determines which outputs $\mathsf{F}_{\texttt{ST}}$ gives on behalf of P_1 and P_2, as in a complete break down. So,

$$\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}} \overset{\mathrm{Env}^{\texttt{poly,static}}}{\geqslant} \mathsf{F}_{\texttt{ST}} \ .$$

To demonstrate that there is a difference between static and adaptive security we now argue that $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$ does not implement $\mathsf{F}_{\texttt{ST}}$ even against just one adaptive corruption, at least if the encryption scheme is a normal encryption scheme like RSA. Assume namely that both parties are honest from the beginning of the protocol and that $\mathcal{Z}$ sends a uniformly random bit string $m \in_{\mathrm{R}} \{0, 1\}^k$ and finishes the protocol, and then corrupts P_1. When playing with $\pi_{\texttt{ST}} \diamond \mathsf{F}_{\texttt{AT}}$, it will

[9] The argument here was easy because we only consider two parties. If we look at the larger protocol for n parties, other things could go wrong. Suppose, for instance, that $\mathcal{Z}$ could make P_2 output a message of the form $(mid, 3, m')$ in a setting where P_3 is honest and where $(mid, 2, m')$ was not input on $\texttt{ST.in}_3$, that is, make P_2 think it received a message from P_3 even though P_3 did not send it. Then $\mathcal{S}$ cannot simulate, as it cannot make $\mathsf{F}_{\texttt{ST}}$ output a message of the form $(mid, 3, m)$ on $\texttt{ST.out}_2$ unless $\mathsf{F}_{\texttt{ST}}$ received $(mid, 2, m)$ on $\texttt{ST.in}_3$ or P_3 is corrupted. It is, however, easy to see that $\mathcal{Z}$ cannot create such a situation, as the protocol uses authenticated transfer as resource.

see pk and $c = E_{pk}(m;r)$ during the execution of the protocol, and when it corrupts P_1 it receives the internal state of P_1, which includes m and r. Then it checks if $c = E_{pk}(m;r)$. If so, it outputs 1, otherwise it outputs 0. It is clear that $\Pr[(\pi_{\mathsf{ST}} \diamond \mathsf{F}_{\mathsf{AT}}) \diamond \mathcal{Z} = 1] = 1$. Consider then the simulation, $\mathsf{F}_{\mathsf{ST}} \diamond \mathcal{S}$. As before $\mathcal{Z}$ sends $m \in_{\mathrm{R}} \{0,1\}^\kappa$. Since P_1 is honest during the execution, $\mathcal{S}$ will only learns $|m| = \kappa$. Then $\mathcal{S}$ must show some pk and some c to $\mathcal{Z}$. Then $\mathcal{Z}$ corrupts P_1 and $\mathcal{S}$ must simulate this by giving an internal state of P_1 which includes some m and r. Sending the correct m is easy, as $\mathcal{S}$ receives m from F_{ST} when P_1 is corrupted. The value r, the simulator must cook up itself. Now $\mathcal{Z}$ checks if $c = E_{pk}(m;r)$. The claim is that the probability that $c = E_{pk}(m;r)$ is at most $2^{-\kappa}$. So, $\Pr[(\mathsf{F}_{\mathsf{ST}} \diamond \mathcal{S}) \diamond \mathcal{Z} = 1] = 2^{-\kappa}$, no matter which strategy $\mathcal{S}$ is using. The reason is that as soon as $\mathcal{S}$ sends pk and c there exists at most one m' for which there exists some r' such that $c = E_{pk}(m';r')$, as an ciphertext c can be an encryption of at most one plaintext. When $\mathcal{S}$ sent (pk, c) it did not know m, only $|m|$. So, the m' which is fixed by (pk, c) is independent of the uniformly random m sent by $\mathcal{Z}$, hence the probability that $m = m'$ is exactly $2^{-\kappa}$, and if $m \neq m'$, then $\mathcal{S}$ cannot produce r such that $c = E_{pk}(m;r)$, even if it had infinite computing power — such an r simply does not exist. If $\mathcal{Z}$ does an adaptive corruption of P_2 instead of P_1, the situation gets even worse, as the simulator would also have to show the secret key to $\mathcal{Z}$.

Non-Committing Encryption

The basic problem demonstrated above is that as soon as $\mathcal{S}$ shows the simulated pk and c to $\mathcal{Z}$, it is committed to the m' inside c and hence it gets caught if m' is not the message that $\mathcal{Z}$ gave to F_{ST}. There actually exist encryption schemes which avoid this problem, called non-committing encryption schemes. A non-committing encryption scheme is an encryption scheme which in normal mode works as a normal encryption scheme. It is, however, possible to produce a simulated public key pk and a simulated ciphertext c, which are indistinguishable from a real public key and a real ciphertext, but where it is possible to take any message m and then efficiently compute r and sk such that: r is indistinguishable from a uniformly random randomizer for the encryption scheme, sk is indistinguishable from a secret for pk and $c = E_{pk}(m;r)$ and $D_{sk}(c) = m$. Using a non-committing encryption scheme in π_{ST} will produce a protocol which is secure also against adaptive corruptions. We do, however, not know any efficient construction of non-committing encryption — all known schemes use at least κ bits to encrypt just one plaintext bit, where κ is the security parameter. Finding a non-committing encryption scheme which can encrypt κ bits using in the order of κ bits of ciphertext is an important open problem in the theory of adaptive secure multiparty computation.

4.3.3 Active Security versus Passive Security

We already saw that protocols can be secure against passive corruptions but not active corruptions, where the environment takes complete control over the corrupted party. By definition environments $\mathcal{Z} \in \text{Env}$ are allowed active corruptions. We use $\text{Env}^{\mathsf{passive}}$ to denote the set of $\mathcal{Z} \in \text{Env}$ which only does passive corruptions. That it is an environment class follows from all simulators being corruption preserving. This defines notions of a passive adversary and passively secure and active adversary and active secure. We also call a protocol which is active secure robust and we call a protocol which is passively secure private.

4.3.4 Unconditional Security versus Computational Security

Recall that our protocol Protocol CEPS (Circuit Evaluation with Passive Security) from Chapter 3 was proven perfectly secure. This means that even an environment $\mathcal{Z}$ with unbounded computing power cannot distinguish the protocol from the simulation. When we use cryptography in a protocol, as in π_{ST}, then we can, however, only expect to prove security against poly-time environments, as stronger environments, e.g., can distinguish encryptions of different messages. When security can be proven against a computationally unbounded adversary, we talk about unconditional security. When security can only be proven against a poly-time adversary, we talk about computational security.

We use $\text{Env}^{\mathsf{poly}}$ to denote the set of recursive poly-time environments. We already saw that this is an environment class.

If it is possible to prove security against unbounded environments and it in addition holds for all $\mathcal{Z}$ that

$$\pi_{\mathsf{F}} \diamond \mathsf{F}_{\mathsf{R}} \diamond \mathcal{Z} \overset{\text{perf}}{\equiv} \mathsf{F}_{\mathsf{F}} \diamond \mathcal{S} \diamond \mathcal{Z}$$

instead of just

$$\pi_{\mathsf{F}} \diamond \mathsf{F}_{\mathsf{R}} \diamond \mathcal{Z} \overset{\text{stat}}{\equiv} \mathsf{F}_{\mathsf{F}} \diamond \mathcal{S} \diamond \mathcal{Z} \ ,$$

then we talk about perfect security.

4.3.5 Synchronous versus Asynchronous

In our framework, we have chosen to clock the parties via the influence ports, which means that it is the environment which determines in which order the parties are clocked. The environment can therefore either choose to clock the parties an equal number of times, modeling that they have synchronized clocks, or it can choose to clock parties a different number of times, modeling that the parties do not have synchronized clocks. This, in particular, means that we have to model clock synchronization simply as a restriction on the environment. Another approach could have been to add clock synchronization as an ideal functionality $\mathsf{F}_{\mathtt{CLOCK\text{-}SYNC}}$. This would have allowed us to talk about securely realizing clock synchronization, by giving a protocol securely realizing $\mathsf{F}_{\mathtt{CLOCK\text{-}SYNC}}$. One disadvantage of our approach of letting clock synchronization being a restriction on the environment is that we cannot talk about how to securely realize it, at least not via the UC Theorem. An advantage of our approach is that it is technically much simpler. Specifying $\mathsf{F}_{\mathtt{CLOCK\text{-}SYNC}}$ in a usable and implementable way is technically very challenging. Since the topic of securely realizing clock synchronization is not a topic of this book, we have chosen the simpler approach.

When talking about synchronous computation we talk in the terms of rounds. Each round will actually consist of two clockings of each party. The first clocking, called inwards clocking, allows the party to send its messages to the ideal functionality for that round. The second clocking, called outwards clocking, allows the party to receive its messages from the ideal functionality for that round.

We explain inwards clocking and outwards clocking using an example. Consider the composed party $\mathcal{P}_1 = \{\mathsf{Q}_1, \mathsf{P}_1\}$ in Fig. 4.5 using the resource $\mathsf{F}_{\mathtt{AT}}$.

When we say that $\mathcal{P}_1$ is inwards clocked, we mean that $\mathcal{Z}$ first clocks Q_1 and then P_1 and then $\mathcal{Z}$ sends $(\mathtt{inclock}, i)$ on $\mathtt{AT.infl}$ and clocks $\mathsf{F}_{\mathtt{AT}}$. Note that this allows $\mathcal{P}_1$ to deliver a message to $\mathsf{F}_{\mathtt{AT}}$ and allows $\mathsf{F}_{\mathtt{AT}}$ to process it.

When we say that $\mathcal{P}_1$ is outwards clocked, we mean that $\mathcal{Z}$ first sends $(\mathtt{outclock}, i)$ on $\mathtt{AT.infl}$, then clocks $\mathsf{F}_{\mathtt{AT}}$, then clocks P_1 and then clocks Q_1. Note that this allows $\mathsf{F}_{\mathtt{AT}}$ to deliver a message to $\mathcal{P}_1$.

We say that $\mathcal{Z}$ is a synchronous environment if it proceeds in rounds, where in each round it behaves as follows:

1. First it does an inwards clocking of all parties which are not actively corrupted. It is up to $\mathcal{Z}$ in which order the parties are inwards clocked.
2. Then it possibly interacts with $\mathsf{F}_{\mathtt{AT}}$ by sending messages on $\mathtt{AT.infl}$, clocking $\mathsf{F}_{\mathtt{AT}}$ and looking at the messages output on $\mathtt{AT.leak}$.
3. Then it does an outwards clocking of all parties which are not actively corrupted. It is up to $\mathcal{Z}$ in which order the parties are outwards clocked.

In each round we call the phase from the point at which the first party P_i which is not actively corrupted is clocked in until the point at which the last party P_i which is not actively corrupted is clocked in the clock-in phase. We call the phase from the point at which the last party P_i which is not actively corrupted is clocked in until the point at which the first party P_i which is not actively corrupted is clocked out the negotiation phase. We call the phase from the point at which the first party P_i which is not actively corrupted is clocked out until the point at which the last party P_i which is not actively corrupted is clocked out the clock-out phase. We call the phase from the point in round r at which the last party P_i which is not actively corrupted is clocked out until the point in round $r + 1$ at which the first party P_i which is not actively corrupted is clocked in the transition phase. We say that the transition phase belongs to round r.

The environment $\mathcal{Z}$ is allowed to do corruptions in all phases. Notice that a phase can end by a corruption. Assume, e.g., that all parties which are not actively corrupted have been clocked in, except P_1. If then P_1 is actively corrupted by $\mathcal{Z}$ it now holds that *all* parties which are not actively corrupted have been clocked in, so the clock-in phase ended and the negotiation phase started.

We use $\mathrm{Env}^{\mathsf{sync}}$ to denote the class of synchronous environments. It is not hard to see that if $\mathcal{Z} \in \mathrm{Env}^{\mathsf{sync}}$ and $\pi \in \mathrm{Pro}$ and $\pi \diamond \mathcal{Z}$ connects the protocols ports of the two systems, then $\pi \diamond \mathcal{Z} \in \mathrm{Env}^{\mathsf{sync}}$. It is also true that if $\mathcal{Z} \in \mathrm{Env}^{\mathsf{sync}}$

and $\mathcal{S} \in \text{Sim}$ and $\mathcal{S} \diamond \mathcal{Z}$ connects the special ports of the two systems, then $\mathcal{S} \diamond \mathcal{Z} \in \text{Env}^{\mathsf{sync}}$. This follows from $\mathcal{S}$ being clock preserving.

Synchronous Communication

If n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ are using $\mathsf{F}_{\texttt{AT}}$ as communication resource, then even though the parties are synchronized, the communication is not. It might happen that $\mathcal{P}_1$ sends a message to $\mathcal{P}_2$ in round r during its inwards clocking and still $\mathcal{P}_2$ does not receive it in round r during its outwards clocking. This is because we formulated $\mathsf{F}_{\texttt{AT}}$ in an inherently asynchronous manner, which made it easier to talk about how to securely implement it, as we did not have to deal with synchronization issues. If we also want to guarantee that messages sent in round r are received in round r, then we talk about synchronous communication, which is technically different from clock synchronization. In fact, implementing synchronous communication securely require that you both have good clock synchronization and that you have an upper bound on message delivery time on the network, such that you can use timeouts. Again, we are not going to talk about how to implement synchronous communication, we are simply going to assume it when we need it. It is simple to turn $\mathsf{F}_{\texttt{AT}}$ into an ideal functionality for synchronous communication by adding the following two rules:

1. On $(\texttt{inclock}, i)$ process all messages on $\texttt{AT.in}_i$ as $\mathsf{F}_{\texttt{AT}}$ does.
2. On $(\texttt{outclock}, i)$, for all stored values (mid, j, i, m), delete (mid, j, i, m) and output (mid, j, m) on $\texttt{AT.out}_i$.

The first rule ensures that when $\mathcal{P}_i$ is inwards clocked, all the messages (mid, j, m) sent in that round will be processed and hence (mid, i, j, m) will be stored. The second rule ensures that when $\mathcal{P}_i$ is outwards clocked, all messages sent to it in that round will be delivered first, such that they are available to $\mathcal{P}_i$. In a synchronous environment, this implies that all honest parties get to deliver all their messages to all parties in all rounds.

When we talk about a synchronous protocol, we mean a protocol using an ideal functionality for synchronous communication run in an synchronous environment, i.e., a synchronous protocol uses both clock synchronization and synchronous communication.

In this book, we will mainly work with synchronous protocols. Assuming synchronous communication is not a very good model of reality, but we allow ourselves to assume it because the problems we look at are hard enough already in this model, and because most of the techniques of multiparty computation are easier to present in the synchronous model.

As an example of an advantage of assuming synchronous communication, assume that we want to tolerate that up to t of the n parties are corrupted and assume that we only have asynchronous communication. Since t parties can be corrupted it is easy to see that if an honest party at some point in the protocol waits for messages from more than $n - t$ parties, then it might potentially be waiting for a message from a corrupted party. If this corrupted party is Byzantine, it might not have sent its message at all, and since no upper bound on message delivery is guaranteed, an unsent message cannot be distinguished from a slow message.[10] The party might therefore end up waiting forever for the unsent message, and the protocol deadlocks. So, in an asynchronous protocol which must tolerate t corruptions and must be dead-lock free, the honest parties cannot wait for messages from more than $n - t$ parties in each round. But this means that *some of the honest parties might not even be able to send their inputs to the other honest parties*, let alone having their inputs securely contribute to the result. Therefore any asynchronous protocol allows input deprivation in the sense that in some executions up to t of the honest parties might not be able to make the result depend on their inputs. In some application, like voting, this is of course intolerable.

4.3.6 Consensus Broadcast versus Point-to-Point Communication

For active adversaries, there is a problem with broadcasting, namely if a protocol requires a player to broadcast a message to everyone, it does not suffice to just ask him to send the same message to all players. If he is corrupt, he may say different things to different players, and it may not be clear to the honest players if he did this or not. And, in some protocols, if a party sends different messages to different parties when it should not, it can make the protocol give incorrect results or make it fail in such a way that the adversary learns more than it should. Hence we sometimes need a mechanism which forces a party to send the same message to all parties. This is known as consensus broadcast or Byzantine agreement.[11]

In a consensus broadcast, all honest receivers are guaranteed to receive the same message *even if the sender and some of the other parties are corrupted.* This problem is so hard that sometimes it is impossible to implement. One therefore in general has to make a distinction between the case where a consensus broadcast channel is given for free as a part of the model, or whether such a channel has to be simulated by a sub-protocol. We return to this issue in more detail later, and look at an implementation of consensus broadcast.

What is important here is that consensus broadcast is not hard to model as an ideal functionality. We therefore do not model it as an environment class. We mention the problem in this section only because it fits into the discussion of the powers of adversaries.

4.3.7 Combining Environment Classes

We can combine the above notions, and, e.g., talk about a synchronous, poly-time bounded, t-threshold adversary only doing static (and active) corruptions. The

[10] In fact, it is easy to see that the ability to distinguish an unsent message from a sent-but-slow message requires that you know an upper bound on delivery time.

[11] Sometimes it is also just known by broadcast, but in the distributed computing literature the term *broadcast* is sometimes used to refer to a communication mechanisms which does not necessarily guarantee consistency if the sender is corrupted. We want to avoid this possible confusion, and therefore use the term *consensus broadcast.*

corresponding environment class is

$$\mathrm{Env}^{\mathtt{sync},\mathtt{poly},t,\mathtt{static}} \overset{\mathrm{def}}{=} \mathrm{Env}^{\mathtt{sync}} \cap \mathrm{Env}^{\mathtt{poly}} \cap \mathrm{Env}^{t} \cap \mathrm{Env}^{\mathtt{static}} \ .$$

Since the intersection of environment classes produces an environment class, we have that $\mathrm{Env}^{\mathtt{sync},\mathtt{poly},t,\mathtt{static}}$ is an environment class.

4.4 Some Ideal Functionalities

In this section we describe some standard ideal functionalities that we will use in the following.

4.4.1 Complete Break Down

Before describing the functionalities, it is useful to introduce the notion of a complete break down. When we say that an ideal functionality $\mathsf{F}_{\mathtt{F}}$ does a complete break down, then we mean that it starts behaving as if all parties were actively corrupted. More specifically it starts ignoring all its other code and only executes the following rules:

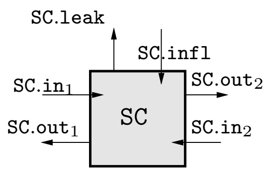
- It first outputs the ideal internal state of *all* parties on `F.leak`.
- On all subsequent inputs $(\mathtt{send}, i, m)$ on `F.infl`, it outputs m on $\mathtt{F.out}_i$.
- On all subsequent inputs m on $\mathtt{F.in}_i$, it outputs (i, m) on `F.leak`.

Notice that if $\mathsf{F}_{\mathtt{F}}$ is used as a communication resource and does a complete break down, then the adversary sees all inputs given to $\mathsf{F}_{\mathtt{F}}$ and it is the adversary who specifies all outputs delivered by $\mathsf{F}_{\mathtt{F}}$. This is an ultimately useless and insecure communication resource. On the other hand, if $\mathsf{F}_{\mathtt{F}}$ is acting as ideal functionality and does a complete break down, then the simulator sees all inputs given to $\mathsf{F}_{\mathtt{F}}$ and it is the simulator $\mathcal{S}$ who specifies all outputs delivered by $\mathsf{F}_{\mathtt{F}}$. This makes simulation trivial. Assume namely that $\mathcal{S}$ is trying to simulate a protocol $\pi_{\mathtt{F}} \diamond \mathcal{R}$. The simulator $\mathcal{S}$ will simply run a copy of $\pi_{\mathtt{F}} \diamond \mathcal{R}$ internally and connect the special ports of $\pi_{\mathtt{F}} \diamond \mathcal{R}$ to the environment $\mathcal{Z}$. Whenever $\mathcal{Z}$ inputs m on some inport $\mathtt{F.in}_i$ of $\mathsf{F}_{\mathtt{F}}$, then $\mathcal{S}$ is given (i, m). In response to this, $\mathcal{S}$ inputs m on the port $\mathtt{F.in}_i$ of its internal copy of $\pi_{\mathtt{F}} \diamond \mathcal{R}$. And, whenever the copy of $\pi_{\mathtt{F}} \diamond \mathcal{R}$ that $\mathcal{S}$ is running outputs a value m on some port $\mathtt{F.out}_i$, then $\mathcal{S}$ inputs $(\mathtt{send}, i, m)$ to $\mathsf{F}_{\mathtt{F}}$, in response to which $\mathsf{F}_{\mathtt{F}}$ sends m to $\mathcal{Z}$ on $\mathtt{F.out}_i$. This means that $\mathcal{Z}$ is essentially just playing with the copy of $\pi_{\mathtt{F}} \diamond \mathcal{R}$ run by $\mathcal{S}$, so clearly $\mathcal{Z}$ cannot distinguish this from a situation where it is playing with $\pi_{\mathtt{F}} \diamond \mathcal{R}$. In fact, $\pi_{\mathtt{F}} \diamond \mathcal{R}$ and $\mathsf{F}_{\mathtt{F}} \diamond \mathcal{S}$ will be perfectly indistinguishable to any $\mathcal{Z}$.

With the above discussion in mind, we can think of a complete break down of an ideal functionality as a technical way to say that from the point of the complete break down, we require no security properties of an implementation of $\mathsf{F}_{\mathtt{F}}$.

Agent $\mathrm{F}_{\texttt{SC}}$

The ideal functionality for secure communication and consensus broadcast between n parties (pictures shows only 2 for simplicity).



initialization The ideal functionality keeps track of three sets, A, P and C, which are all initially empty. When party i is actively corrupted it adds i to A and C. When party i is passively corrupted it adds i to P and C.

honest inputs On input $(\texttt{clockin}, i)$ on $\texttt{SC.infl}$, read a message from $\texttt{SC.in}_i$. If there was a message on $\texttt{SC.in}_i$, then parse it on the form $(m_{i,1}, \dots, m_{i,n}, m_i)$. Here $m_{i,j}$ is the message that P_i sends to P_j and m_i is the message P_i is broadcasting. It then outputs $(\{m_{i,j}\}_{j \in C}, m_i)$ on $\texttt{SC.leak}$ (this means that as soon as a honest party sends messages, the adversary learns the messages intended for the corrupted parties). Finally it stores $(m_{i,1}, \dots, m_{i,n}, m_i)$. Furthermore, if at some point during the round the set C grows, then output $(i, j, m_{i,j})$ on $\texttt{SC.leak}$ for the new $j \in C$.

corrupted inputs On input $(\texttt{change}, i, (m_{i,1}, \dots, m_{i,n}, m_i))$ on $\texttt{SC.infl}$, where $i \in A$ and at a point before the clock-out phase started, store $(m_{i,1}, \dots, m_{i,n}, m_i)$, overriding any previous input from the actively corrupted party i in this round (this means that as long as no passively corrupted party or honest party received outputs, the corrupted parties are allowed to change their inputs).

delivery On input $(\texttt{clockout}, i)$ on $\texttt{SC.infl}$, if there is a party P_i for which no $(m_{i,1}, \dots, m_{i,n}, m_i)$ is stored, then store $(m_{i,1}, \dots, m_{i,n}, m_i) = (\epsilon, \dots, \epsilon)$ for all such parties. Then output $(m_{1,i}, \dots, m_{n,i}, (m_1, \dots, m_n))$ on $\texttt{SC.out}_i$.

4.4.2 Secure Synchronous Communication and Broadcast

In Agent $\mathrm{F}_{\texttt{SC}}$ we give an ideal functionality for secure *syncronous* communication and *consensus* broadcast. In each round each party P_i specifies one message $m_{i,j}$ for each of the other parties P_j plus a message m_i that P_i wants to broadcast. At the end of the round each P_j receives the messages $m_{i,j}$ and m_i from each P_i. The message m_i output to different parties P_j and P_k are guaranteed to be the same. This means that all messages are guaranteed to be delivered and that there is no way to send different broadcast messages to different parties.

Notice that we allow the corrupted parties to see their messages first and we allow them to change their mind on their own messages until the round is over. This allows them to choose their own messages based on what the honest parties are sending. This is called rushing. If we do not allow rushing, then it is very hard and sometimes impossible to securely implement $\mathsf{F}_{\texttt{SC}}$, the reason essentially being that some party must send its messages first, and the corrupted parties can always wait a little longer than the honest parties.

Notice that if a party fails to send a message in some round, we simply set all its messages to be the empty string ϵ. This holds for both the honest parties and the corrupted parties. In terms of implementation, this just means that if you do not receive a message from some party before the timeout for a given round, you define the message to be ϵ. For an implementation to be secure it is therefore essential that the timeout is so long that any honest party who tries to deliver a message will always have it delivered before the timeout, and this should hold in the worst case and except with negligible probability. This can make a secure implementation of synchronous communication very inefficient, as each round suffer a delay which is at least as long as the worst case delivery time of the network.

4.4.3 Secure Synchronous Function Evaluation

In Agent F^f_{SFE} we give an ideal functionality for secure synchronous function evaluation. For simplicity we focus on the case where each party gives just one input, and we give an ideal functionality which can be used only once.

As for F_{SC} we allow that actively corrupted parties can change their inputs as long as the function was not evaluated, and we give all corrupted parties their outputs as soon as they are defined. This rushing behaviour makes it easier to implement $\mathsf{F}^f_{\mathsf{SFE}}$ as a protocol is allowed to have the same influence and leakage.

We allow that the adversary determines when the function is evaluated, but we require that it waits until all parties which are not actively corrupted have given their inputs, or we would implicitly have allowed the adversary to set the inputs of honest parties or passively corrupted parties to 0.

Notice that until the ideal functionality provides outputs, it sends no messages on its outports $\mathtt{SFE.out}_i$. This means that some larger protocol using $\mathsf{F}^f_{\mathsf{SFE}}$ will just observe silence from $\mathsf{F}^f_{\mathsf{SFE}}$ during most of the rounds.

Simultaneous Input

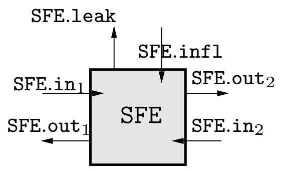
Notice also that we implicitly require that all honest parties provide their inputs in the same round. Namely, if they don't do this, we make $\mathsf{F}^f_{\mathsf{SFE}}$ do a complete break down. This allows a secure implementation of $\mathsf{F}^f_{\mathsf{SFE}}$ to do the same. In other words, we do not require any security properties from an implementation if the parties in the protocol do not get inputs in the same round. This makes implementation much easier, as the implementation can just assume that all parties get inputs from the environment in the same round, and ignore what happens in case this is not true: any behavior of the protocol in case the simultaneous input assumption breaks can be trivially simulated as the ideal functionality does a complete break down.

Assuming that all parties which are not actively corrupted get their inputs in the same round is called simultaneous input. The reason why we do not require security of a protocol when there is non-simultaneous input is that it is very hard to securely handle this, and sometimes impossible.

As an example of this, consider the protocol Protocol CEPS (CIRCUIT EVALUATION WITH PASSIVE SECURITY). Assume that honest parties only start doing

Agent $F^f_{\mathtt{SFE}}$

The ideal functionality for one secure function evaluation between n parties of a function $f : \mathbb{F}^n \to \mathbb{F}^n$ for a finite field $\mathbb{F}$ (picture shows only 2 parties for simplicity).



initialize The ideal functionality keeps track of three sets, A, P and C as F_{SC}. It also keeps bits `delivery-round`, `evaluated`, `inputs-ready`, $\texttt{input-ready}_1, \ldots, \texttt{input-ready}_n \in \{0, 1\}$, initially set to 0.

honest inputs On input $(\texttt{clockin}, i)$ on `SFE.infl` for $i \notin A$, read a message from $\texttt{SFE.in}_i$. If there was a message x_i on $\texttt{SFE.in}_i$ and $x_i \in \mathbb{F}$ and $\texttt{input-ready}_i = 0$, then set $\texttt{input-ready}_i \leftarrow 1$, store (i, x_i) and output $(\texttt{input}, i)$ on `SFE.leak`.

corrupted inputs On input $(\texttt{change}, i, x_i)$ on `SFE.infl`, where $i \in A$ and $x_i \in \mathbb{F}$ and $\texttt{evaluated} = 0$, set $\texttt{input-ready}_i \leftarrow 1$ and store (i, x_i), overriding any such previous value stored for party i (this means that as long as the function has not been evaluated on the inputs, the corrupted parties are allowed to change their inputs).

simultaneous inputs If it holds in some round that after the clock-in phase ends there exist $i, j \notin A$ such that $\texttt{input-ready}_i = 0$ and $\texttt{input-ready}_j = 1$, then do a complete break down.

If it happens in some round that after the clock-in phase ends that $\texttt{input-ready}_i = 1$ for all $i \notin A$ and $\texttt{inputs-ready} = 0$, then set $\texttt{inputs-ready} \leftarrow 1$ and for each $i \in A$ where $\texttt{input-ready}_i = 0$, store $(i, x_i) = (i, 0)$.

evaluate function On input $(\texttt{evaluate})$ on `SFE.infl` where $\texttt{inputs-ready} = 1$ and $\texttt{evaluated} = 0$, set $\texttt{evaluated} \leftarrow 1$, let $(x_1, \ldots, x_n)$ be the values defined by the stored values (i, x_i) for $i = 1, \ldots, n$ and compute $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$. Then output $\{(i, y_i)\}_{i \in C}$ on `SFE.leak`, and if C later grows, then output (j, y_j) on `SFE.leak` for the new $j \in C$.

simultaneous output On input $(\texttt{delivery-round})$ on `SFE.infl`, where $\texttt{evaluated} = 1$ and $\texttt{delivery-round} = 0$, proceed as follows: if we are at a point where not party $i \notin A$ was clocked out yet, then set $\texttt{delivery-round} \leftarrow 1$.

delivery On input $(\texttt{clockout}, i)$ on `SFE.infl`, where $\texttt{delivery-round} = 1$, output y_i on $\texttt{SFE.out}_i$.

the protocol when they get inputs from the environment, and say that for instance party P_1 gets input in round 42 and party P_2 gets input in round 1042. This means that when P_1 starts running the protocol $\pi_{\mathtt{CEPS}}$, the party P_2 will not be running the protocol yet, so P_1 will not receive shares of the input of P_2 and P_2 will not be ready to receive and store the share of the input of P_1. And it is provably impossible to do anything about this if we don't want input deprivation and if we want to tolerate active corruptions and want to guarantee that the protocol terminates. Namely, from the point of view of P_1, the party P_2 might either be an honest party which did not get its input yet or P_2 might be a corrupted party who is deviating from the protocol. Any sub-protocol which tries to resolve this must have the property that it does not start the protocol until all honest parties received their inputs. On the other hand, it should allow the honest parties to proceed even if some corrupted party claims that it is honest but just did not get its input yet. These two requirements are clearly contradictory.

Simultaneous Output

Notice that the way we handle delivery means that all parties which are not actively corrupted will receive their outputs in the *same* round. This is called simultaneous output.

Having simultaneous output is very convenient in a synchronous environment. To see why, consider a synchronous protocol $\pi_{\mathtt{N}}$ which uses $\mathsf{F}^{f}_{\mathtt{SFE}}$ as a sub routine in some larger protocol. Consider the following plausible behavior: At some point the parties Q_i call $\mathsf{F}^{f}_{\mathtt{SFE}}$. Then they wait until $\mathsf{F}^{f}_{\mathtt{SFE}}$ delivers outputs, and in that round they then continue with the outer protocol, where the next step is to invoke some other sub protocol $\pi_{\mathtt{SUB}}$. Now, if $\mathsf{F}^{f}_{\mathtt{SFE}}$ delivers outputs to different parties in different rounds, then the parties Q_i would continue with the outer protocol in different rounds. This would make them give inputs to $\pi_{\mathtt{SUB}}$ in different rounds. So, if $\pi_{\mathtt{SUB}}$ assumes simultaneous inputs, the overall protocol will now break down. Put briefly, we need simultaneous outputs to guarantee simultaneous inputs in later parts of the protocol. This means that simultaneous outputs is an important *security property.*

It is instructive to see how that property is captured by the UC framework. Let us say that $\pi_{\mathtt{N}}$ is secure when it use $\mathsf{F}^{f}_{\mathtt{SFE}}$, and that $\pi_{\mathtt{N}}$ depends on $\mathsf{F}^{f}_{\mathtt{SFE}}$ having simultaneous output. If we then replace $\mathsf{F}^{f}_{\mathtt{SFE}}$ by a protocol $\pi_{\mathtt{SFE}}$ which does not have simultaneous output, then the overall protocol would break.

Fortunately our framework would exactly catch such a bad protocol before we substitute it for $\mathsf{F}^{f}_{\mathtt{SFE}}$, as it cannot be proven to be a secure implementation of $\mathsf{F}^{f}_{\mathtt{SFE}}$. Assume namely that $\mathsf{F}^{f}_{\mathtt{SFE}}$ has simultaneous output and $\pi_{\mathtt{SFE}}$ does not, then it will be very easy to distinguish the two: No matter which simulator we attach to $\mathsf{F}^{f}_{\mathtt{SFE}}$ the system $\mathsf{F}^{f}_{\mathtt{SFE}} \diamond \mathcal{S}$ will have simultaneous output when run in a synchronous environment, as $\mathcal{S}$ is clock preserving. And we assumed that the protocol $\pi_{\mathtt{SFE}}$ does not have simultaneous output. So, $\mathcal{Z}$ just looks at whether the outputs are delivered in the same round, and outputs 1 if and only if they are. Then $\pi_{\mathtt{SFE}} \diamond \mathcal{Z}$ will output 0 and $\mathsf{F}^{f}_{\mathtt{SFE}} \diamond \mathcal{S} \diamond \mathcal{Z}$ will outout 1. The same distinguishing strategy works even if $\pi_{\mathtt{SFE}}$ fails to have simultaneous outputs with some non-negligible probability.

It might seem that simultaneous output is easy to achieve. That is true, but only if consensus broadcast is given for free by the communication model. If consensus broadcast is to be implemented using a secure protocol which only uses point-to-point communication, then it is possible to prove the following two claims:

1. Any secure realization of consensus broadcast which can tolerate up to t ac-

tively corrupted parties and which has simultaneous output uses at least $t+1$ communication rounds, even when no parties are corrupted.

2. There exist a secure realization of consensus broadcast which can tolerate up to t actively corrupted parties and which has non-simultaneous output which uses only $\min(t+1, f+2)$ communication rounds, where $f \in \{0, \ldots, t\}$ is the number of parties who were actually corrupted during the execution of the protocol.

So, if we for instance have $n = 1001$ parties and want to tolerate $t = 500$ actively corrupted parties, but in typical executions of the protocol no parties are actually deviating from the protocol, then requiring simultaneous output would make each consensus broadcast cost 501 communication rounds, whereas tolerating non-simultaneous output would typically allow to run each consensus broadcast in 2 communication rounds. If we are in a network where the worst case delivery time of a message is one minute, this would make the difference between each consensus broadcast taking two minutes or more than eight hours. I.e., with non-simultaneous output we only pay a high price when there are many parties who deviate from the protocol. If we insist on simultaneous output, then we pay a high price even if no parties deviate from the protocol.

Adversarially Chosen Output Round

Finally notice that we allow the environment to specify in which round the outputs are delivered, we call this adversarially chosen output round. First of all, the reason why we don't just compute and give the outputs in the round where the inputs are given is that we want to allow an implementation to use several rounds of communication. If outputs were delivered immediately by the ideal functionality, a secure implementation would be allowed to use only 1 round of communication: the environment can observe the number of rounds that passes between inputs and outputs, so if the ideal functionality uses 1 round and the protocol uses 2 or more rounds, then the environment could trivially distinguish. However, secure multiparty computation is impossible in one round of communication. This opens the question of which round we should then specify as the output round. If we choose round 7, then for the same reasons as above, only protocols with exactly 7 rounds would be deemed secure. Furthermore, if we pick any fixed round as the output round, then only protocols with a fixed round complexity would be secure. However, some protocols naturally have a varying output round. In fact, some of our MPC protocols will use more rounds when there is adversarial behavior, as sub-protocols must be run to deal with this adversarial behavior. We could, of course, pad the runs where there is no adversarial behaviour with some dummy rounds to ensure all executions take the same number of rounds, but introducing such inefficiency just because the model artificially asks for a fixed output round would not be desirable nor elegant. The correct solution is to note that if the adversary in the protocol has influence on which round is the output round, and we want to tolerate such influence, then we should add it to the ideal functionality as allowed influence.

EXAMPLE 4.10 We now argue that our protocol Protocol CEPS (CIRCUIT EVALUATION WITH PASSIVE SECURITY) is secure in the UC model. Let π^f_{CEPS} denote the protocol Protocol CEPS (CIRCUIT EVALUATION WITH PASSIVE SECURITY) for a function f with the addition that all parties wait with outputting their output y_i until the results y_j have been reconstructed towards all parties. Then all parties output their y_i in the same round. We show that

$$\pi^f_{\mathsf{CEPS}} \diamond \mathsf{F}_{\mathsf{SC}} \overset{\mathrm{Env}^{t,\mathtt{static},\mathtt{sync}}}{\geqslant} \mathsf{F}^f_{\mathsf{SFE}} ,$$

for all $t < n/2$.

For this purpose we construct the simulator Agent $\mathcal{S}_{\mathsf{CEPS}}$ shown in the box.

Agent $\mathcal{S}_{\mathsf{CEPS}}$

A simulator for $\pi^f_{\mathsf{CEPS}} \diamond \mathsf{F}_{\mathsf{SC}}$ for the static case where the environment gives inputs to all parties in the same round.

Initialize: In the preamble, the environment $\mathcal{Z}$ specifies the set C of passively corrupted parties. After this, the simulator sets up a copy of $\pi^f_{\mathsf{CEPS}} \diamond \mathsf{F}_{\mathsf{SC}}$ where it passively corrupts the players in C and connects the special ports of $\pi^f_{\mathsf{CEPS}} \diamond \mathsf{F}_{\mathsf{SC}}$ to $\mathcal{Z}$.

Input Sharing: After the clock-in phase in the round where all inputs are given, $\mathcal{S}_{\mathsf{CEPS}}$ does passive corruptions of $\mathsf{P}_i \in C$ on $\mathsf{F}^f_{\mathsf{SFE}}$, which then makes $\mathsf{F}^f_{\mathsf{SFE}}$ leak the corresponding inputs x_i to $\mathcal{S}_{\mathsf{CEPS}}$.

Now $\mathcal{S}_{\mathsf{CEPS}}$ does the input sharing steps of π^f_{CEPS} with the true inputs for the corrupted parties and with 0 as input for honest parties. That is, for each player P_j, $\mathcal{S}_{\mathsf{CEPS}}$ distributes $[x^{(0)}_j; f_{x^{(0)}_j}]_t$, where $x^{(0)}_j = x_j$ for $\mathsf{P}_j \in C$ and $x^{(0)}_j = 0$ for $\mathsf{P}_j \notin C$.

Computation Phase: In the computation phase the simulator simply follows the protocol steps of π^f_{CEPS} according to the gates that need to be processed.

Output Reconstruction: First $\mathcal{S}_{\mathsf{CEPS}}$ instructs $\mathsf{F}^f_{\mathsf{SFE}}$ to evaluate the function, thereby learning the outputs y_j for all $\mathsf{P}_j \in C$.

For each $\mathsf{P}_j \in C$, π^f_{CEPS} takes the $|C|$ shares that the corrupted parties hold in the output of P_j and defines $f_{y_j}(0) \overset{\text{def}}{=} y_j$. Also, for $t - |C|$ honest players P_i, it defines $f_{y_j}(i)$ to be a random value. After this, $f_{y_j}(\mathtt{X})$ is defined in $t+1$ points and hence $\mathcal{S}_{\mathsf{CEPS}}$ can use Lagrange interpolation to compute the unique degree-t polynomial consistent with these points. Then, for $\mathsf{P}_i \notin C$ it computes the share $f_{y_j}(i)$ and sends it to P_j on behalf of P_i.

When $\mathcal{Z}$ clocks out P_i in the copy of $\pi^f_{\mathsf{CEPS}} \diamond \mathsf{F}_{\mathsf{SC}}$ so that P_i should produce an output, the simulator first inputs (`delivery-round`) to $\mathsf{F}^f_{\mathsf{SFE}}$ if it has not done so already. Then $\mathcal{S}_{\mathsf{CEPS}}$ inputs (`clock-out`, i) to $\mathsf{F}^f_{\mathsf{SFE}}$ to make it output y_i on $\mathtt{SFE.out}_i$, exactly as P_i would have done in the protocol.

By inspecting $\mathcal{S}_{\mathsf{CEPS}}$ we can see that the produced views are distributed as in the real world. Actually, $\mathcal{S}_{\mathsf{CEPS}}$ is producing the exact same views as the simulator S from Section 3.3.3. (Repeating the argument, $\mathcal{S}_{\mathsf{CEPS}}$ runs the corrupt players on their true inputs, so this has the exact same distribution. During the protocol, the environment will never see more than t shares of values belonging to honest players. No information on these values is given since they are shared using random polynomials of degree t. In output reconstruction, the environment sees for each corrupted player's output y_i all the shares sent by the honest parties. But these are already given as those consistent with the output being y_i and the t shares held by the corrupted parties. If less than t players are corrupted, the simulator chooses the missing shares at random. This can indeed be done independently for each output: Recall that we assume that every output is produced by a multiplication gate and the protocol for such a gate produces a fresh random set of shares with the only constraint that they determine the correct value.) Furthermore, both $\pi^f_{\mathtt{CEPS}} \diamond \mathsf{F}_{\mathtt{SC}}$ and $\mathsf{F}^f_{\mathtt{SFE}}$ output the correct y_i, and hence *the same* y_i. Finally, $\mathcal{S}_{\mathtt{CEPS}}$ makes $\mathsf{F}^f_{\mathtt{SFE}}$ give output on $\mathtt{SFE.out}_i$ exactly when P_i would give output on $\mathtt{SFE.out}_i$. Hence, the perfect security follows.

We now describe how to modify $\mathcal{S}_{\mathtt{CEPS}}$ to handle the case where the environment does not give inputs to all parties in the same round. Again $\mathcal{S}_{\mathtt{CEPS}}$ learns the set of corrupted parties C in the preamble. If, in some round $\mathcal{Z}$ gives inputs to some parties and not others, then $\mathsf{F}^f_{\mathtt{SFE}}$ does a complete break down, so $\mathcal{S}$ learns all inputs $(x_1, \ldots, x_n)$. It then runs $\pi^f_{\mathtt{CEPS}} \diamond \mathsf{F}_{\mathtt{SC}}$ on these inputs and connects the special ports of $\pi^f_{\mathtt{CEPS}} \diamond \mathsf{F}_{\mathtt{SC}}$ to $\mathcal{Z}$. Whenever $\mathcal{Z}$ makes a party P_i output some value y'_i on $\mathtt{SFE.out}_i$, the simulator instructs $\mathsf{F}^f_{\mathtt{SFE}}$ to do the same, which it is allowed to do as $\mathsf{F}^f_{\mathtt{SFE}}$ is in a complete break down. This perfectly simulates the protocol, no matter how it behaves when there is non-simultaneous input. There is one little twist though. In the round where $\mathcal{Z}$ gives inputs x_i to some P_i and not to others, $\mathsf{F}^f_{\mathtt{SFE}}$ does not do a complete break down until there is some P_i which does not get an input, and $\mathcal{Z}$ might choose to, e.g., first give input to P_2 and then P_5 and then fail to give input to P_3 during the clock-in of P_3. In this case the simulator would have to simulate P_2 and P_5 while $\mathsf{F}^f_{\mathtt{SFE}}$ is not in complete break down. This is, however, easy. If, e.g., P_2 is corrupted, then $\mathcal{S}$ learns x_2 from $\mathsf{F}^f_{\mathtt{SFE}}$ and inputs x_2 to P_2 in $\pi^f_{\mathtt{CEPS}} \diamond \mathsf{F}_{\mathtt{SC}}$. If, e.g., P_5 is honest, then $\mathcal{S}$ does not learn x_5 from $\mathsf{F}^f_{\mathtt{SFE}}$, so it just inputs 0 to P_5 in $\pi^f_{\mathtt{CEPS}} \diamond \mathsf{F}_{\mathtt{SC}}$, which makes P_5 do a sharing of 0, and this will show at most t shares to $\mathcal{Z}$, namely those sent to P_i for $i \in C$. When $\mathcal{Z}$ then fails to give input to P_3, the complete break down will give the true value of x_5 to $\mathcal{S}$. Then $\mathcal{S}$ computes the secret sharing of x_5 which is consistent with $f(0) = x_5$ and the t shares already shown to $\mathcal{Z}$. From $f(\mathtt{X})$ it computes the shares that P_5 would have sent to the other honest parties and then it updates the internal state of P_5 and $\mathsf{F}_{\mathtt{SC}}$ to be consistent with P_5 having sent these shares — this is possible as the internal state of the honest P_5 was not shown to $\mathcal{Z}$ at this point and because $\mathsf{F}_{\mathtt{SC}}$ only leaked what P_5 sent to the corrupted parties. Now the state of $\pi^f_{\mathtt{CEPS}} \diamond \mathsf{F}_{\mathtt{SC}}$ is consistent with P_5 having run with x_5. The simulator patches the state this way for all honest parties which received inputs before the complete break down, and then $\mathcal{S}_{\mathtt{CEPS}}$ finishes the simulation as above, by simply running the protocol on the true inputs of the parties. △

Exercise 4.4 Argue that

$$\pi^f_{\mathtt{SFE}} \diamond \mathsf{F}_{\mathtt{SC}} \overset{\mathrm{Env}^{t,\mathtt{sync}}}{\geqslant} \mathsf{F}^f_{\mathtt{SFE}} \ ,$$

for all $t < n/2$. I.e., argue that the protocol is also adaptively secure. [Hint: Look at how the simulator in the example above handled the case where it first had to simulate P_5 without knowing its input and then had to patch the state of P_5 to be consistent with x_5 when the total break down occurred.]

4.5 Adaptive versus Static Security Revisited

In this section, we consider again the relation between static and adaptive security and show a general proof strategy for proving adaptive security that generalizes what we saw in the previous section, Example 4.10 and the following exercise. The idea is to first prove static security and then construct, from the simulator $\mathcal{S}$ we built, a new simulator $\mathcal{S}'$ for the adaptive case. Roughly speaking, the strategy for $\mathcal{S}'$ is to follow the algorithm of $\mathcal{S}$, but every time a new player P_i is corrupted, $\mathcal{S}'$ cooks up a view for P_i that "looks convincing", gives this to the environment, patches the state of $\mathcal{S}$ accordingly and continues.

It turns out that there is a class of unconditionally secure protocols and functionalities where this idea works and our goal will be to characterize this class and point out what the procedure run by $\mathcal{S}'$ to handle corruptions must satisfy. This will mean it becomes easier to give proofs for adaptive security later in the book. We will consider the case of perfect security first and later show that the results are also true in some cases for statistical security.

So we will assume we are given protocol π, auxiliary functionality $\mathcal{R}$, ideal functionality F, and simulator $\mathcal{S}$ such that $\pi \diamond \mathcal{R} \overset{\text{perf}}{\equiv} \mathcal{S} \diamond \mathsf{F}$ with respect to static, unbounded environments that corrupt at most t players. We will assume synchronous protocols only, for simplicity.

We will need some notation in the following:

Definition 4.15 *For an agent A that is part of an interactive system $\mathcal{IS}$, the* view *of A is a random variable, written $V_{\mathsf{A}}(\mathcal{IS})$, and is defined to be the ordered concatenation of all messages exchanged on the ports of A and of the random choices of A. We use $V_{\mathsf{A}}(\mathcal{IS}|E)$ to denote the view when conditioned on some event E occurring, and $V_{\mathsf{A}}(\mathcal{IS})_j$ to denote the view truncated to contain only the first j rounds.*

Definition 4.16 *For a player P_i in a protocol π running with auxiliary functionality $\mathcal{R}$, and environment $\mathcal{Z}$, the* conversation *of P_i is a random variable, written $\mathrm{Conv}_{\mathsf{P}_i}(\mathcal{Z} \diamond \pi \diamond \mathcal{R})$, and is defined to be the ordered concatenation of all messages P_i exchanges with honest players and $\mathcal{R}$. The conversation of $\mathcal{Z}$, written $\mathrm{Conv}_{\mathcal{Z}}(\mathcal{Z} \diamond \pi \diamond \mathcal{R})$, is the ordered concatenation of all messages exchanged on the special ports of $\mathcal{Z}$. For conversations, we denote truncation and conditioning on events in the same way as for views.*

Note that the conversation of a party is a substring of its view. Also note that when a player P_i is corrupted, its view becomes a substring of the conversation of $\mathcal{Z}$, because $\mathcal{Z}$ learns the entire view of P_i up to the corruption and then decides (or sees) all messages P_i sends and recieves. We may think of $\mathrm{Conv}_{\mathcal{Z}}(\mathcal{Z} \diamond \pi \diamond \mathcal{R})$ as the total information $\mathcal{Z}$ gets from attacking the protocol. Recall, however that $\mathcal{Z}$ also learns the inputs and outputs of the honest players.

We will need to assume that the ideal functionality has a certain behavior. First it must ensure that whenever it receives input or gives output, the time at which this happens is publicly known, i.e., it is indicated on the leakage port of the ideal functionality which party just received an input. The second demand is meant to capture the idea that the functionality should treat players who are corrupt but behave honestly in the same way as if they were honest. The formulation may look strange at first sight, we give an intuitive explanation after the definition.

DEFINITION 4.17 *The ideal functionality F is said to be input-based if the following is satisfied:*

Honest behavior equivalence: *Consider executions of F where some set A is corrupted from the start and where a fixed (ordered) set of inputs I_{F} are given to F during the execution*[12].

In any such execution the outputs produced by F and its state at the end has the same distribution, in particular the distribution does not depend on the corruptions that occur during the execution[13].

Publicly known input/output provision: *Each time F receives an input from P_i, F sends a message on its leakage port specifying that some input from P_i has been received. Each time F sends a private output to P_i, it also leaks a message, specifying that an output was given, but not the value.*

The "Honest Behavior Equivalence" condition can be intuitively explained as follows: an ideal functionality knows which players are corrupt and its actions may in general depend arbitrarily on this information. The condition puts a limitation on this: Consider first an execution where all players outside A remain honest and F gets I_{F} as input. Compare this to a case where $\mathsf{P}_i \notin A$ is corrupted, but F still gets the same inputs. This means in particular that P_i sends the same inputs, so he "behaves honestly" towards F. Therefore, the demand we make loosely speaking means that as long as a corrupt player behaves honestly towards the functionality, the actions it takes will be the same as if that player had been honest.

Our results will also be valid for a slightly more general case where the outputs produced by F do not have to be the same in all executions, but the outputs in one execution can be efficiently computed from I_{F} and the outputs in any other execution. For simplicity we do not treat this general case explicitly in the following.

[12] Notice that these inputs may arrive on the protocol port or on the influence port, depending on which players happen to be corrupted in the given execution.

[13] Note that technically, F must keep track of which parties are corrupted, to know which can be controlled via the leakage port, so the state cannot be exactly the same in all cases. However, we require that up to the fact the different sets of corrupted players are stored, the state is exactly the same.

We also need to make some assumptions on how the simulator $\mathcal{S}$ behaves, more precisely on how it decides on the inputs it sends to F on behalf of corrupted players (recall that once a player is corrupted, the simulator gets to decide which inputs this player provides to F). We will assume that $\mathcal{S}$ uses a standard technique to decide on these inputs, namely at the time where the input is given, it looks at the conversation of the corrupt player and decides on the input based on this. This is formalized as follows:

Definition 4.18 *The simulator $\mathcal{S}$ is conversation-based if the following is satisfied:*

Conversation-based inputs: *If $\mathcal{S}$ sends an input x to F on behalf of P_i in round j, it computes x as $x = \mathrm{Inp}_i(c)$ where Inp_i is a function specified by the protocol and $c = \mathrm{Conv}_{\mathsf{P}_i}(\mathcal{Z} \diamond \mathcal{S} \diamond \mathsf{F})_j$.*

Honest behavior implies correct inputs: *If P_i is corrupt but has followed the protocol honestly then it is always the case that $\mathrm{Inp}_i(c_i)$ equals the corresponding input P_i was given from the environment. By a corrupt P_i following the protocol honestly, we mean that the environment decides the actions of P_i by running a copy of the code of the honest P_i on the inputs and messages that P_i receives in the protocol*[14].

Corruption-consistent input functions: *Consider a conversation c of P_i. Form a new conversation c' by deleting the messages exchanged with some of the honest players, but such that at least $n - t$ honest players remain. For all such c, c' and all input functions, it must be the case that $\mathrm{Inp}_i(c) = \mathrm{Inp}_i(c')$.*

Note that input functions only have to be defined on conversations that actually occur in π. Also note that the corruption consistency of the input functions model the following reasonable intuition: consider a run of π that leads to certain inputs. Now suppose we run π again with the same random coins, however some players that were honest before are now corrupted, but are told to play honestly. Since all players make the same moves in the two cases, it is reasonable to expect that the resulting inputs should be the same, and this is what the corruption consistence of the input functions implies.

A typical example of an input function is where P_i provides inputs by secret-sharing them using polynomials of degree at most t. Here the input function reconstructs the input from the shares held by honest players using Lagrange interpolation. If the protocol guarantees that the shares are consistent with some polynomial of degree at most t, even if P_i is actively corrupt, then the input function only has to be defined on such sets of shares and is indeed corruption consistent.

Now, suppose we are given an adaptive environment $\mathcal{Z}$ and we want to show that the protocol is secure with respect to this environment. For this, we need to

[14] It is slightly tricky to formally and *generally* define what "following the protocol honestly" means, as a corrupted party takes all its instructions from the environment, via the influence port. However, in our context the definition we give here will do.

think about how we can use a static simulator $\mathcal{S}$. Of course, we cannot just run it against $\mathcal{Z}$ because $\mathcal{S}$ does not know how to handle corruptions that occur in the middle of the protocol. So instead, we will construct a family of static environments from $\mathcal{Z}$: for each set A that $\mathcal{Z}$ may corrupt, we construct environment $\mathcal{Z}_A$. Informally, what $\mathcal{Z}_A$ does is that it corrupts set A, but initially, it lets all players in A play honestly. It runs internally a copy of $\mathcal{Z}$ and lets it interact with the protocol as usual, where the only difference is that players in A are run honestly by $\mathcal{Z}_A$ instead of running by themselves. When $\mathcal{Z}$ corrupts a player in A, $\mathcal{Z}_A$ gives control of that player to $\mathcal{Z}$ and continues, if the corrupted player is not in A, $\mathcal{Z}_A$ outputs guess 0 and terminates. If $\mathcal{Z}$ outputs a guess $c \in \{0, 1\}$ without corrupting anyone outside A, then $\mathcal{Z}_A$ outputs the same guess c. A formal description is found below.

Agent $\mathcal{Z}_A$

Static environment constructed from $\mathcal{Z}$. It has the same port structure as $\mathcal{Z}$.

1. In the preamble, corrupt set A. Set up internally a (honest) copy P'_j of each player $\mathsf{P}_j \in A$. Also set up internally a copy of $\mathcal{Z}$. Connect the special ports of $\mathcal{Z}$ corresponding to each player $\mathsf{P}_j \in A$ to the special ports of P'_j. Connect all other special ports and the protocol ports of $\mathcal{Z}$ to the corresponding external ports.
2. When the system executes and some $\mathsf{P}_j \in A$ is activated, then if P_j has not been corrupted by $\mathcal{Z}$ (see next item) $\mathcal{Z}_A$ does the following: in case of passive corruption, a copy of the messages received by P_j is read from its leakage port and a copy of its state is kept inside P'_j. In case of active corruption, the messages received by P_j are read from its leakage port, and are given to the (honest) P'_j who determines the messages to send. $\mathcal{Z}_A$ makes P_j send these messages via its influence port. In both cases, leakage from P'_j is sent to $\mathcal{Z}$ as if P'_j was uncorrupted.
3. If $\mathcal{Z}$ decides to corrupt $\mathsf{P}_j \in A$, $\mathcal{Z}_A$ gives the current state of P'_j to $\mathcal{Z}$ and gives (passive or active) control of P_j to $\mathcal{Z}$.
4. If $\mathcal{Z}$ decides to corrupt $\mathsf{P}_j \notin A$, $\mathcal{Z}_A$ halts and outputs the guess 0.
5. If $\mathcal{Z}$ halts (having corrupted no player outside A) with guess $c \in \{0, 1\}$, $\mathcal{Z}_A$ halts and outputs guess c.

We know that $\mathcal{S}$ can do perfect simulation against any of the $\mathcal{Z}_A$ we just defined and this will be the basis of the adaptive simulator we construct later. Before we can construct the adaptive simulator, we need some auxiliary lemmas on how $\mathcal{S}$ behaves when interacting with the $\mathcal{Z}_A$'s.

Some notation: In the following $\mathbf{s}$ will denote an ordered sequence of players, and $A(\mathbf{s})$ will denote the set of players that occur in $\mathbf{s}$. $E_{\mathbf{s}}$ will be the event that the first $|\mathbf{s}|$ corruptions done by the environment are exactly those in $\mathbf{s}$ (in the specified order). Below, when we write views or conversations with subscript $\mathbf{s}$, for instance as in $V_{\mathcal{Z}}(\mathcal{Z} \diamond \pi \diamond \mathcal{R} | E_{\mathbf{s}})_{\mathbf{s}}$, this means that if a corruption outside $\mathbf{s}$ occurs, we truncate the view at the point where this corruption happens.

Finally, consider the copy of $\mathcal{Z}$ that is run internally by $\mathcal{Z}_{A(\mathbf{s})}$ where we execute $\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{IS}$ for some interactive system $\mathcal{IS}$ (such as $\pi \diamond \mathcal{R}$). We then let $V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{IS} | E_{\mathbf{s}})$ be its view, conditioned on $E_{\mathbf{s}}$. Since $\mathcal{Z}_{A(\mathbf{s})}$ runs $\mathcal{Z}$ "in the head" it is clear that $V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{IS}|E_{\mathbf{s}})$ can be deterministically and easily extracted from $V_{\mathcal{Z}_{A(\mathbf{s})}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{IS}|E_{\mathbf{s}})$, we will write this as

$$V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{IS}|E_{\mathbf{s}}) = \mathrm{Extr}(V_{\mathcal{Z}_{A(\mathbf{s})}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{IS}|E_{\mathbf{s}}))$$

We can now show that assuming $E_{\mathbf{s}}$ occurs, then $\mathcal{S}$ can be used to perfectly simulate (a part of) the view $\mathcal{Z}$ sees in the protocol, because it can simulate the view of $\mathcal{Z}_{A(\mathbf{s})}$:

Lemma 4.19 *Assuming $\mathcal{R}$ is input based, we have*

$$V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond F|E_{\mathbf{s}}) \overset{\mathrm{perf}}{\equiv} V_{\mathcal{Z}}(\mathcal{Z} \diamond \pi \diamond \mathcal{R}|E_{\mathbf{s}})_{\mathbf{s}}.$$

Proof We have $V_{\mathcal{Z}_{A(\mathbf{s})}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}) \overset{\mathrm{perf}}{\equiv} V_{\mathcal{Z}_{A(\mathbf{s})}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \pi \diamond \mathcal{R})$, since $\mathcal{S}$ is a good static simulator. So the two distributions are also the same when conditioning on $E_{\mathbf{s}}$, that is, we have

$$V_{\mathcal{Z}_{A(\mathbf{s})}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s}}) \overset{\mathrm{perf}}{\equiv} V_{\mathcal{Z}_{A(\mathbf{s})}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \pi \diamond \mathcal{R}|E_{\mathbf{s}}) \ . \tag{4.9}$$

The two distributions remain equal if we apply the same deterministic function to both of them, so if we apply Extr on both sides of (4.9) we get

$$V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s}}) \overset{\mathrm{perf}}{\equiv} V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \pi \diamond \mathcal{R}|E_{\mathbf{s}}) \ . \tag{4.10}$$

Moreover, from the point of view of $\mathcal{Z}$ (and still conditioning on $E_{\mathbf{s}}$), the only difference between $\mathcal{Z}_{A(\mathbf{s})} \diamond \pi \diamond \mathcal{R}$ and $\mathcal{Z} \diamond \pi \diamond \mathcal{R}$ is that in the first case the parties in A are run honestly by $\mathcal{Z}_{A(\mathbf{s})}$ until $\mathcal{Z}$ wants to corrupt them while in the second case they run as honest players in π. This makes no difference to $\mathcal{R}$ since it is input based (by the honest behaviour equivalence property) and hence it makes no difference to $\mathcal{Z}$ either. So we have

$$V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \pi \diamond \mathcal{R}|E_{\mathbf{s}}) \overset{\mathrm{perf}}{\equiv} V_{\mathcal{Z}}(\mathcal{Z} \diamond \pi \diamond \mathcal{R}|E_{\mathbf{s}})_{\mathbf{s}} \ . \tag{4.11}$$

The lemma now follows from (4.10) and (4.11). □

We also need to consider a connection between simulation against several different $\mathcal{Z}_{A(\mathbf{s})}$'s: For a sequence of players $\mathbf{s}$, we can append a player P_i (who is not in $\mathbf{s}$) at the end of the sequence. We write this new sequence as $\mathbf{s}, i$, and define $E_{\mathbf{s},i}$ and $A(\mathbf{s}, i)$ as before.

Lemma 4.20 *Assuming $\mathcal{R}$ is input based, we have*

$$V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond F|E_{\mathbf{s},i}) \overset{\mathrm{perf}}{\equiv} V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond F|E_{\mathbf{s},i})_{\mathbf{s}}.$$

Proof Since $\mathcal{S}$ is a good static simulator, we have by a similar argument as in the proof of Lemma 4.19 that

$$\begin{aligned} V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s},i}) &= \mathrm{Extr}(V_{\mathcal{Z}_{A(\mathbf{s})}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s},i})) && (4.12)\\ &\overset{\mathrm{perf}}{\equiv} \mathrm{Extr}(V_{\mathcal{Z}_{A(\mathbf{s})}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \pi \diamond \mathcal{R}|E_{\mathbf{s},i})) && (4.13)\\ &= V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \pi \diamond \mathcal{R}|E_{\mathbf{s},i}) \ . && (4.14) \end{aligned}$$

Note that, assuming $E_{\mathbf{s},i}$ occurs, the only difference between $\mathcal{Z}_{A(\mathbf{s})} \diamond \pi \diamond \mathcal{R}$ and $\mathcal{Z}_{A(\mathbf{s},i)} \diamond \pi \diamond \mathcal{R}$ is that in the latter case P_i is run honestly by $\mathcal{Z}_{A(\mathbf{s},i)}$ whereas in the former it plays honestly as party in the protocol. As $\mathcal{R}$ is input based, this makes no difference to $\mathcal{R}$ and hence also no difference to the view of $\mathcal{Z}$ as long as we only consider what happens up to the point where P_i is corrupted. So we have

$$V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \pi \diamond \mathcal{R}|E_{\mathbf{s},i}) \overset{\text{perf}}{\equiv} V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s},i)} \diamond \pi \diamond \mathcal{R}|E_{\mathbf{s},i})_{\mathbf{s}} \ . \tag{4.15}$$

Using again that $\mathcal{S}$ is a good static simulator, it follows in that same way as before that

$$V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s},i)} \diamond \pi \diamond \mathcal{R}|E_{\mathbf{s},i})_{\mathbf{s}} \overset{\text{perf}}{\equiv} V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s},i})_{\mathbf{s}} \ . \tag{4.16}$$

The lemma now follows from (4.14), (4.15) and (4.16). □

We now want to show that if we consider both the view of $\mathcal{Z}$ and the inputs and outputs that F exchanges, we still have a similar result as in the previous lemma. This does not have to be true in general, but is indeed true if $\mathcal{S}$ is conversation-based and if F is input-based:

Lemma 4.21 *Let* $\mathrm{St}_{\mathsf{F}}(\cdot)$ *be the state of* F *after running in some interactive system. Then, if* $\mathcal{S}$ *is conversation-based and* F *is input-based, we have*

$$(V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s},i}),\ \mathrm{St}_{\mathsf{F}}(\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s},i}))$$

$$\overset{\text{perf}}{\equiv} (V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s},i})_{\mathbf{s}},\ \mathrm{St}_{\mathsf{F}}(\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s},i})_{\mathbf{s}}).$$

Proof We already have from Lemma 4.20 that the view of $\mathcal{Z}$ has the same distribution in the two systems. We then prove the lemma by arguing that because $\mathcal{S}$ is conversation-based, all inputs sent to F follow deterministically from the view of $\mathcal{Z}$ and will be the same in both systems, so since F is input-based, the distribution of its state must be the same as well.

In more detail, consider a view v for $\mathcal{Z}$ that occurs with non-zero probability (in both systems). Note first that by public input/output provision of F, one can infer from v in which rounds inputs were provided to F or outputs were sent, so these must be the same in the two systems. Consider a particular input and say it comes from player P_j. We do a case analysis:

$\mathsf{P}_j \notin A(\mathbf{s},i)$: In this case P_j is honest throughout in both systems. This means $\mathcal{Z}$ provides the input directly to F so the input occurs in v and is the same in both systems.

$\mathsf{P}_j = \mathsf{P}_i$: In the system $\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}$, P_i is honest and $\mathcal{Z}$ provides directly to F the input, say x, that occurs in v. In $\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}$, P_i is corrupt but is told to play honestly on input x as provided by $\mathcal{Z}$. Then $\mathcal{S}$ decides on the input to F based on the conversation of P_i, and this will result in x by the "honest behavior implies correct input" property.

$\mathsf{P}_j \in A(\mathbf{s})$ and has not been corrupted by $\mathcal{Z}$ when the input is provided: In this case, in both systems $\mathcal{Z}$ provides input x to P_j who plays honestly and $\mathcal{S}$ decides the input to F based on the conversation, which will be x, again by the "honest behavior implies correct input" property.

$\mathsf{P}_j \in A(\mathbf{s})$ and has been corrupted by $\mathcal{Z}$ when the input is provided: in this case $\mathcal{S}$ will in both systems decide on the input based on the conversation of P_j. Note first that the messages P_j has exchanged with all players that $\mathcal{Z}$ has not corrupted yet is part of v and is therefore the same in both systems (this includes at least all players outside $A(\mathbf{s})$). However, from the point of view of $\mathcal{S}$, the conversation of P_j is *not* the same in the two systems: in $\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}$, it consists of messages exchanged with players outside $A(\mathbf{s})$, while in $\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}$ it consists of a subset of these messages, namely those exchanged with players outside $A(\mathbf{s}, i)$. However, since the input functions are corruption-consistent, the input computed by $\mathcal{S}$ is nevertheless the same in the two systems. □

In the following, we will consider a situation where we execute the system $\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}$ until a point where $E_{\mathbf{s},i}$ has occurred. At this point, $\mathcal{Z}_{A(\mathbf{s})}$ would halt. However, by Lemma 4.21, as far as $\mathcal{Z}$ and the state of F is concerned, we might as well have been running $\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}$, and unlike $\mathcal{Z}_{A(\mathbf{s})}$, $\mathcal{Z}_{A(\mathbf{s},i)}$ would be able to continue even after P_i is corrupted. So if we could somehow "pretend" that in fact it was the latter system we ran, we would not have to stop when P_i is corrupted.

To help us do this trick, we consider an execution of $\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}$ where $E_{\mathbf{s},i}$ occurs. Say that the values of $V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s},i})_{\mathbf{s}}$ and $\mathrm{St}_{\mathsf{F}}(\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s},i})_{\mathbf{s}}$ are v and w. We then define $D_{v,w}$ to be the joint distribution of the states of $\mathcal{Z}_{A(\mathbf{s},i)}$ and $\mathcal{S}$ at the point where P_i is corrupted, given v and w. Note that since we assume that $E_{\mathbf{s},i}$ occurred, the state of $\mathcal{Z}_{A(\mathbf{s},i)}$ consists of a state of $\mathcal{Z}$ that is fixed by v and a view of P_i who has been playing honestly so far. So we can think of the output of $D_{v,w}$ as a view of P_i plus a state of $\mathcal{S}$.

Lemma 4.22 *Consider an execution of the system $\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}$ until a point where $E_{\mathbf{s},i}$ has occurred. Let*

$$V_{\mathcal{Z}}(\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s},i})_{\mathbf{s}} = v \text{ and } \mathrm{St}_{\mathsf{F}}(\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}|E_{\mathbf{s},i})_{\mathbf{s}} = w.$$

Let $\mathrm{io}_{\mathcal{S}}$ *be the string of inputs and outputs $\mathcal{S}$ has exchanged with F, and let* $\mathrm{Conv}_{\mathcal{Z}}$ *be the conversation of $\mathcal{Z}$ in the execution. Then one can sample from the distribution $D_{v,w}$ if given* $\mathrm{io}_{\mathcal{S}}$ *and* $\mathrm{Conv}_{\mathcal{Z}}$. *In particular, $D_{v,w}$ depends only on* $\mathrm{io}_{\mathcal{S}}$ *and* $\mathrm{Conv}_{\mathcal{Z}}$.

Proof Recall that $\mathcal{Z}_{A(\mathbf{s},i)}$ consists of a copy of $\mathcal{Z}$ and copies of players in $A(\mathbf{s}, i)$. However, since $E_{\mathbf{s},i}$ occurs, all players in $A(\mathbf{s})$ have been corrupted earlier by $\mathcal{Z}$, so their entire view until they were corrupted by $\mathcal{Z}$ is part of $\mathrm{Conv}_{\mathcal{Z}}$.

The sampling procedure we claim is now very simple: for each possible set of coins for P_i and for $\mathcal{S}$, we will test if these coins are consistent with the values of $\mathrm{io}_{\mathcal{S}}$ and $\mathrm{Conv}_{\mathcal{Z}}$ we are given. We do the test by simulating P_i and $\mathcal{S}$ running as part of the system $\mathcal{Z}_{A(\mathbf{s},i)} \diamond \mathcal{S} \diamond \mathsf{F}$. This is possible because the given values $\mathrm{io}_{\mathcal{S}}$ and $\mathrm{Conv}_{\mathcal{Z}}$ specify the entire communication that P_i and $\mathcal{S}$ should have with F and $\mathcal{Z}$. If the current random coins lead to P_i or $\mathcal{S}$ sending a message that is inconsistent with $\mathrm{io}_\mathcal{S}, \mathrm{Conv}_\mathcal{Z}$, we throw away this set of coins. Finally, we choose randomly a set of coins among those that survived and output the resulting view of P_i and state of $\mathcal{S}$. □

In view of the result of Lemma 4.22, we will write $D_{\mathrm{io}_\mathcal{S},\mathrm{Conv}_\mathcal{Z}}$ instead of $D_{v,w}$ in the following. We can now specify the final tool we need to build an adaptive simulator, namely the sampling we have just seen must be possible to do efficiently:

Definition 4.23 *Consider a probabilistic algorithm* Patch *that takes as input strings* $\mathrm{io}_\mathcal{S}$ *and* $\mathrm{Conv}_\mathcal{Z}$ *of the form as specified in Lemma 4.22.* Patch *is said to be a* **good sampling function** *if it satisfies the following:*

- *It is polynomial time computable.*
- *The output* $\mathrm{Patch}(\mathrm{io}_\mathcal{S}, \mathrm{Conv}_\mathcal{Z})$ *is distributed according to* $D_{\mathrm{io}_\mathcal{S},\mathrm{Conv}_\mathcal{Z}}$.

We are now finally ready to specify the main result of this section:

Theorem 4.24 *Assume we are given a simulator $\mathcal{S}$ for protocol π and functionality F such that $\mathcal{Z}_{static} \diamond \pi \diamond \mathcal{R} \overset{\text{perf}}{\equiv} \mathcal{Z}_{static} \diamond \mathcal{S} \diamond \mathsf{F}$ for any static and synchronous environment $\mathcal{Z}_{static}$ corrupting at most t parties.*

Assume further that $\mathcal{S}$ is conversation-based, F and $\mathcal{R}$ are input-based, and that we are given a good sampling function Patch. *Then there exists a simulator $\mathcal{S}'$ such $\mathcal{Z} \diamond \pi \diamond \mathcal{R} \overset{\text{perf}}{\equiv} \mathcal{Z} \diamond \mathcal{S} \diamond \mathsf{F}$ for any adaptive and synchronous environment $\mathcal{Z}$ corrupting at most t parties.*

Proof We specify the algorithm of our adaptive simulator $\mathcal{S}'$. To do this, suppose that if we are given a string $\mathrm{io}_\mathcal{S}$ containing inputs and outputs that $\mathcal{S}$ has exchanged with F on behalf of corrupted players in a set A. Suppose we are also given the inputs and outputs io_i that some honest player P_i has exchanged with F in the same execution. Then we can merge these strings in a natural way: we define $\mathrm{Merge}(\mathrm{io}_\mathcal{S}, \mathrm{io}_i)$ to be the string that contains, for every protocol round, the inputs to F that occur in either $\mathrm{io}_\mathcal{S}$ or io_i, and also outputs from F that occur in either $\mathrm{io}_\mathcal{S}$ or io_i. Note that $\mathrm{Merge}(\mathrm{io}_\mathcal{S}, \mathrm{io}_i)$ is a string of inputs and outputs that $\mathcal{S}$ might have exchanged with F if $A \cup \{\mathsf{P}_i\}$ had been the corrupted set (and P_i had behaved honestly).

To see that this simulation works, note first that it is obvious that $\mathrm{Conv}_\mathcal{Z}$ contains at all times the conversation of $\mathcal{Z}$ so far, and that $\mathrm{io}_\mathcal{S}$ contains at all times the inputs and outputs we have exchanged with F so far.

We can now show the following

Claim: whenever $\mathcal{S}'$ enters step 2 the state of $\mathcal{Z}, \mathcal{S}$ and F are distributed exactly as in a run of $\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}$ where $E_\mathbf{s}$ occurs.

We show this by induction: the claim is trivial when we enter step 2 the first time since here $\mathbf{s}$ is empty. So consider a later stage where we enter step 2, and write the current $\mathbf{s}$ as $\mathbf{s} = \mathbf{s}', i$. The previous time we entered step 2, by induction hypothesis, the state of $\mathcal{Z}, \mathcal{S}$ and F were distributed exactly as in a run

Agent $\mathcal{S}'$

Adaptive simulator constructed from $\mathcal{S}$.

1. Set $\mathbf{s}$ be the empty sequence. Set $\text{io}_{\mathcal{S}}, \text{Conv}_{\mathcal{Z}}$ to be the empty strings. Set up a copy of $\mathcal{S}$ in its initial state. Tell $\mathcal{S}$ as input (in the preamble) that the empty set is the corrupted set.
2. Whenever $\mathcal{S}'$ is activated, if the input is a request to corrupt a new player P_i, it goes to the next step. Otherwise, it runs $\mathcal{S}$ on the input received and sends the output $\mathcal{S}$ produces on the corresponding output port of its own. Messages exchanged with $\mathcal{Z}$ are appended to $\text{Conv}_{\mathcal{Z}}$, inputs/outputs exchanged with F are appended to $\text{io}_{\mathcal{S}}$.
3. Set $\mathbf{s} = \mathbf{s}, i$. Send a request to corrupt P_i to F and get a string io_i back. Set $\text{io}_{\mathcal{S}} = \text{Merge}(\text{io}_{\mathcal{S}}, \text{io}_i)$. Compute $(v_i, \text{St}') = \text{Patch}(\text{io}_{\mathcal{S}}, \text{Conv}_{\mathcal{Z}})$. Put $\mathcal{S}$ in state St', send v_i to the environment, append v_i to $\text{Conv}_{\mathcal{Z}}$ and go to Step 2.

of $\mathcal{Z}_{A(\mathbf{s}')} \diamond \mathcal{S} \diamond \mathsf{F}$ where $E_{\mathbf{s}'}$ occurs. During the following execution of step 2, $\mathcal{S}'$ simply ran $\mathcal{S}$, so when the i'th player is corrupted, the state of $\mathcal{Z}, \mathcal{S}$ and F were distributed exactly as in a run of $\mathcal{Z}_{A(\mathbf{s}')} \diamond \mathcal{S} \diamond \mathsf{F}$ where $E_{\mathbf{s}',i}$ occurs. Now, by Lemma 4.21, the views (and hence state) of $\mathcal{Z}$ and the state of F are distributed as in a run of $\mathcal{Z}_{A(\mathbf{s}',i)} \diamond \mathcal{S} \diamond \mathsf{F}$ where $E_{\mathbf{s},i}$ occurs.

Then Patch was run and the claim now follows by assumption on Patch, if we show that the inputs $\text{Conv}_{\mathcal{Z}}, \text{io}_{\mathcal{S}}$ we use have the distribution they would have in $\mathcal{Z}_{A(\mathbf{s}',i)} \diamond \mathcal{S} \diamond \mathsf{F}$, given the current values of the view of $\mathcal{Z}$ and the state of F. This is trivially true for $\text{Conv}_{\mathcal{Z}}$ as it follows deterministically from the view of $\mathcal{Z}$. For $\text{io}_{\mathcal{S}}$, note that the inputs to F that occur in this string also follow deterministically from the view of $\mathcal{Z}$, we argued this in the proof of Lemma 4.21. But since F is input-based, the resulting outputs from F will be the same regardless of whether we run $\mathcal{Z}_{A(\mathbf{s})}$ or $\mathcal{Z}_{A(\mathbf{s}',i)}$, and so $\text{io}_{\mathcal{S}}$ has the desired distribution.

We can now argue that $V_{\mathcal{Z}}(\mathcal{Z} \diamond \pi \diamond \mathcal{R}) \overset{\text{perf}}{\equiv} V_{\mathcal{Z}}(\mathcal{Z} \diamond \mathcal{S}' \diamond \mathsf{F})$ which clearly implies the theorem.

We will consider the executions of step 2 one by one. In the first execution, by the above claim, the state of $\mathcal{Z}, \mathcal{S}$ and F are distributed exactly as in a run of $\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}$ where $E_{\mathbf{s}}$ occurs. But here $\mathbf{s}$ is the empty sequence so $E_{\mathbf{s}}$ always occurs. In step 2, we simply run $\mathcal{S}$, so by Lemma 4.19, we obtain a perfect simulation of $\mathcal{Z}$'s view until the point where is halts or corrupts the first player. In particular, in the latter case, that player is chosen with the distribution we would also see in a real execution of the protocol. When we have executed step 3, again by the claim, the state of $\mathcal{Z}, \mathcal{S}$ and F are distributed exactly as in a run of $\mathcal{Z}_{A(\mathbf{s})} \diamond \mathcal{S} \diamond \mathsf{F}$ where $E_{\mathbf{s}}$ occurs, and where $\mathbf{s}$ now contains one player. Again by Lemma 4.19, while excuting step 2 we obtain a perfect simulation of $\mathcal{Z}$'s view from the point where it corrupts the first player until the point where is halts or corrupts the second player.

Repeating this argument at most t times (since $\mathcal{Z}$ can corrupt only so many players), we see that we get a perfect simulation of the entire view of $\mathcal{Z}$. □

Using Theorem 4.24

In order to use Theorem 4.24 on a concrete protocol, one has to first construct a static simulator $\mathcal{S}$, verify that is is conversation-based and that the target functionality F is input-based. This is usually quite easy. Then one has to construct an efficient procedure Patch. This may seem harder because the formal definition is quite technical and involves two static environments constructed from an arbitrary adaptive environment.

We therefore give an explanation in more "human" language of what Patch must be able to do. Recall that when Patch is called, players in the sequence $\mathbf{s}$ were corrupted earlier and a new player P_i has just been corrupted. We know $\text{io}_{\mathcal{S}}$, i.e., all the inputs and outputs that players in $\mathbf{s}, i$ have exchanged with the functionality F. and we know $\text{Conv}_{\mathcal{Z}}$, that is, the protocol execution as seen by $\mathcal{Z}$ until now. Patch now has two tasks that must solved efficiently:

The first one is to construct a complete view of P_i playing honestly in the protocol until now, and this must be consistent with $\text{io}_{\mathcal{S}}$ and $\text{Conv}_{\mathcal{Z}}$. In particular, $\text{io}_{\mathcal{S}}$ contains the inputs and outputs P_i has exchanged with F and $\text{Conv}_{\mathcal{Z}}$ contains the messages that P_i has sent to players who were corrupted earlier. The reason why this can be feasible for, e.g., protocols based on secret sharing is that the corrupted players (actually, $\mathcal{Z}$) have seen less than t shares of the secrets of P_i. This leaves the secrets undetermined, so when we now learn the actual secret values of P_i (from $\text{io}_{\mathcal{S}}$), we are able create a full set of shares that is consistent with the secrets and the shares of the corrupt players.

The second task is to create a new state for the simulator $\mathcal{S}$. This must be the state as it would have looked if we had run $\mathcal{S}$ with all players in $\mathbf{s}, i$ being corrupt from the start, but where P_i plays honestly until the current point in time.

In this book we will often consider a strategy for building $\mathcal{S}$ that makes both tasks easier: Initially, $\mathcal{S}$ sets up a copy of the honest players in the protocol, and gives them dummy inputs. It now simulates by essentially letting these "virtual players" do the protocol with the corrupt players (controlled by $\mathcal{Z}$). The state of $\mathcal{S}$ is simply the state of the virtual players. Now, when P_i is corrupted, Patch will compute how the view of the virtual copy of P_i should change, now that we know its actual inputs, and will then update the state of the other virtual players to make everything consistent, including $\text{io}_{\mathcal{S}}$ and $\text{Conv}_{\mathcal{Z}}$. This creates the required view of P_i and the new state of $\mathcal{S}$ is the state of the updated virtual players, except for P_i.

In the following, we will use the method given in Theorem 4.24 in most cases when proving adaptive security. But we will not always explicitly verify that the functionality is input-based and the static simulator is conversation-based, as this would be rather tedious reading.

Extension to Statistical Security

It is not clear that Theorem 4.24 is true for statistical security in general. But it not hard to see that it holds in an important special case: suppose we can define an "error event" E such that E occurs with negligible probability, and we can make a static simulator $\mathcal{S}$ that simulates perfectly if E does not occur. Then we can redo the proof of Theorem 4.24 while conditioning throughout on E not occurring. We leave the details to the reader.

4.6 Notes

In hindsight, it can seem quite surprising that at the time when the first completeness results for multiparty computation were established, we did not have generally accepted definitions of security for multiparty computation protocols. These crystallized only much later after a lot of research work. There are many reasons for this, but a main complication was the issue of composition: It is a natural requirement that protocols should remain secure even when composed with other protocols and for a long time it was not clear how a definition should be put together to ensure this.

In 1991, Micali and Rogaway [139] as well as Beaver[9] put forward definitions. It was realized already here that simulation-based security is the way to go, but it was not until the work around 2000 of Canetti [37] and independently Pfitzmann, Schunter and Waidner [152] that security under arbitrary concurrent composition could be expressed. A related, but different approach known as "constructive cryptography" was intiated recently by Ueli Maurer [136]. This framework is also simulation-based and gives security under composition, but is technically different from the UC model in several ways.

It is natural to ask if the early protocols actually turned out to be secure under these later definition? It turns out that all the protocols with unconditional security underlying the completeness results from the late 80-ties are in fact secure in the strongest sense: they are UC secure against adaptive adversaries (see below for more on the history of the security definitions). In contrast, the computationally secure protocols, for instance [103] are only known to be statically secure. This is mainly due to a technical problem related to the fact that when simulating encrypted communication, the simulator will implicitly commit itself to some plaintext that it would have to change if the sender or receiver are corrupted. Adaptive security can be obtained using so called non-commiting encryption [40], but only at the cost of significant technical complications.

It may superficially seem that all the technical problems that prevent us from proving adaptive security in the computational case go away in the information theoretic setting. This is not true, however, there are natural examples of information theoretically secure protocols that are statically secure but not adaptively secure [66].

The model and security definitions in this book are based on Canetti's work on Universal Composition, the UC model. We depart somewhat, however, from his original model. Basically, we sacrifice some generality to have a simpler model that is easier to explain and use. For instance, we demand that agents pass control in a certain nicely structured way, in order to preserve polynomial running time when we compose agents. We also define our agents as the more abstract IO automata rather than Interactive Turing Machines. The modifications make our model somewhat similar to the constructive cryptography model [136], although several technical differences remain.

It should also be mentioned that several variants of the UC model have been proposed more recently in the literature, see for instance the GUC framework by Canetti, Dodis, Pass and Walfish [39] and the GNUC framework by Hofheintz and Shoup [111].

5

Information Theoretic Robust MPC Protocols

Contents

5.1 Introduction

In this chapter, we show how to modify the protocol from Chapter 3 to make it secure also against active attacks. In the terms introduced in the previous chapter, the goal is to implement $\mathsf{F}^f_{\mathtt{SFE}}$, where we will assume that we have available the ideal functionality $\mathsf{F}_{\mathtt{SC}}$ for secure communication and consensus broadcast, and also another ideal commitment functionality $\mathsf{F}_{\mathtt{COM}}$ that we explain in detail below. In Section 5.4 we show how the commitment functionality can be implemented based only on $\mathsf{F}_{\mathtt{SC}}$ under appropriate assumptions on t, the number of corrupted players.

The final results are collected in Section 5.5. We note already now, however, that if $t < n/3$ then $\mathsf{F}^f_{\mathtt{SFE}}$ can be implemented with perfect security without using broadcast, but assuming secure channels between each pair of players. If we additionally assume broadcast and we allow a non-zero error probability, then $t < n/2$ will be sufficient. These bounds are tight, as we shall see later.

Agent $\mathrm{F}_{\texttt{COM}}$

Ideal functionality for homomorphic commitment. The functionality does a complete breakdown if inputs from honest players are not given as specified. Output from a command is sent once the round in which to deliver is specified on the influence port.

commit: This command is executed if in some round honest player P_i sends $(\texttt{commit}, i, \mathit{cid}, a)$ and in addition all other honest players send $(\texttt{commit}, i, \mathit{cid}, ?)$. Leak $(\texttt{commit}, i, \mathit{cid}, ?)$ [a] and store the triple (i, cid, a). Here, cid is just a commitment identifier, and a is the value committed to [b]. Output $(\texttt{commit}, i, \mathit{cid}, \texttt{success})$ to all parties.
The command is also executed if P_i is corrupt and all honest players send $(\texttt{commit}, i, \mathit{cid}, ?)$. In this case leak $(\texttt{commit}, i, \mathit{cid}, ?)$. Then, if P_i sends $(\texttt{commit}, i, \mathit{cid}, a)$ before output is delivered, store data and output success as if P_i was honest. Otherwise, store nothing, and output $(\texttt{commit}, i, \mathit{cid}, \texttt{fail})$ to all parties.

public commit: This command is executed if in some round all honest parties send $(\texttt{pubcommit}, i, \mathit{cid}, a)$. In this case $\mathsf{F}_{\texttt{COM}}$ records the triple (i, cid, a). Leak $(\texttt{pubcommit}, i, \mathit{cid}, a)$ and output $(\texttt{pubcommit}, i, \mathit{cid}, \texttt{success})$ to all parties.[c]

open: This command is executed if in some round all honest players send $(\texttt{open}, i, \mathit{cid})$ and some (i, cid, a) is stored. Leak $(\texttt{open}, i, \mathit{cid})$ or, if any party is corrupted, leak $(\texttt{open}, i, \mathit{cid}, a)$.
If P_i is honest, output $(\texttt{open}, i, \mathit{cid}, a)$ to all parties. If P_i is corrupted and inputs $(\texttt{open}, i, \mathit{cid})$ before output is delivered, output a to all parties as above. Otherwise output $(\texttt{open}, i, \mathit{cid}, \texttt{fail})$ to all parties.

designated open: This command is executed if in some round all honest players send $(\texttt{open}, i, \mathit{cid}, j)$ and some (i, cid, a) is stored. Leak $(\texttt{open}, i, \mathit{cid}, j)$ or, if P_j is corrupted, leak $(\texttt{open}, i, \mathit{cid}, j, a)$.
If P_i is honest, output $(\texttt{open}, i, \mathit{cid}, j, a)$ to P_j and also output $(\texttt{open}, i, \mathit{cid}, j, \texttt{success})$ to all other parties. If P_i is corrupted and inputs $(\texttt{open}, i, \mathit{cid}, j)$ send output as if P_i was honest. Otherwise output $(\texttt{open}, i, \mathit{cid}, \texttt{fail})$ to all parties.

add: This command is executed if in some round all honest players send $(\texttt{add}, i, \mathit{cid}_1, \mathit{cid}_2, \mathit{cid}_3)$ and some (i, cid_1, a_1) is stored and some (i, cid_2, a_2) is stored. Leak $(\texttt{add}, i, \mathit{cid}_1, \mathit{cid}_2, \mathit{cid}_3)$. As a result $\mathsf{F}_{\texttt{COM}}$ stores $(i, \mathit{cid}_3, a_1 + a_2)$. The output to all parties is $(\texttt{add}, i, \mathit{cid}_1, \mathit{cid}_2, \mathit{cid}_3, \texttt{success})$.

mult by constant: This command is executed if in some round all honest players send $(\texttt{mult}, i, \alpha, \mathit{cid}_2, \mathit{cid}_3)$ and $\alpha \in \mathbb{F}$ and some (i, cid_2, a_2) is stored. As a result $\mathsf{F}_{\texttt{COM}}$ stores $(i, \mathit{cid}_3, \alpha a_2)$. The output to all parties is $(\texttt{mult}, i, \alpha, \mathit{cid}_2, \mathit{cid}_3, \texttt{success})$. Leak $(\texttt{mult}, i, \alpha, \mathit{cid}_2, \mathit{cid}_3)$.

[a] We leak $(\texttt{commit}, i, \mathit{cid}, ?)$ to say that an implementation need not hide that a commitment is being made. More technically, in the simulation $\mathsf{S} \diamond \mathsf{F}_{\texttt{COM}}$, the simulator will this way learn that it should start simulating a commitment. If all parties are honest, the simulator has no other way to learn this essential fact.

[b] We require that all honest players agree to the fact that a commitment should be made because an implementation will require the active participation of all honest players.

[c] The difference here is that all parties input a and that P_i is forced to commit. In an implementation the other parties can in principle just remember that P_i is committed to the known a, but it is convenient to have an explicit command for this.

5.2 Model for Homomorphic Commitments and some Auxiliary Protocols

We will assume that each player P_i can commit to a value $a \in \mathbb{F}$, while keeping his choice secret. He may, however, choose to reveal a later. We shall see how this can be implemented by distributing and/or broadcasting some information to other players. We model it here by assuming that we have an ideal functionality $\mathsf{F}_{\mathtt{COM}}$. To commit, one simply sends a to $\mathsf{F}_{\mathtt{COM}}$, who will then keep it until P_i asks to have it revealed. Formally, we assume $\mathsf{F}_{\mathtt{COM}}$ is equipped with commands `commit` and `open` as described in the box.

Some general remarks on the definition of $\mathsf{F}_{\mathtt{COM}}$: since the implementation of any of the commands may require all (honest) players to take part actively, we require that all honest players in a given round send the same commands to $\mathsf{F}_{\mathtt{COM}}$ in order for the command to be executed. In some cases, such as a commitment, we can of course not require that all players send exactly the same information since only the committing players knows the value to be committed to. So in such a case, we require that the committer sends the command and his secret input, while the others just send the command. If $\mathsf{F}_{\mathtt{COM}}$ is not used as intended, e.g., the honest players do not agree on the command to execute, $\mathsf{F}_{\mathtt{COM}}$ will do a complete breakdown. It will also do a complete breakdown if an honest party commits under the same *cid* twice or uses the same cid_3 twice in an addition command or a multiplication command. This is to ensure that all values are stored under unique identifiers. It will also do a complete breakdown if the honest parties use some cid_2 or cid_3 which has not been defined yet, i.e., if they for instance input $(\mathtt{add}, i, cid_1, cid_2, cid_3)$, but no (i, cid_1, a) is stored.

Below we will use the following convention for specifying $\mathsf{F}_{\mathtt{COM}}$:

- When we say that a command gives a particular output, then this output is not delivered immediately. Instead it will be specified via `COM.infl` in which round to deliver, and in that round the output to all parties will be given, i.e., $\mathsf{F}_{\mathtt{COM}}$ guarantees simultaneous output with adversarially chosen output round – see Section 4.4.3 for a motivation for these choices.
- If a party P_i is corrupted in between a command is given to $\mathsf{F}_{\mathtt{COM}}$ and the output delivery round, then we allow that the input of P_i to the command is changed via `COM.infl`, and the outputs will be recomputed accordingly. As an example, if P_i becomes corrupted in **commit** before the output is delivered, then we allow the environment to delete the input of P_i, thereby making the output $(\mathtt{commit}, i, cid, \mathtt{fail})$. This models that if a committer P_i becomes (adaptively) corrupted during the implementation of the commitment protocol, it is allowed that the now corrupted P_i can make the protocol fail. This is important for being able to prove adaptive security: Requiring that the protocol will always terminate correctly if just the first round was run honestly is often impossible to implement.
- When we say that a command leaks a value, then the value is output on `COM.leak` in the round where inputs to the command are given, not in the

delivery round. For a discussion of this so-called rushing behavior, see the specification and discussion of F_{SC}.

Below we will use the following short-hand for describing interactions with $\mathsf{F}_{\mathsf{COM}}$. The symbol $\langle\cdot\rangle_i$ denotes a variable in which $\mathsf{F}_{\mathsf{COM}}$ keeps a committed value received from player P_i. Thus when we write $\langle a\rangle_i$, this means that player P_i has committed to a. We also need the following notation:

$\langle a\rangle_i \leftarrow a$: the `commit` command is executed to let P_i commit to a.
$\langle a\rangle_i \Leftarrow a$: the `pubcommit` command is used to force P_i to commit to a.
$a \leftarrow \langle a\rangle_i$: the `open` command is executed to let all parties learn a.
$(\mathsf{P}_j)a \leftarrow \langle a\rangle_i$: the designated `open` command is executed to let P_j learn a.

It is clear from the description that all players know at any point which committed values have been defined. Of course, the value committed to is not known to the players (except the committer), but nevertheless, they can ask $\mathsf{F}_{\mathsf{COM}}$ to manipulate committed values, namely to add committed values and multiply them by public constants, as long as the involved variables belong to the same party. This is done by issuing the `add` or `mult` commands. The following notation will stand for execution of these commands:

$$\langle a_3\rangle_i \leftarrow \langle a_1\rangle_i + \langle a_2\rangle_i \quad \langle a_3\rangle_i \leftarrow \alpha\langle a_2\rangle_i,$$

where it is understood that $a_3 = a_1 + a_2$ respectively $a_3 = \alpha a_2$.

Agent $\mathsf{F}_{\mathsf{COM}}$ (continued)

Advanced manipulation commands for the functionality $\mathsf{F}_{\mathsf{COM}}$:

transfer: This command is executed if in some round all honest players send $(\mathtt{transfer}, i, cid, j)$ and some (i, cid, a) is stored. Leak $(\mathtt{transfer}, i, cid, j)$ or, if P_j is corrupted, leak $(\mathtt{transfer}, i, cid, j, a)$.
If P_i is honest, store (j, cid, a). Output $(\mathtt{transfer}, i, cid, j, \mathtt{success})$ to all parties except P_j, and $(\mathtt{transfer}, i, cid, j, a)$ to P_j. If P_i is corrupted and inputs $(\mathtt{transfer}, i, cid, j)$, store data and output as if P_i was honest. Otherwise store nothing and output $(\mathtt{transfer}, i, cid, j, \mathtt{fail})$ to all parties.

mult: This command is executed if in some round all honest players send $(\mathtt{mult}, i, cid_1, cid_2, cid_3)$ and some (i, cid_1, a_1) is stored and some (i, cid_2, a_2) is stored. Leak $(\mathtt{mult}, i, cid_1, cid_2, cid_3)$.
If P_i is honest, store (i, cid_3, a_1a_2) and output $(\mathtt{mult}, i, cid_1, cid_2, cid_3, \mathtt{success})$ to all parties. If P_i is corrupt and inputs $(\mathtt{mult}, i, cid_1, cid_2, cid_3)$, output success as if P_i was honest. Otherwise no value is stored and the output to all parties is $(\mathtt{mult}, i, cid, j, \mathtt{fail})$.

The final part of the description of $\mathsf{F}_{\mathsf{COM}}$ includes two *advanced manipulation* commands. The `transfer` command transfers a committed value from one party to another. The idea is that even if P_i has committed to a value a, after a transfer to P_j, we are in a situation equivalent to what we would have if P_j had committed to a in the first place – except, of course, that P_i also knows a. The `mult` command is used by a player to create, given commitments to a_1, a_2 a new commitment that is guaranteed to contain $a_1 a_2$. We write

$$\langle a\rangle_j \leftarrow \langle a\rangle_i, \quad \langle a_3\rangle_i \leftarrow \langle a_1\rangle_i\langle a_2\rangle_i$$

to denote executions of these commands.

The first step towards using $\mathsf{F}_{\texttt{COM}}$ for secure function evaluation, is to implement the advanced manipulation commands from the basic set of operations. Formally speaking, we can think of this as follows: we are given a functionality $\mathsf{F}_{\texttt{COM-SIMPLE}}$ that has all the commands of $\mathsf{F}_{\texttt{COM}}$, except the the advanced manipulation commands. We then build protocols that will implement $\mathsf{F}_{\texttt{COM}}$ when given access to $\mathsf{F}_{\texttt{COM-SIMPLE}}$. Once we show that these protocols work as defined in the previous chapter, the UC theorem allows us to assume in the following that we have $\mathsf{F}_{\texttt{COM}}$ available. On the other hand, when it comes to implementing commitments, we will only need to implement $\mathsf{F}_{\texttt{COM-SIMPLE}}$.

Protocol TRANSFER

If any command fails in Steps 1-3, it will be clear that either P_i or P_j is corrupt. In this case, go to Step 4

1. $(\mathsf{P}_j) a \leftarrow \langle a\rangle_i$.
2. $\langle a\rangle_j \leftarrow a$.
3. For $k = 1, \dots, n$ do:[a]
   1. P_i picks a uniformly random $r \in_R \mathbb{F}$ and sends it privately to P_j.
   2. Execute $\langle r\rangle_i \leftarrow r$ and $\langle r\rangle_j \leftarrow r$.
   3. P_k broadcasts a random challenge $e \in \mathbb{F}$.
   4. The parties execute $\langle s\rangle_i \leftarrow e\langle a\rangle_i + \langle r\rangle_i$ and $\langle s\rangle_j \leftarrow e\langle a\rangle_j + \langle r\rangle_j$.
   5. $s \leftarrow \langle s\rangle_i$ and $s' \leftarrow \langle s\rangle_j$ (we use s' here to indicate that the opened values may be different if P_i or P_j is corrupt).
   6. If $s \neq s'$, all players go to step 4.

   If all n iterations of the loop were successful, parties output `success`, except P_j who outputs a.
4. This is the "exception handler". It is executed if it is known that P_j or P_i is corrupt. First execute $a \leftarrow \langle a\rangle_i$. If this fails, all parties output `fail`. Otherwise do $\langle a\rangle_j \Leftarrow a$ (this cannot fail). Parties output `success`, except P_j who outputs a.

[a] The description of this step assumes that $1/|\mathbb{F}|$ is negligible in the security parameter. If that is not the case, we repeat the step u times in parallel, where u is chosen such that $(1/|\mathbb{F}|)^u$ is negligible.

5.2.1 *Implementations of the $transfer$ Command*

We will present two implementations of this command. One is only statistically secure but makes no assumption on t, the number of corrupted players, while the other is perfectly secure but requires $t < n/2$.

The basic idea is in both cases that $\langle a\rangle_i$ will be opened to P_j only, and then P_j commits to a. If P_j is corrupt and fails to do a commitment, then we make a

Protocol Perfect Transfer

If any command fails in Steps 1-4, it will be clear that either P_i or P_j is corrupt. In this case, go to Step 5

1. $(\mathsf{P}_j)a \leftarrow \langle a\rangle_i$.
2. $\langle a\rangle_j \leftarrow a$.
3. To check that P_i and P_j are committed to the same value, we do the following: P_i selects a polynomial $f(\mathtt{X}) = a + \alpha_1\mathtt{X} + \cdots + \alpha_t\mathtt{X}^t$, for uniformly random $\alpha_1, \ldots, \alpha_t$, and executes $\langle \alpha_l\rangle_i \leftarrow \alpha_l$ for $l = 1, \ldots, t$. We invoke the `add` and `mult` commands so all players collaborate to compute, for $k = 1, \ldots, n$:
$$\langle f(k)\rangle_i = \langle a\rangle_i + k\langle \alpha_1\rangle_i + \cdots + k^t\langle \alpha_t\rangle_i$$
and then $(\mathsf{P}_k)f(k) \leftarrow \langle f(k)\rangle_i$.
Then P_i sends $\alpha_1, \ldots, \alpha_t$ privately to P_j and we execute $\langle \alpha_l\rangle_j \leftarrow \alpha_l$ for $l = 1, \ldots, t$. In the same way as above, we compute $\langle f(k)\rangle_j$ and execute $(\mathsf{P}_k)f(k) \leftarrow \langle f(k)\rangle_j$.
4. For $k = 1, \ldots, n$: P_k compares the two values that were opened towards him. If they agree, he broadcasts `accept`, else he broadcasts `reject`.
For every P_k who said reject, execute $a_k \leftarrow \langle f(k)\rangle_i$ and $a'_k \leftarrow \langle f(k)\rangle_j$. If $a_k = a'_k$ for all k, all players output `success`, except P_j who outputs a. Otherwise go to step 5.
5. This is the "exception handler". It is executed if it is known that P_j or P_i is corrupt. First execute $a \leftarrow \langle a\rangle_i$. If this fails, all parties output `fail`. Otherwise do $\langle a\rangle_j \Leftarrow a$ (this cannot fail). Parties output `success`, except P_j who outputs a.

public and force a commitment to a for P_j. This is secure, as when P_j is corrupted, then a becomes know to the adversary in the ideal world also (as a is output to P_j). So, the simulator learns a and can simulate the protocol simply by running it on the correct a.

This is, however, not sufficient. If P_j is corrupted it could do $\langle a'\rangle_j \leftarrow a'$ for $a' \neq a$. This cannot be solved by doing $(\mathsf{P}_i)a' \leftarrow \langle a'\rangle_j$ and let P_i check that $a' = a$, because P_i could be corrupted as well. The ideal functionality ensures that $a' = a$ even if P_i and P_j are corrupted, therefore this should hold also for the protocol.

The final step in both protocols is therefore that P_i and P_j prove to the other parties that $a' = a$.

We sketch the argument why Protocol Transfer is secure: It is clear that if P_i and P_j are honest, then all proofs will be accepting. Second, if P_i and P_j are honest, then r is random and the only value leaked to the other parties is $ea + r$, which is just a uniformly random field element. This is secure, as the simulator can simulate $ea + r$ using a uniformly random value. If either P_i or P_j is corrupted, there is nothing to simulate as the simulator learns a in the ideal world. What remains is to argue if that if $a' \neq a$, then the proof will fail with high probability. This ensures that a case where $a' \neq a$, but still parties output `success`, will occur with only negligible probability. This is necessary, as such a case would constitute a chance to distinguish between the ideal case and the protocol execution.

So, assume that $a' \neq a$. That is, P_j made a commitment $\langle a + \Delta_a\rangle_j$ for $\Delta_a \neq 0$.

Then P_i and P_j makes commitments $\langle r \rangle_i$ respectively $\langle r + \Delta_r \rangle_i$. Again P_j could pick $\Delta_r \neq 0$, but could also use $\Delta_r = 0$ — we do not know. We do however know that $s = ea + r$ and that $s' = e(a + \Delta_a) + (r + \Delta_r) = s + (e\Delta_a + \Delta_r)$. Therefore the proof is accepted if and only if $e\Delta_a + \Delta_r = 0$, which is equivalent to $e = \Delta_a^{-1}(-\Delta_r)$ (recall that $\Delta_a \neq 0$ and thus invertible.) Since e is uniformly random when P_k is honest and picked after Δ_a and Δ_r are fixed, the probability that $e = \Delta_a^{-1}(-\Delta_r)$ is exactly $1/|\mathbb{F}|$. If that is not negligible the whole process can be repeated a number of times in parallel as mentioned in the protocol description.

Note that if the proof fails, it could be due to P_j being corrupted, so we run the "exception handler" to give P_i a chance to reveal a and let the other parties do a forced commitment of P_j to a.

In Protocol Perfect Transfer, the idea is to check in a different way that the values committed by P_i and P_j are the same, namely we ask them to commit also to two polynomials that should be the same, furthermore these polynomial both determine the value a in question. We then check that the two polynomials agree in so many points that they must be equal and this in turn implies that the values committed by P_i and P_j are indeed the same. A more detailed argument can be found in the proof of Theorem 5.1.

Protocol Commitment Multiplication

If any command used during this protocol fails, all players output `fail`.

```
1. $\mathsf{P}_i : \langle c \rangle_i \leftarrow ab$.
2. For $k = 1, \dots, n$, do:[a]
    1. $\mathsf{P}_i$ chooses a uniformly random $\alpha \in \mathbb{F}$.
    2. $\langle \alpha \rangle_i \leftarrow \alpha$; $\langle \gamma \rangle_i \leftarrow \alpha b$.
    3. $\mathsf{P}_k$ broadcasts a uniformly random challenge $e \in \mathbb{F}$.
    4. $\langle A \rangle_i \leftarrow e\langle a \rangle_i + \langle \alpha \rangle_i$; $A \leftarrow \langle A \rangle_i$.
    5. $\langle D \rangle_i \leftarrow A\langle b \rangle_i - e\langle c \rangle_i - \langle \gamma \rangle_i$; $D \leftarrow \langle D \rangle_i$.
    6. If $D \neq 0$ all players output fail.
3. If we arrive here, no command failed during the protocol and all $D$-values were 0. All
   players output success.
```

[a] The description of this step assumes that $1/|\mathbb{F}|$ is negligible in the security parameter. If that is not the case, we repeat the step u times in parallel, where u is chosen such that $(1/|\mathbb{F}|)^u$ is negligible.

5.2.2 Implementations of the `mult` Command

For the `mult`-command, we will follow a similar pattern as for `transfer`: we present an implementation that is only statistically secure but makes no assumption on t, the number of corrupted players, while the other implementation is perfectly secure but requires $t < n/3$.

The idea in Protocol Commitment Multiplication is that P_i commits to what he claims is the product ab and then proves to each other player P_k that this was done correctly. It is easy to show that if P_i remains honest, then the proof succeeds and all values opened are random (or fixed to 0). So they reveal no extra information to the adversary and are easy to simulate. Using an analysis similar to the one for Protocol Transfer, it can be shown that if $c = ab + \Delta$ for $\Delta \neq 0$, then for each P_k the proof fails except with probability $1/|\mathbb{F}|$.

Protocol Perfect Commitment Multiplication

If any command used during this protocol fails, all players output `fail`.

1. $\langle c \rangle_i \leftarrow ab$.
2. P_i chooses polynomials $f_a(\mathtt{X}) = a + \alpha_1 \mathtt{X} + \cdots + \alpha_t \mathtt{X}^t$ and $f_b(\mathtt{X}) = b + \beta_1 \mathtt{X} + \cdots + \beta_t \mathtt{X}^t$, for random α_j, β_j. He computes $h(\mathtt{X}) = f_a(\mathtt{X}) f_b(\mathtt{X})$, and writes $h(\mathtt{X})$ as $h(\mathtt{X}) = c + \gamma_1 \mathtt{X} + \cdots + \gamma_{2t} \mathtt{X}^{2t}$. Then establish commitments as follows:
$$\langle \alpha_j \rangle_i \leftarrow \alpha_j \text{ for } j = 1, \ldots, t$$
$$\langle \beta_j \rangle_i \leftarrow \beta_j \text{ for } j = 1, \ldots, t$$
$$\langle \gamma_j \rangle_i \leftarrow \gamma_j \text{ for } j = 1, \ldots, 2t$$
3. The `add` and `mult` commands are invoked to compute, for $k = 1, \ldots, n$:
$$\langle f_a(k) \rangle_i = \langle a \rangle_i + k \langle \alpha_1 \rangle_i + \cdots + k^t \langle \alpha_t \rangle_i$$
$$\langle f_b(k) \rangle_i = \langle b \rangle_i + k \langle \beta_1 \rangle_i + \cdots + k^t \langle \beta_t \rangle_i$$
$$\langle h(k) \rangle_i = \langle b \rangle_i + k \langle \gamma_1 \rangle_i + \cdots + k^{2t} \langle \gamma_{2t} \rangle_i$$
Then $(\mathsf{P}_k) a_k \leftarrow \langle f_a(k) \rangle_i$, $(\mathsf{P}_k) b_k \leftarrow \langle f_b(k) \rangle_i$ and $(\mathsf{P}_k) c_k \leftarrow \langle h(k) \rangle_i$ are executed.
4. For $k = 1, \ldots, n$, do: P_k checks that $a_k b_k = c_k$, and broadcasts `accept` if this is the case, else broadcast `reject`.
5. For each P_k who said `reject`, do $a_k \leftarrow \langle f_a(k) \rangle_i$, $b_k \leftarrow \langle f_b(k) \rangle_i$ and $c_k \leftarrow \langle h(k) \rangle_i$, all players check whether $a_k b_k = c_k$ holds. If this is not the case, all players output `fail`.
6. If we arrive here, no command failed during the protocol and all relations $a_k b_k = c_k$ that were checked, hold. All players output `success`.

The idea in Protocol Perfect Commitment Multiplication is to have P_i commit to polynomials f_a, f_b (determining a, b) and to what he claims is the product $h = f_a f_b$ of the polynomials. Then players check that $h(k) = f_a(k) f_b(k)$ in a number of points sufficient to guarantee that indeed $h = f_a f_b$. A more detailed argument can be found in the proof of Theorem 5.1.

Let $\pi_{\mathtt{TRANSFER,MULT}}$ be the protocol which implements the basic commands by simply relaying the commands to $\mathsf{F}_{\mathtt{COM\text{-}SIMPLE}}$ and which implements the transfer and multiplication commands by running Protocol Transfer respectively Protocol Commitment Multiplication on $\mathsf{F}_{\mathtt{COM\text{-}SIMPLE}}$. Let $\pi_{\mathtt{PTRANSFER,PMULT}}$ be the protocol which implements the basic commands by relaying the commands to $\mathsf{F}_{\mathtt{COM\text{-}SIMPLE}}$ and which implements the transfer and multiplication commands by running Protocol Perfect Transfer respectively Protocol Perfect Com-

MITMENT MULTIPLICATION on $\mathsf{F}_{\texttt{COM-SIMPLE}}$.[1] We can then summarize the protocols and results we have shown so far as follows:

THEOREM 5.1 *Let $\mathsf{F}_{\texttt{COM-SIMPLE}}$ be a functionality that has all the commands of $\mathsf{F}_{\texttt{COM}}$, except the advanced manipulation commands. Then $\pi_{\texttt{TRANSFER,MULT}} \diamond \mathsf{F}_{\texttt{SC}} \diamond \mathsf{F}_{\texttt{COM-SIMPLE}}$ implements $\mathsf{F}_{\texttt{COM}}$ in* $\mathrm{Env}^{t,sync}$ *with statistical security for all $t < n$. Moreover, $\pi_{\texttt{PTRANSFER,PMULT}} \diamond \mathsf{F}_{\texttt{SC}} \diamond \mathsf{F}_{\texttt{COM-SIMPLE}}$ implements $\mathsf{F}_{\texttt{COM}}$ in* $\mathrm{Env}^{t,sync}$ *with perfect security for all $t < n/3$.*

PROOF We prove here in detail the case for perfect security. The case of statistical security can be proved along the same lines, based on the arguments we gave above. This is straightforward (but quite tedious) and is left to the reader.

The first step is constructing the simulator $\mathcal{S}_{\texttt{COM}}$. On a high level, the idea is that the simulator will relay messages connected to the basic commitment commands while for `transfer` and `mult`, it will run internally a copy of $\pi_{\texttt{PTRANSFER,PMULT}} \diamond \mathsf{F}_{\texttt{SC}} \diamond \mathsf{F}_{\texttt{COM-SIMPLE}}$, where it uses its instance of $\mathsf{F}_{\texttt{COM-SIMPLE}}$ for basic commitment commands related to executing $\pi_{\texttt{PTRANSFER,PMULT}}$. Here, the simulator will as usual play the roles of the honest parties.

We see that the distribution produced by $\mathcal{S}_{\texttt{COM}}$ is the same as in the real case: Since messages concerning the basic commands are just relayed between the environment and $\mathsf{F}_{\texttt{COM}}$, this is exactly the same. During `transfer` if either P_i or P_j is corrupt, the simulator learns the value to be transferred. Similarly in `mult`, if P_i is corrupt, the simulator knows the values to be multiplied. Hence in these cases the protocol $\pi_{\texttt{PTRANSFER,PMULT}}$ is run on the true values. If, on the other hand, the players are honest then for both `transfer` and `mult` the environment sees at most t points of polynomials of degree at least t. Thus, no information on the actual values is given.

What remains to argue is that the protocol execution is consistent with the inputs/outputs to/from $\mathsf{F}_{\texttt{COM}}$. That is, if the protocol finishes successfully, then the output is correct, meaning that the players have indeed transferred a commitment or multiplied values correctly. On the other hand, if the simulated protocol fails then this is also consistent with the state of $\mathsf{F}_{\texttt{COM}}$.

First during `transfer`, if P_i and P_j are honest it is clear that all the proofs will be accepting. Moreover, if one or both are corrupted but we still reach the point where values are compared, no command failed earlier. At this point we know we have strictly more than $2t$ honest players. Hence, if at most t players said `reject` then at least $t+1$ honest players said `accept` which implies that indeed P_i and P_j are committed to the same polynomials. In particular, their degree-0 terms are also the same, so it follows that $\langle a \rangle_i$ and $\langle a \rangle_j$ do contain the same value.

As for `mult`, it is clear from the way the polynomials are committed to, that even if P_i is corrupt, we have $f_a(0) = a, f_b(0) = b, h(0) = c$, and that $\deg(f_a), \deg(f_b) \leq t$ and $\deg(h) \leq 2t$. Moreover, if the players output `success` at the end, it follows that $f_a(k)f_b(k) = h(k)$ for every player P_k who remains honest.

[1] The protocols for advanced commands should pick commitment identifiers for intermediary values as to not collide with the identifiers used by the basic commands.

Agent $\mathcal{S}_{\texttt{COM}}$

A simulator for $\pi_{\texttt{PTRANSFER,PMULT}} \diamond \mathsf{F}_{\text{SC}} \diamond \mathsf{F}_{\texttt{COM-SIMPLE}}$ for the static case.

Basic Commands: When any of the basic commands are executed, the simulator will just forward messages back and forth between the environment and $\mathsf{F}_{\texttt{COM}}$[a]. When a corrupted player commits to a value, $\mathcal{S}_{\texttt{COM}}$ will store it.

Transfer: When $\mathsf{F}_{\texttt{COM}}$ is executing a `transfer` of some value a from P_i to P_j, then P_i has already committed to that value. Therefore, if P_i is corrupted, the simulator has stored a. On the other hand, if P_j is corrupt, then a is leaked from $\mathsf{F}_{\texttt{COM}}$ as soon as `transfer` is initiated. Now, $\mathcal{S}_{\texttt{COM}}$ can run its copy of $\pi_{\texttt{PTRANSFER}} \diamond \mathsf{F}_{\text{SC}} \diamond \mathsf{F}_{\texttt{COM-SIMPLE}}$ with the environment using as input the correct value a, and where it simulates the parts of the honest players.

If the simulated protocol terminates correctly, the simulator inputs $(\texttt{transfer}, i, cid, j)$ and asks for delivery of outputs on $\texttt{COM.infl}$. If P_i is corrupt and makes some command fail, the simulator will ask for for delivery of outputs on without inputting $(\texttt{transfer}, i, cid, j)$, which will make $\mathsf{F}_{\texttt{COM}}$ output fail, matching the real scenario.

If both parties are honest, then the transfer protocol is simulated with some random value.

Multiplication: When $\mathsf{F}_{\texttt{COM}}$ is executing a `mult` for P_i of some values a, b, then P_i has already committed to these values. Hence, if P_i is corrupted, the simulator has already stored a, b and can do $\pi_{\texttt{PMULT}} \diamond \mathsf{F}_{\text{SC}} \diamond \mathsf{F}_{\texttt{COM-SIMPLE}}$ with the correct values. If P_i is honest, $\pi_{\texttt{PMULT}} \diamond \mathsf{F}_{\text{SC}} \diamond \mathsf{F}_{\texttt{COM-SIMPLE}}$ is done with random values.

If the simulated protocol terminates correctly, the simulator inputs $(\texttt{mult}, i, cid_1, cid_2, cid_3)$ and asks for delivery of outputs on $\texttt{COM.infl}$. If P_i is corrupt and makes some command fail, the simulator will ask for for delivery of outputs on without inputting $(\texttt{mult}, i, cid_1, cid_2, cid_3)$, which will make $\mathsf{F}_{\texttt{COM}}$ output fail, matching the real scenario.

[a] Note here, that this does *not* include the basic commands needed for executing $\pi_{\texttt{PTRANSFER,PMULT}}$. This is only for the basic commands that $\mathsf{F}_{\texttt{COM}}$ will execute.

The assumption that $t < n/3$ implies that there are at least $2t+1$ honest players. Since $2t+1$ points determine any polynomial of degree at most $2t$ uniquely, it follows that $f_a(\texttt{X})f_b(\texttt{X}) = h(\texttt{X})$, and hence that $ab = f_a(0)f_b(0) = h(0) = c$. It is also easy to check that if P_i remains honest, the protocol will always be successful.

Finally, if the simulated protocol fails at some point because of a corrupted P_i the simulator outputs `fail` which is consistent with the internal state of $\mathsf{F}_{\texttt{COM}}$ since, in this case, $\mathcal{S}_{\texttt{COM}}$ also makes $\mathsf{F}_{\texttt{COM}}$ fail.

Adaptive Corruption

We now show how to modify $\mathcal{S}_{\texttt{COM}}$ to handle adaptively corrupted players. Hence, we allow a player P_k to be corrupted in the beginning of any round during the protocol. If this happens, the simulator learns the true values of P_k's inputs and then it has to show the internal state of P_k to the environment. For this, the simulator must patch the state so that it is consistent with the rest of the values shown to environment during the execution.

After creating a correctly patched state of the simulation we give the state of the newly corrupted P_k to the environment and then simulation continues exactly as described in $\mathcal{S}_{\texttt{COM}}$.

We describe how to patch up for the advanced commands. The basic commands are executed by an ideal functionality, so nothing needs to be handled there.

First assume a player different from P_i or P_j for `transfer` or different from P_i for `mult` becomes corrupted. Here, nothing needs to be patched, the simulator simply shows the environment the values that were used in the execution. This is perfectly fine, since for both `transfer` and `mult`, the environment has seen less than t points of polynomials of degree at least t. Therefore, showing the set of points which was used is still consistent with what has already been shown, and since at most t points are shown for each polynomial, still no information is given on the actual inputs to the command.

Similarly for `transfer`, no patching is needed if at a point where one of P_i and P_j is corrupted, the other player gets corrupted. The protocol $\pi_{\texttt{PTRANSFER}}$ was run with the correct value, so the simulator simply shows the environment the values that were used in the execution.

However, if in `mult`, P_i gets corrupted or in `transfer` either P_i or P_j gets corrupted where before both were honest, then the state needs to be patched. When the player becomes corrupted the simulator will learn the true value of the player's input. That is, for `transfer`, $\mathcal{S}_{\texttt{COM}}$ will learn the true value to be transferred and for `mult`, $\mathcal{S}_{\texttt{COM}}$ will learn the values to be multiplied.

Now $\mathcal{S}_{\texttt{COM}}$ patches the state by recomputing the steps of the protocol with the true values. For the steps that involve secret sharing, $\mathcal{S}_{\texttt{COM}}$ recomputes new polynomials which are consistent with the degree-0 term equaling the true values and the (strictly less than t) shares already shown to the environment. For example for `transfer` this means that the polynomial $f(X)$ from step 3 is recomputed such that it is consistent with the shares already shown to the environment and $f(0) = a$, where a is the true value to be transferred. Note that this patching can easily be done even if a player gets corrupted in the middle of a command. The steps are simply recomputed up the point where the player gets corrupted.

$\square$

5.3 A Secure Function Evaluation Protocol for Active Adversaries

In this section we show how to implement $\mathsf{F}^f_{\texttt{SFE}}$, given access to $\mathsf{F}_{\texttt{SC}}$ and $\mathsf{F}_{\texttt{COM}}$. The basic idea is to do the same as in our passively secure protocol, but in addition make sure that all parties are committed to all shares they hold.

We note for future reference that the definition of $\mathsf{F}_{\texttt{COM}}$ implies that if one of its commands fails, this is always because a particular corrupt player has caused this to happen by not providing the correct input. Therefore the honest players can always conclude from failure of a command that some particular player is corrupt.

We will need the following notation for $a \in \mathbb{F}$ and f a polynomial of degree at

Protocol CEAS (Circuit Evaluation with Active Security)

The protocol assumes that $t < n/3$ and proceeds in three phases: the input sharing, computation and output reconstruction phases.

Input Sharing: Each player P_i holding input $x_i \in \mathbb{F}$ distributes $[\![x_i; f_{x_i}]\!]_t$. If this fails, P_i is corrupt, so we will instead assign 0 as default input for P_i: execute $\langle 0 \rangle_k \Leftarrow 0$ for $k = 1, \dots, n$, thus creating $[\![0; o]\!]_t$ where o is the zero-polynomial.

We then go through the circuit and process the gates one by one in the computational order. Just after the input sharing phase, we consider all input gates as being processed. We will maintain the following:

Invariant: Consider an input or an output wire for any gate, and let $a \in \mathbb{F}$ be the value assigned to this wire. Then, if the gate has been processed, the players hold $[\![a; f_a]\!]_t$ for a polynomial f_a.

We then continue with the last two phases of the protocol:

Computation Phase: Repeat the following until all gates have been processed (then go to the next phase): Consider the first gate in the computational order that has not been processed yet. According to the type of gate, do one of the following

Addition gate: The players hold $[\![a; f_a]\!]_t, [\![b; f_b]\!]_t$ for the two inputs a, b to the gate. The players compute $[\![a + b; f_a + f_b]\!]_t = [\![a; f_a]\!]_t + [\![b; f_b]\!]_t$.[a]

Multiply-by-constant gate: The players hold $[\![a; f_a]\!]_t$ for the input a to the gate. The players compute $[\![\alpha a; \alpha f_a]\!]_t = \alpha[\![a; f_a]\!]_t$.

Multiplication gate: The players hold $[\![a; f_a]\!]_t, [\![b; f_b]\!]_t$ for the two inputs a, b to the gate.

1. The players compute $[\![ab; f_a f_b]\!]_{2t} = [\![a; f_a]\!]_t * [\![b; f_b]\!]_t$. Note that this involves each P_j committing to $f_a(j) f_b(j)$ and proving that this was done correctly. So this may fail for up to t players.
2. Define $h \overset{\text{def}}{=} f_a f_b$. Then $h(0) = f_a(0) f_b(0) = ab$ and the parties hold $[\![ab; h]\!]_{2t}$, where P_i is committed by $\langle h(i) \rangle_i$. Each P_i distributes $[\![h(i); f_i]\!]_t$ from $\langle h(i) \rangle_i$. Also this step may fail for up to t players.
3. Let S be the indices of at the (at least $n - t$) players for which the previous two steps did not fail. Note that since $t < n/3$, $\deg(h) = 2t < n - t$. Let $\mathbf{r}_S$ be a recombination vector for S, that is, a vector $\mathbf{r}_S = (r_1, \dots, r_n)$ for which it holds that $h(0) = \sum_{i \in S} r_i h(i)$ for any polynomial h of degree $\leq 2t$. The players compute

$$\sum_{i \in S} r_i [\![h(i); f_i]\!]_t = [\![\sum_{i \in S} r_i h(i); \sum_{i \in S} r_i f_i]\!]_t$$
$$= [\![h(0); \sum_{i \in S} r_i f_i]\!]_t = [\![ab; \sum_{i \in S} r_i f_i]\!]_t \ .$$

Output Reconstruction: At this point all gates, including the output gates have been processed. So do the following for each output gate (labeled i): The players hold $[\![y; f_y]\!]_t$ where y is the value assigned to the output gate. Now we open $[\![y; f_y]\!]_t$ towards P_i.

[a] Whenever we say that the parties compute a command, we implicitly say that they wait until $\mathsf{F}_{\mathtt{COM}}$ returns `success` or `fail` before they proceed with the next command.

most t over $\mathbb{F}$:

$$[\![a; f]\!]_t = (\langle f(1)\rangle_1, \ldots, \langle f(n)\rangle_n) \ .$$

Thus this notation refers to n variables held by $\mathsf{F}_{\mathtt{COM}}$ and claims that the i'th variable contains the appropriate value $f(i)$, owned by P_i.

When we say that P_i distributes $[\![a; f_a]\!]_t$, this means the following: P_i chooses a polynomial $f_a(\mathtt{X}) = a + \alpha_1\mathtt{X} + \cdots + \alpha_t\mathtt{X}^t$ with $\alpha_j \in \mathbb{F}$ at random, then P_i commits to the coefficients of f_a, the addition and multiplication commands are used to compute a commitment to $f_a(k)$ owned by P_i, for $k = 1, \ldots, n$, and finally this commitment is transferred to P_k. More formally, the following sequence of commands of $\mathsf{F}_{\mathtt{COM}}$ are executed:

1. $\langle a\rangle_i \leftarrow a$, $\langle \alpha_j\rangle_i \leftarrow \alpha_j, j = 1, \ldots, t$.
2. $\langle f_a(k)\rangle_i \leftarrow \langle a\rangle_i + \sum_{j=1}^{t} k^j\langle \alpha_j\rangle_i, k = 1, \ldots, n$
3. $\langle f_a(k)\rangle_k \leftarrow \langle f_a(k)\rangle_i, k = 1, \ldots, n$

It is very easy to see that when P_i distributes $[\![a; f_a]\!]_t$, this does in fact create the object $[\![a; f_a]\!]_t$, or all players agree that one of the commands failed, in which case they can conclude that P_i is corrupt. Note that this fact is not a result of proving the above sequence of commands secure as a protocol, it simply follows by definition of $\mathsf{F}_{\mathtt{COM}}$. Above, we said that P_i should commit to a as a part of the process. But P_i could instead use an already existing commitment $\langle a\rangle_i$ to some value a. In this case, we say that P_i distributes $[\![a; f_a]\!]_t$ from $\langle a\rangle_i$. Again by definition of $\mathsf{F}_{\mathtt{COM}}$, it is guaranteed that the new object determines the same value that the existing commitment contains.

Distributing $[\![a; f_a]\!]_t$ is actually a concept known from the literature as Verifiable Secret Sharing: a dealer (here P_i) distributes shares of a secret to the players in such a way that the honest players are guaranteed to get consistent shares of a well defined secret, or they will agree that the dealer failed. Moreover, the correct secret can always be reconstructed, even if the dealer does not participate. This is easily seen to be the case for $[\![a; f_a]\!]_t$ if $t < n/2$: In the following, when we say that $[\![a; f_a]\!]$ is opened, this means we execute

$$f_a(k) \leftarrow \langle f_a(k)\rangle_k, \text{ for } k = 1, \ldots, n \ .$$

This will result in up to t failures, but also in at least $n - t > t$ correct values, from which all players can compute a by Lagrange interpolation (see Section 3.2). We can also open $[\![a; f_a]\!]$ towards P_j. This means we execute $(\mathsf{P}_j) f_a(k) \leftarrow \langle f_a(k)\rangle_k$, for $k = 1, \ldots, n$, so that only P_j learns a.

We can also do arithmetic on objects of form $[\![a; f_a]\!]_t$, in much the same way as we did for $[a, f_a]_t$ when we did passively secure protocols. Now, however we have to do arithmetic on committed values. Hence, for $\alpha, a, b \in \mathbb{F}$ and f_a, f_b polynomials of degree at most t, we define $[\![a; f_a]\!]_t + [\![b; f_b]\!]_t$, $\alpha[\![a; f_a]\!]_t$ and $[\![a; f_a]\!]_t * [\![b; f_b]\!]_t$ to denote sequences of commands from $\mathsf{F}_{\mathtt{COM}}$, as follows

- $[\![a; f_a]\!]_t + [\![b; f_b]\!]_t$ means that we execute $\langle f_a(k) + f_b(k)\rangle_k \leftarrow \langle f_a(k)\rangle_k + \langle f_b(k)\rangle_k$ for $k = 1, \ldots, n$.

- $\alpha[\![a; f_a]\!]_t$ means that we execute $\langle \alpha f_a(k)\rangle_k \leftarrow \alpha\langle f_a(k)\rangle_k$ for $k = 1, \dots, n$.
- $[\![a; f_a]\!]_t * [\![b; f_b]\!]_t$ means that we execute $\langle f_a(k) f_b(k)\rangle_k \leftarrow \langle f_a(k)\rangle_k \langle f_b(k)\rangle_k$ for $k = 1, \dots, n$.

Again by definition of $\mathsf{F}_{\texttt{COM}}$, it is easy to see that these command sequences create objects of the form we naturally would expect, namely $[\![a + b; f_a + f_b]\!]_t$, $[\![\alpha a; \alpha f_a]\!]_t$, and $[\![ab; f_a f_b]\!]_{2t}$, respectively, again unless one of the commands fail. Therefore, by a slight abuse of notation, in the following we will write:

$$[\![a + b; f_a + f_b]\!]_t = [\![a; f_a]\!]_t + [\![b; f_b]\!]_t,$$

$$[\![\alpha a; \alpha f_a]\!]_t = \alpha[\![a; f_a]\!]_t,$$

$$[\![ab; f_a f_b]\!]_{2t} = [\![a; f_a]\!]_t * [\![b; f_b]\!]_t$$

to denote that we execute the command sequence on the right side to create the object on the left side.

The notation we have defined is used in Protocol CEAS, whose security follows almost directly from the security of the passive secure protocol. In particular, since the manipulation commands of $\mathsf{F}_{\texttt{COM}}$ are used, all parties compute all shares exactly as in the passively secure protocol, and $\mathsf{F}_{\texttt{COM}}$ keeps the shares secret. Therefore, as we see below, essentially the same simulator works in this case as well. Letting $\pi_{\texttt{CEAS}}$ denote Protocol CEAS, we have

Theorem 5.2 *$\pi^f_{\texttt{CEAS}} \diamond \mathsf{F}_{\texttt{SC}} \diamond \mathsf{F}_{\texttt{COM}}$ implements $\mathsf{F}^f_{\texttt{SFE}}$ in* $\mathrm{Env}^{t,sync}$ *with perfect security for any f and for all $t < n/3$.*

Proof We construct a simulator for $\pi^f_{\texttt{CEAS}} \diamond \mathsf{F}_{\texttt{SC}} \diamond \mathsf{F}_{\texttt{COM}}$ where the idea is very similar to the simulator, $\mathcal{S}_{\texttt{CEPS}}$, for the passive case. The simulator runs a copy of $\pi^f_{\texttt{CEAS}} \diamond \mathsf{F}_{\texttt{SC}} \diamond \mathsf{F}_{\texttt{COM}}$ where it plays the role of the honest parties and executes the protocol with the environment which is controlling the actively corrupted players. Since the simulator is running $\mathsf{F}_{\texttt{COM}}$ it is able to keep track of all the corrupted parties' values. We present first the simulator Agent $\mathcal{S}_{\texttt{CEAS}}$ for the static case, where the environment provides inputs to all parties in the same round. The case where not all parties receive inputs in the same round is handled as in the passive case. After arguing static security, we show how to modify the simulator to achieve adaptive security as well.

Perfect security is argued similarly to the passive case. First, the shares sent by honest players to corrupt players in the input distribution and computation phase are distributed exactly as in the simulation, namely t uniformly random shares for each shared value. Furthermore, since the commands of $\mathsf{F}_{\texttt{COM}}$ are used, the shares computed by local computation are the same as in the passively secure protocol. This follows from the definition of $\mathsf{F}_{\texttt{COM}}$ which guarantees that either the players are committed to the correct shares or they all agree that a command failed because a particular player P_i is corrupt. However, this can happen for at most t players which we by construction can tolerate when $t < n/3$. Therefore, the invariant in the protocol holds, which implies that the output is correct and

Agent $\mathcal{S}_{\texttt{CEAS}}$

A simulator for $\pi^f_{\texttt{CEAS}} \diamond \mathsf{F}_{\texttt{SC}} \diamond \mathsf{F}_{\texttt{COM}}$ for the static case where the environment gives inputs to all parties in the same round.

Initialize: In the preamble, the environment $\mathcal{Z}$ specifies the set C of actively corrupted parties. After this, the simulator sets up a copy of $\pi^f_{\texttt{CEAS}} \diamond \mathsf{F}_{\texttt{SC}} \diamond \mathsf{F}_{\texttt{COM}}$ where it actively corrupts the players in C and connects the special ports of $\pi^f_{\texttt{CEAS}} \diamond \mathsf{F}_{\texttt{SC}} \diamond \mathsf{F}_{\texttt{COM}}$ to $\mathcal{Z}$. $\mathcal{S}_{\texttt{CEAS}}$ also does active corruptions of $\mathsf{P}_i \in C$ on $\mathsf{F}^f_{\texttt{SFE}}$, which means that $\mathsf{F}^f_{\texttt{SFE}}$ will ignore all inputs on $\texttt{SFE.in}_i$ and stop giving outputs on $\texttt{SFE.out}_i$. Instead the simulator must now provide the input to $\mathsf{F}^f_{\texttt{SFE}}$ on behalf of the corrupt players.

Input Sharing: The simulator performs the steps of input sharing with the environment (which runs the corrupted parties). That is, $\mathcal{S}_{\texttt{CEAS}}$ runs the honest parties, so it will for each $\mathsf{P}_i \notin C$ distribute a sharing of 0.
When the corrupt parties share their inputs the simulator will receive all the values they want to commit to (since the simulator is running $\mathsf{F}_{\texttt{COM}}$). In particular $\mathcal{S}_{\texttt{CEAS}}$ will for each $\mathsf{P}_j \in C$ receive x_j which it can then pass on to $\mathsf{F}^f_{\texttt{SFE}}$.

Computation Phase: In the computation phase the simulator simply performs the protocol steps of $\pi^f_{\texttt{CEAS}}$ together with the environment according to the gates that need to be processed.

Output Reconstruction: First $\mathcal{S}_{\texttt{CEAS}}$ instructs $\mathsf{F}^f_{\texttt{SFE}}$ to evaluate the function, thereby learning the outputs y_j for all $\mathsf{P}_j \in C$.
For each output of a corrupted party P_j, it takes the t shares that the corrupted parties hold, it defines $f_{y_j}(0) \overset{\text{def}}{=} y_j$, and if $t' < t$ players are corrupted, then for $t - t'$ honest players P_i i sets $f_{y_j}(i)$ to be a random value. After this, $f_{y_j}(\texttt{X})$ is defined in $t+1$ points and hence $\mathcal{S}_{\texttt{CEAS}}$ can compute the unique degree-t polynomial consistent with these points. Then, for $\mathsf{P}_i \notin C$ it sets the committed share of P_i to be $f_{y_j}(i)$ and opens it to P_j on behalf of P_i.
When $\mathcal{Z}$ clocks out P_i in the copy of $\pi^f_{\texttt{CEAS}} \diamond \mathsf{F}_{\texttt{SC}} \diamond \mathsf{F}_{\texttt{COM}}$ so that P_i should produce an output, the simulator first inputs ($\texttt{delivery-round}$) to $\mathsf{F}^f_{\texttt{SFE}}$ if it has not done so already. Then $\mathcal{S}_{\texttt{CEAS}}$ inputs ($\texttt{clock-out}, i$) to $\mathsf{F}^f_{\texttt{SFE}}$ to make it output y_i on $\texttt{SFE.out}_i$, exactly as P_i would have done in the protocol. (Since the environment is running the corrupted parties this will only happen for the honest parties.)

hence *the same* in both $\pi^f_{\texttt{CEAS}} \diamond \mathsf{F}_{\texttt{SC}} \diamond \mathsf{F}_{\texttt{COM}}$ and $\mathsf{F}^f_{\texttt{SFE}}$. Furthermore, $\mathcal{S}_{\texttt{CEAS}}$ makes $\mathsf{F}^f_{\texttt{SFE}}$ give output on $\texttt{SFE.out}_i$ at the same point when P_i would. Finally, note that in the case where less than $t' < t$ players are corrupted the simulator makes random choices for the remaining $t - t'$ shares, independently for each output. This matches the distribution of a real execution since each output comes from a multiplication gate (by the assumption on the circuit we made in Chapter 3). The protocol for a multiplication gate produces a random polynomial that determines the correct output from the gate. We conclude that the views produced by $\mathcal{S}_{\texttt{CEAS}}$ are distributed exactly as in the real case.

Adaptive Corruption

We now show how to modify $\mathcal{S}_{\texttt{CEAS}}$ such that it can handle adaptively corrupted players. Hence, we allow a player P_k to be corrupted in the beginning of any round during the protocol. If this happens, the simulator learns the true value of x_k and then it has to show the internal state of P_k to the environment. For this, the simulator must patch the state of P_k so that it is consistent with x_k and the rest of the values shown to environment during the execution. After creating a correctly patched state of the simulation we give the state of the newly corrupted P_k to the environment and then simulation continues exactly as described in $\mathcal{S}_{\texttt{CEAS}}$.

We describe here the case where P_k gets corrupted after the protocol is completed. Then, handling corruption at an earlier stage will simply consist of only doing a smaller part of the patching steps (namely up to the point where the player gets corrupted). Basically, the way to patch the state is to recompute every honest player's share which is affected by the input of P_k. This is done as follows [2]:

- *Input Sharing.* When $\mathcal{S}_{\texttt{CEAS}}$ learns x_k it can compute a new random secret sharing of x_k which is consistent with the (strictly less than t) shares shown to the environment and $f_{x_k}(0) = x_k$. From $f_{x_k}(\texttt{X})$ the simulator recomputes the steps of input distribution, not involving other corrupt parties. That is, first $\mathcal{S}_{\texttt{CEAS}}$ sets up the commitments to the new coefficients of $f_{x_k}(\texttt{X})$ (by modifying the values held by its internal copy of $\mathsf{F}_{\texttt{COM}}$). Then it sets up new commitments of shares to honest parties and transfers the commitments to them, obtaining $\langle f_{x_k}(i)\rangle_i$.
- *Addition or Multiplication by a constant.* Here the players only do local computations so $\mathcal{S}_{\texttt{CEAS}}$ can simply adjust the commitments of all honest parties' shares which were affected by $f_{x_k}(\texttt{X})$.
- *Multiplication.* The first round in the processing of a multiplication gate only has local computations and hence, the simulator adjusts as above commitments of the affected shares of honest parties. Then in the second round, new shares are distributed of the product of two polynomials $f_a f_b$. If $f_{x_k}(\texttt{X})$ is involved in one of these polynomials, $\mathcal{S}_{\texttt{CEAS}}$ must compute a new random secret sharing of $f_a(0)f_b(0)$ as it did for x_k in input sharing. In the third and last round the simulator is again able to recompute the shares of the honest parties by local computations. Recall that the recombination vector is independent of $f_a f_b(\texttt{X})$; it is the same for all polynomials of degree at most $n-t$, so it is indeed possible to redo this step with the new product.
- *Output Reconstruction.* We will get from the ideal functionality the output value y_k belonging to the player that was just corrupted. We then create shares for honest players such that the total set of shares determines y_k, exactly as in the output step of $\mathcal{S}_{\texttt{CEAS}}$, and list these shares as those sent to P_k in the round where he was to learn y_k.

  Now let C be the set of already corrupted players. For every y_j belonging

[2] It might be a good idea to have the protocol $\pi_{\texttt{CEAS}}$ in mind when reading this.

to a player $\mathsf{P}_j \in C$ note that the environment has already seen (points on) a polynomial f_{y_j} that $\mathcal{S}_{\texttt{CEAS}}$ created earlier. Therefore we must patch the state such that P_k ends up holding $f_{y_j}(k)$ as his share. Recall that f_{y_j} is produced by the multiplication gate protocol. Our patching procedure applied to this gate produces a polynomial f_i for each player, and then the local recombination step determines a polynomial $f = \sum_{i \in S} r_i f_i$. Since the patching never changes the shares of players in C, we have $f_{y_j}(i) = f(i)$ for each $\mathsf{P}_i \in C$. But we also need that $f_{y_j}(k) = f(k)$, and this is not guaranteed.

To correct for this, note that there must exist an honest player P_{i_0} such that $r_{i_0} \neq 0$, otherwise the corrupt players could reconstruct the product on their own. We now choose a random polynomial g of degree at most t, subject to

$$g(0) = 0, \quad g(i) = 0 \text{ for all } \mathsf{P}_i \in C, \text{ and } g(k) = r_{i_0}^{-1}(f_{y_j}(k) - f(k)).$$

This is possible since at most $t - 1$ players could be corrupted before P_k, so we fix the value of g in at most $t + 1$ points. We now adjust the state of (the simulator's copy of) P_{i_0}, such that we replace its polynomial f_{i_0} by $f_{i_0} + g$. This keeps the state of P_{i_0} internally consistent because $f_{i_0}(0) = (f_{i_0} + g)(0)$, and the shares of players in C are unchanged. However, we have now replaced f by $f + r_{i_0} g$, and clearly $(f + r_{i_0} g)(k) = f_{y_j}(k)$ as desired.

As mentioned, if a player gets corrupted earlier, patching is done only up to that point. Note that this is also possible even if the player gets corrupted before a round in the middle of some command which has more than one round. Patching is simply done up to the round just before the player gets corrupted and then the simulation proceeds as in the static case. Also, if a player gets corrupted just before the input sharing phase but after the environment has provided the inputs, the simulation is essentially the same as before. The only difference is that giving $\mathsf{F}_{\mathtt{SFE}}^{f}$ the input of a corrupted party now means overriding any previous value stored for that party. □

We note that it is possible to modify Protocol CEAS to make it work also for $t < n/2$. The difficulty in this case is that if a player fails during processing of a multiplication gate, we cannot proceed since all players are needed to determine a polynomial of degree $2t$, when we only assume $t < n/2$. The simplest way to handle such failures is to go back to the start of the computation and open the input values of the players that have just been disqualified. Now we restart the protocol, while the honest players simulate the disqualified players. Note that we can use default random choices for these players, so there is no communication needed for the honest players to agree on the actions of the simulated ones. This allows the adversary to slow down the protocol by a factor at most linear in n. In doing this, it is important that we do not have to restart the protocol after some party received its output — this is ensured by starting the Output Reconstruction phase only after all multiplication gates have been handled. If we allowed some party P_i to see its output before all multiplication gates were handled, then it could for instance first run with $x_i = 1$ to let the corrupted parties learn their parts of the output $f(x_1, \ldots, x_n)$ when $x_i = 1$. Then it could make the next multiplication gate fail. Then the parties would rerun, but now using the dummy input $x_i = 0$ for P_i. This would let the corrupted parties learn their parts of the output $f(x_1, \ldots, x_n)$ when $x_i = 0$. Seeing f evaluated on two points might let the corrupted learn more about the inputs of the honest parties than is possible in the ideal world. It is, however, clear that it is secure to rerun as long as no party received its output, as the prefix of the protocol not including the Reconstruction phase leaks no information to any party.

Exercise 5.1 The above way to handle corrupted parties allows the corrupted parties to delay the computation by a factor t, as they might one by one refuse to participate in the last multiplication gate, forcing yet another rerun. It is possible to ensure that they can delay by at most a factor $O(\log(t))$. We sketch the first part of how this is done, and the exercise is to realize how to generalize. Assume that in the first execution of the protocol all parties follow the protocol, except that P_n at some point refuses to participate in the execution of the protocol for a multiplication gate. Then we can let $n' = n-1$, $t' = t-1$ and $f'(x_1, \ldots, x_{n'}) = (y_1, \ldots, y_{n'})$, where the outputs are given by $(y_1, \ldots, y_{n'}, y_n) = f(x_1, \ldots, x_{n'}, 0)$. Then we can let the parties $\mathsf{P}_1, \ldots, \mathsf{P}_{n'}$ run the protocol for securely evaluating f' among themselves using secret sharings of degree t' instead of t. It is secure to only use degree t' as at most t' of the remaining parties are corrupted, as we started with at most t corrupted parties and just excluded P_n. This approach also leads to a correct result, as the honest parties learn their part of the output $f(x_1, \ldots, x_{n-1}, 0)$. So, all P_n got out of cheating was that it was forced to use input $x_n = 0$ and that it does not learn its output. Notice, however, that if we started with $2t+1 \leq n$ we now have that $2t'+1 \leq n'-1$, as $2t'+1 = 2(t-1)+1 = (2t+1) - 2 \leq n-2$ and $n' = n-1$. Since $2t'+1 \leq n'-1$ and we are running the protocol with polynomials of degree at most t' and has n' parties which are holding shares, it follows that we can run the protocol for a multiplication gate even if 1 party do not share its $h(i)$ value — we will still have $n'-1$ of the values $h(i)$ which is enough to interpolate $h(0)$ as $h(\mathtt{X})$ has degree at most $2t'$. This means that to force a rerun of the protocol, at least 2 of the remaining corrupted parties have to stop participating. In that case we exclude these 2 parties and restart with $n'' = n' - 2$ parties and at most $t'' = t' - 2$ corrupted parties. So, we can use polynomials of degree at most $t'' = t - 3$ and hence tolerate that even more corrupted parties stop participating before we need to do the next rerun — we can in fact tolerate that up to 3 of the n'' parties do not help in the multiplications.

1. In the first execution we can tolerate that 0 corrupted parties refuse to participate in the protocol for a multiplication gate. In the second execution we can tolerate that 1 corrupted party refuses to participate in the protocol for a multiplication gate. Argue that in the third execution we can tolerate that 3 corrupted parties refuse to participate in the protocol for a multiplication gate and that we hence can exclude 4 more parties before the fourth execution if we are forced to do a fourth execution.

2. How many refusing parties can we tolerate in the fourth execution?
3. How many refusing parties can we tolerate in the i'th execution?
4. If we start with $2t+1=n$, what is the worst-case number of reruns which the corrupted parties can force?

Let $\pi^f_{\texttt{CEAS-N/2}}$ be $\pi^f_{\texttt{CEAS}}$ modified to handle $t < n/2$ as we just discussed. We then have the following theorem, the proof of which we leave to the reader:

Theorem 5.3 *$\pi^f_{\textit{CEAS-N/2}} \diamond \mathsf{F}_{\textit{SC}} \diamond \mathsf{F}_{\textit{COM}}$ implements $\mathsf{F}^f_{\textit{SFE}}$ in* $\mathrm{Env}^{t,sync}$ *with perfect security for any f and for all $t < n/2$.*

5.3.1 Reactive Functionalities

So far, our results for general secure computing have only been about secure function evaluation. It is easy to define a *reactive functionality* that is much more general than $\mathsf{F}^f_{\mathtt{SFE}}$, namely a functionality that keeps internal state and offers to repeatedly do the following: compute some function that depends on both the state and inputs from the players, and update the state as well as deliver outputs to players.

Using the way we represent data in $\pi_{\texttt{CEAS}}$, it is easy to design a secure implementation of such a functionality: we can represent the state as a number of objects of form $[\![a; f_a]\!]$. We can assume we get the inputs from players represented in the same form, and then we evaluate whatever function we need using the subprotocols for addition and multiplication. When we have representations of the outputs, we keep those that represent the new state, and open those that contain output to the players.

It is straightforward to formalize this construction, we leave the details to the reader.

5.4 Realization of Homomorphic Commitments

We assume throughout this section that we are in the information theoretic scenario with secure point-to-point channels and consensus broadcast. Most of the time we assume that $t < n/3$, though we will look at the case $t < n/2$ at the end of the section. We show how to implement a commitment scheme with the desired basic commands and simple manipulation commands.

The idea that immediately comes to mind in order to have a player D commit to a is to ask him to secret share a. At least this will hide a from the adversary if D is honest, and will immediately ensure the homomorphic properties we need, namely to add commitments, each player just adds his shares, and to multiply by a constant, all shares are multiplied by the constant.

However, if D is corrupt, he can distribute inconsistent shares, and can then easily "open" a commitment in several ways, as detailed in the exercise below.

Exercise 5.2 A player D sends a value a_i to each player P_i (also to himself). D

is supposed to choose these such that $a_i = f(i)$ for all i, for some polynomial $f(\mathtt{X})$ of degree at most t where $t < n/3$ is the maximal number of corrupted players. At some later time, D is supposed to reveal the polynomial $f(\mathtt{X})$ he used, and each P_i reveals a_i, The polynomial is accepted if values of at most t players disagree with $f(\mathtt{X})$ (we cannot demand fewer disagreements, since the corrupted parties might send wrong values, so we may get t of them even if D was honest).

1. We assume here (for simplicity) that $n = 3t+1$. Suppose the adversary corrupts D. Show how to choose two different polynomials $f(\mathtt{X}), f'(\mathtt{X})$ of degree at most t and values $\tilde{a}_i$ for D to send, such that D can later reveal and have accepted both $f(\mathtt{X})$ and $f'(\mathtt{X})$.
2. Suppose for a moment that we would settle for computational security, and that D must send to P_i, not only a_i, but also his digital signature s_i on a_i. We assume that we can force D to send a valid signature even if he is corrupt. We can now demand that to be accepted, a polynomial must be consistent with *all* revealed and properly signed shares. Show that now, the adversary cannot have two different polynomials accepted, even if up to $t \leq n/3$ players may be corrupted before the polynomial is to be revealed. [Hint: First argue that the adversary must corrupt D before the a_i, s_i are sent out (this is rather trivial). Then, assume $f_1(\mathtt{X})$ is later successfully revealed and let C_1 be the set that is corrupted when f_1 is revealed. Assume the adversary could also choose to let D reveal $f_2(\mathtt{X})$, in which case C_2 is the corrupted set. Note that if we assume the adversary is adaptive, you cannot assume that $C_1 = C_2$. But you can still use the players outside C_1, C_2 to argue that $f_1(\mathtt{X}) = f_2(\mathtt{X})$.]
3. (Optional) Does the security proved above still hold if $t > n/3$? Why or why not?

To prevent the problems outlined above, we must find a mechanism to ensure that the shares of all uncorrupted players after committing consistently determine a polynomial $f(\mathtt{X})$ of degree at most t, without harming privacy of course.

Minimal Distance Decoding

Before we do so, it is important to note that n shares out of which at most t are corrupted still uniquely determine the committed value a, even if we don't know which t of them are corrupted. This is based on an observation also used in error correcting codes.

Concretely, let $f(\mathtt{X})$ be a polynomial of degree at most t and define the shares

$$\mathbf{s}_f \stackrel{\text{def}}{=} (f(1), \ldots, f(n)) \ ,$$

and let $\mathbf{e} \in \mathbb{F}^n$ be an arbitrary "error vector" subject to

$$w_H(\mathbf{e}) \leq t \ ,$$

where w_H denotes the Hamming-weight of a vector (i.e., the number of its non-zero coordinates), and define

$$\tilde{\mathbf{s}} \stackrel{\text{def}}{=} \mathbf{s} + \mathbf{e} \ .$$

Then a is uniquely defined by $\tilde{\mathbf{s}}$.

In fact, more is true, since the entire polynomial $f(\mathtt{X})$ is. This is easy to see from Lagrange Interpolation and the fact that $t < n/3$.

Namely, suppose that $\tilde{\mathbf{s}}$ can also be "explained" as originating from some other polynomial $g(\mathtt{X})$ of degree at most t together with some other error vector $\mathbf{u}$ with Hamming-weight at most t. In other words, suppose that

$$\mathbf{s}_f + \mathbf{e} = \mathbf{s}_g + \mathbf{u} \ .$$

Since $w_H(\mathbf{e}), w_H(\mathbf{u}) \leq t$ we have that $w_H(\mathbf{e}+\mathbf{u}) \leq 2t$, so since $n > 3t$ there are at least $n - 2t > t$ positions in which the coordinates of both are simultaneously zero. Thus, for at least $t + 1$ values of i we have

$$f(i) = g(i) \ .$$

Since both polynomials have degree at most t, this means that

$$f(\mathtt{X}) = g(\mathtt{X}) \ .$$

The conclusion is that if a player P_i has distributed shares of some value $a \in \mathbb{F}$, and we can somehow guarantee that the shares are consistent with one polynomial f of degree at most $t < n/3$, then this can serve as a commitment to a. P_i can open by broadcasting the polynomial, and then each player can state whether his share agrees with the polynomial claimed by P_i. The opening is accepted is at most t players disagree. By the above discussion, if P_i tries to open a polynomial different from f, at least $t + 1$ players would disagree.

Note that we do not rely on Lagrange interpolation or any efficient method for decoding in presence of errors: we rely on the committer to supply the correct polynomial, we are fine as long as it is uniquely defined.

Exercise 5.3 This exercises asks you to realize how minimal distance decoding can be used to make a noisy channel reliable. Assume that we have a sender S and a receiver R which are connected be a noisy channel which allows to transfer bit-strings. The effect of the noise is that sometimes a bit sent as 0 will be received as 1 and vice versa. If the errors are probabilistic in nature and not too common, then the first step in handling the errors is to send long bit strings. Then, even though individual bits might flip, most of the bit string will be received correctly. The next step is then to send a redundant string, such that receiving most of the string correctly will allow to compute the original string. This exercise focuses on the second step.

Let t be some fixed parameter, let $n = 3t+1$ and let $\ell = \lceil \log_2(n) \rceil$, and let $\mathbb{F}$ be the finite field with 2^ℓ elements. This allows to consider a bit string $m_i \in \{0,1\}^\ell$ as an element from $\mathbb{F}$, and it allows to consider a bit string $m \in \{0,1\}^{(t+1)\ell}$ as an element from $\mathbb{F}^{t+1}$, as follows: split m into $t + 1$ blocks $m = (m_0, \ldots, m_t)$, each being ℓ bits long. Then consider m_i as an element from $\mathbb{F}$. Now $m \in \mathbb{F}^{t+1}$. Finally, given an element $(m_0, \ldots, m_t) \in \mathbb{F}^{t+1}$ we can define the polynomial $f_m(\mathtt{X}) \overset{\text{def}}{=} \sum_{i=0}^{t} m_i \mathtt{X}^i$ over $\mathbb{F}$, i.e., for each bit string $m \in \{0,1\}^{(t+1)\ell}$ we have a unique polynomial $f_m(\mathtt{X})$ of degree at most t and for each polynomial $f(\mathtt{X})$ of degree at most t we have a bit string $m \in \{0,1\}^{(t+1)\ell}$. To send a bit string from $\{0,1\}^{(t+1)\ell}$ we can hence focus on sending a polynomial of degree at most t.

Show that when S is given a bit string $m \in \{0,1\}^{(t+1)\ell}$ it can send a bit string $c \in \{0,1\}^{n\ell}$ to R which allows R to recover m even if up to t of the bits of c are flipped during transmission.

A Perfect Commitment Protocol

Based on the above observation, what we need is a protocol ensuring that all honest parties end up holding consistent shares of some value, of course while preserving privacy if the committer is (remains) honest.

The idea for ensuring consistent behavior is to have the committer P_i not only secret share the value he commits to, but also distribute some extra "redundant" information that the other players can check.

Redundant sharing using bivariate polynomials.

To do a redundant sharing, P_i chooses a polynomial in two variables:

$$f_a(\mathtt{X},\mathtt{Y}) = \sum_{\sigma,\tau=0}^{t} \alpha_{\sigma,\tau}\mathtt{X}^{\sigma}\mathtt{Y}^{\tau},$$

where the $\alpha_{\sigma,\tau}$ are random, subject to two constraints: First $\alpha_{0,0} = a$, or put differently $f_a(0,0) = a$. Second the polynomial is *symmetric*, i.e., $\alpha_{\sigma,\tau} = \alpha_{\tau,\sigma}$. The committer will then send a polynomial in one variable to each player P_k, namely

$$f_k(\mathtt{X}) = f_a(\mathtt{X},k) = \sum_{\sigma=0}^{t}(\sum_{\tau=0}^{t} \alpha_{\sigma,\tau}k^{\tau})\ \mathtt{X}^{\sigma},$$

where by sending a polynomial, we simply means sending its coefficients.

Interpretation in terms of standard secret sharing.

Let us first explain how this connects to the more standard form of secret sharing we have seen before: consider the polynomial

$$g_a(\mathtt{X}) = f_a(\mathtt{X},0) = \sum_{\sigma=0}^{t} \alpha_{\sigma,0}\mathtt{X}^{\sigma},$$

which is clearly a polynomial of degree at most t such that $g_a(0) = a$. Furthermore, because f_a is symmetric, we have $g_a(k) = f_a(k,0) = f_a(0,k) = f_k(0)$. In other words, by distributing the information we described, the committer has in particular secret shared a in the standard way, using the polynomial $g_a(\mathtt{X})$, where the k'th share is $f_k(0)$.

Furthermore, the coefficients in $f_k(\mathtt{X})$ are $c_\sigma = \sum_{\tau=0}^{t} \alpha_{\sigma,\tau}k^{\tau}$, for $\sigma = 0,\ldots,t$. Note that each c_σ can be interpreted as the value of a degree t polynomial evaluated in point k, namely the polynomial with coefficients $\alpha_{\sigma,0},\ldots,\alpha_{\sigma,t}$. So each c_σ is in fact also a share of a secret, namely the degree-0 coefficient $\alpha_{\sigma,0}$. Note that $\alpha_{\sigma,0}$ is also a coefficient in $g_a(\mathtt{X})$.

This means that the process of choosing $f_a(\mathtt{X},\mathtt{Y})$ and sending $f_k(\mathtt{X})$ to P_k can also be interpreted as follows: first secret share a in the standard way using polynomial $g_a(\mathtt{X})$, and then secret share all the coefficients in $g_a(\mathtt{X})$, also in the standard way, and send shares to the respective players. To make this be exactly equivalent to what we described above, the polynomials for the last step must be chosen such that their coefficients satisfy the symmetry condition. One might think that this could hurt privacy, since then coefficients are not independent. But this is not the case, as we show below.

Consistency Check

The key to checking consistency of the information P_i distributes is to observe that by symmetry of $f_a(\mathtt{X},\mathtt{Y})$, we have for any two players $\mathsf{P}_k, \mathsf{P}_j$ that

$$f_k(j) = f_a(j,k) = f_a(k,j) = f_j(k).$$

The idea for the protocol is then as follows: after each player P_k has received $f_k(\mathtt{X})$, he will check with each other player P_j whether it is the case that $f_k(j) = f_j(k)$. If all pairwise checks go through for a set of at least $t+1$ honest players, their information is sufficient to determine a polynomial of degree at most t that all honest players eventually agree on. Of course the protocol has to take care of what should happen if the checks are not satisfied, here the idea is to ask P_i to help resolve any conflicts.

The complete protocol for implementing the `commit`-command from $\mathsf{F}_{\texttt{COM-SIMPLE}}$ is described in Protocol Commit. The implementation of the other commands is specified in Protocol Perfect-Com-Simple.

Protocol Commit

1. On input $(\mathtt{commit}, i, cid, a)$, P_i samples a bivariate symmetric polynomial $f_a(\mathtt{X},\mathtt{Y})$ of degree at most t, such that $f_a(0,0) = a$. He sends the polynomial $f_k(\mathtt{X}) = f_a(\mathtt{X}, k)$ to each P_k (therefore P_k also learns $\beta_k = f_k(0)$).
2. Each P_j computes $\beta_{k,j} = f_j(k)$ and sends $\beta_{k,j}$ to each P_k.
3. Each P_k checks that $\deg(f_k) \le t$ and that $\beta_{k,j} = f_k(j)$ for $j = 1, \ldots, n$. If so, he broadcasts `success`. Otherwise, he broadcasts $(\mathtt{dispute}, k, j)$ for each inconsistency.
4. For each dispute reported in the previous step, P_i broadcasts the correct value of $\beta_{k,j}$.
5. If any P_k finds a disagreement between what P_i has broadcast and what he received privately from P_i, he knows P_i is corrupt and broadcasts $(\mathtt{accuse}, k)$.
6. For any accusation from P_k in the previous step, P_i broadcasts $f_k(\mathtt{X})$.
7. If any P_k finds a new disagreement between what P_i has now broadcast and what he received privately from P_i, he knows P_i is corrupt and broadcasts $(\mathtt{accuse}, k)$.
8. If the information broadcast by P_i is not consistent, or if more than t players have accused P_i, players output `fail`.
   Otherwise, players who accused P_i and had a new polynomial $f_k(\mathtt{X})$ broadcast will accept it as their polynomial. All others keep the polynomial they received in the first step. Now each P_k outputs `success` and stores (cid, i, β_k). In addition P_i stores the polynomial $g_a(\mathtt{X}) = f_a(\mathtt{X}, 0)$.

Before arguing the security of the protocols, we show two simple results:

Protocol Perfect-Com-Simple

commit: See Protocol Commit.

public commit: On input $(\texttt{pubcommit}, i, cid, a)$ a party P_k lets $a_k = a$ (this is a share of the polynomial $a(\texttt{X}) = a$), outputs $(\texttt{pubcommit}, i, cid, \texttt{success})$ and stores (cid, i, a_k).

open: On input $(\texttt{open}, i, cid)$ where some $(cid, i, a(\texttt{X}))$ is stored, the party P_i broadcasts $(cid, i, a(\texttt{X}))$.
On input $(\texttt{open}, i, cid)$ where some (cid, i, a_k) is stored a party $\mathsf{P}_k \neq \mathsf{P}_i$ broadcasts (cid, i, a_k).
If $\deg(a(\texttt{X})) \leq t$ and $a_k = a(k)$ for at least $n-t$ parties, then all parties output $(\texttt{open}, i, cid, a(0))$. Otherwise they output $(\texttt{open}, i, cid, \texttt{fail})$.

designated open: As above, but the polynomial and the shares are only sent to P_j. If P_j rejects the opening, it broadcasts a public complaint, and P_i must then do a normal opening. If that one fails too, all parties output $\texttt{fail}$.

add: On input $(\texttt{add}, i, cid_1, cid_2, cid_3)$ where some $(i, cid_1, a(\texttt{X}))$ and $(i, cid_2, b(\texttt{X}))$ are stored the party P_i stores $(i, cid_3, a(\texttt{X}) + b(\texttt{X}))$ and outputs $(\texttt{add}, i, cid_1, cid_2, cid_3, \texttt{success})$.
On input $(\texttt{add}, i, cid_1, cid_2, cid_3)$ where some (i, cid_1, a_k) and (i, cid_2, b_k) are stored a party $\mathsf{P}_k \neq \mathsf{P}_i$ stores $(i, cid_3, a_k + b_k)$ and outputs $(\texttt{add}, i, cid_1, cid_2, cid_3, \texttt{success})$.

multiplication by constant: On input $(\texttt{mult}, i, \alpha, cid_2, cid_3)$ where some $(i, cid_2, b(\texttt{X}))$ is stored the party P_i stores $(i, cid_3, \alpha b(\texttt{X}))$ and outputs $(\texttt{mult}, i, \alpha, cid_2, cid_3, \texttt{success})$.
On input $(\texttt{mult}, i, \alpha, cid_2, cid_3)$ where some (i, cid_2, b_k) is stored a party $\mathsf{P}_k \neq \mathsf{P}_i$ stores $(i, cid_3, \alpha b_k)$ and outputs $(\texttt{mult}, i, \alpha, cid_2, cid_3, \texttt{success})$.

Lemma 5.4 *If P_i remains honest throughout Protocol* Commit*, the view of any t corrupted players is independent of the committed value a, and all players who are honest at the end of the protocol will output shares consistent with a and the polynomial $g_a(\texttt{X})$ that P_i distributed.*

Proof We first argue that the information revealed in the first step is independent of a: by Lagrange interpolation, let $h(\texttt{X})$ be a polynomial of degree at most t, such that $h(0) = 1$ and $h(k) = 0$ for all corrupt P_k. Then $h(\texttt{X})h(\texttt{Y})$ is a symmetric polynomial with value 1 in $(0,0)$. Suppose P_i used polynomial $f_a(\texttt{X},\texttt{Y})$. Let $f_0(\texttt{X},\texttt{Y}) = f_a(\texttt{X},\texttt{Y}) - ah(\texttt{X})h(\texttt{Y})$. Clearly, $f_0(0,0) = 0$ and each corrupt P_k will receive the same information whether f_a or f_0 is used. Moreover, adding $-ah(\texttt{X})h(\texttt{Y})$ is a bijection between symmetric polynomials consistent with a and those consistent with 0. Since P_i chooses a uniform polynomial consistent with a, it follows that any t corrupt players in Step 1 see the same distribution regardless of a, namely the one that follows from sharing 0.

In the following steps, one easily sees that the set of corrupt players is told nothing they did not already know: there can be no dispute unless at least one of $\mathsf{P}_j, \mathsf{P}_k$ is corrupt, so they already know all $\beta_{k,j}$ broadcast. Likewise, only a corrupt P_k will accuse an honest P_i, so the corrupt players already know all polynomials and values broadcast. The fact that an honest player never accuses P_i also means that honest players will always output the shares they got in the first step. This implies the last conclusion of the lemma because $g_a(k) = f_a(k, 0) = f_a(0, k) = f_k(0) = \beta_k$ and $g_a(0) = f_a(0, 0) = a$. □

LEMMA 5.5 *If $t < n/3$, then no matter how corrupt players behave in Protocol* COMMIT, *players who are honest at the end of the protocol will all output* **`fail`** *or will output a set of shares in some value a' all consistent with a polynomial $g_{a'}(\mathtt{X})$ of degree at most t.*

PROOF The decision to output fail is based on public information, so it is clear that players who remain honest will either all output `fail` or all output some value, so assume the latter case occurs. This means that at least $n - t$ players did not accuse P_i, and hence by assumption in the lemma at least $n - 2t \geq t + 1$ players who remained honest did not accuse P_i. Let S be the set of indices of such honest players, and consider an honest P_j. We claim that for every $k \in S$ with P_k holding polynomial $f_k(\mathtt{X})$ we have that the $\beta_{k,j}$ that follows from the polynomial $f_j(\mathtt{X})$ held by P_j at the end satisfies

$$f_k(j) = \beta_{k,j}.$$

We argue this as follows: if P_j had a new polynomial broadcast for him in Step 6, then he keeps that one, and the claim is true, since otherwise P_k would have accused in Step 7. On the other hand, suppose no new values were broadcast for P_j. Then he keeps the values he was given in the Step 1, as does P_k, and we can also assume that neither P_j nor P_k accused in Step 5. But our claim holds for the values sent in Step 1, for if not, a $(\mathtt{dispute}, k, j)$ would have been reported, and either P_j or P_k would have accused P_i, and by assumption this did not happen.

Now, let $(r_k)_{k \in S}$ be the reconstruction vector for reconstructing a secret from shares of players in S. That is, for any polynomial f of degree at most t we have $f(0) = \sum_{k \in S} r_k f(k)$, such a vector exists because $|S| \geq t + 1$.

Now define the polynomial $g_{a'}(\mathtt{X})$ as

$$g_{a'}(\mathtt{X}) = \sum_{k \in S} r_k f_k(\mathtt{X})$$

and set $a' = g_{a'}(0)$. Since an honest P_j outputs β_j, we see that in order to show the lemma, we need to show that $g_{a'}(j) = \beta_j$ for all honest P_j. Recall that P_j computes $\beta_{k,j}$ as $\beta_{k,j} = f_j(k)$. Now, by the claim above we can compute as follows:

$$g_{a'}(j) = \sum_{k \in S} r_k f_k(j) = \sum_{k \in S} r_k \beta_{k,j} = \sum_{k \in S} r_k f_j(k) = f_j(0) = \beta_j$$

□

From the above lemmas, we get the following theorem:

THEOREM 5.6 *$\pi_{\textsf{PERFECT-COM-SIMPLE}} \diamond \mathcal{F}_{\textsf{SC}}$ implements $\mathcal{F}_{\textsf{COM-SIMPLE}}$ in* $\mathrm{Env}^{t,sync}$ *with perfect security for all $t < n/3$.*

Agent $\mathcal{S}_{\texttt{PERFECT-COM-SIMPLE}}$

Simulator for $\pi_{\texttt{PERFECT-COM-SIMPLE}}$ for the case of static corruption.

Intialize. In the preamble, the environment $\mathcal{Z}$ specifies the set C of actively corrupted players. Then the simulator sets up an internal copy of $\mathsf{F}_{\texttt{SC}}$, where it corrupts players in C and connects the special ports of $\mathsf{F}_{\texttt{SC}}$ to $\mathcal{Z}$. It also corrupts C on $\mathsf{F}_{\texttt{COM-SIMPLE}}$. Finally, it sets up a copy of every honest player. These "virtual" players are called $\bar{\mathsf{P}}_i, \bar{\mathsf{P}}_k$ and $\bar{\mathsf{P}}_j$; their internal variables will be called $\bar{a}, \bar{\beta}_k$ etc. The basic idea is to let the virtual players execute the protocol with the corrupt players (controlled by $\mathcal{Z}$), letting the virtual players commit to dummy values. Below, we describe in more detail how this is done for each command.

For every committed value a (dummy or not), the simulator stores a polynomial $f_a(\mathtt{X},\mathtt{Y})$ determining the complete set of shares of a held by the (corrupt and virtual) players.

For every honest player P_i we maintain an invariant: Note that every value b that P_i opened via an `open` command is a linear combination of all his committed values so far $a_1, \dots, a_T$: $b = \gamma_1 a_1 + \cdots + \gamma_T a_T$, where the γ_i's are public as they follow from the executed sequence of `add` and `mult` commands. The invariant now is: the set of "committed" dummy values $\{\bar{a}_j\}$ held by $\bar{\mathsf{P}}_i$ is a random set of values consistent with every b that has been opened, i.e., it holds that $b = \gamma_1 \bar{a}_1 + \cdots + \gamma_T \bar{a}_T$, and furthermore each polynomial $f_{\bar{a}_j}$ held by the virtual player is a random symmetric polynomial of degree at most t that is consistent with $\bar{a}_j$ and the shares held by corrupt players.

`commit` If P_i is honest, execute the `commit` protocol giving a random $\bar{a}$ as input to $\bar{\mathsf{P}}_i$. This maintains the invariant as no opened value depends on value we commit now. The simulator stores the polynomial $f_{\bar{a}}$ used by $\bar{\mathsf{P}}_i$. If P_i is corrupt, we execute the `commit` protocol, if it fails, we make $\mathsf{F}_{\texttt{COM-SIMPLE}}$ fail as well, otherwise we use Lemma 5.5 to compute the committed value a' and give this as input to $\mathsf{F}_{\texttt{COM-SIMPLE}}$. The simulator also stores the polynomial $g_{a'}(\mathtt{X})$ guaranteed by the lemma.

`add`, `mult` There is no communication to simulate, we just let the virtual players do the local computation prescribed in the protocol.

`open` If the committer is honest, we get the value z to be revealed from leakage of $\mathsf{F}_{\texttt{COM-SIMPLE}}$. We now adjust the views of the virtual players so that it is consistent with the value z and hence the invariant is maintained. We do this by considering all equations of form $b = \gamma_1 a_1 + \cdots + \gamma_T a_T$ for revealed values b (including the new equation for z), and set $(\bar{a}_1, \dots, \bar{a}_t)$ to be a random solution to this system of equations. Note that a solution must exist, namely $(\bar{a}_1, \dots, \bar{a}_t) = (a_1, \dots, a_t)$, the set of values actually input by $\mathcal{Z}$. The set of committed values held by the virtual player is set to be the new solution we found, and we adjust each polynomial $f_{\bar{a}_j}(\mathtt{X},\mathtt{Y})$ so that the invariant is maintained. This is done by computing the new polynomial as $f_{\bar{a}_{j,new}}(\mathtt{X},\mathtt{Y}) = f_{\bar{a}_{j,old}}(\mathtt{X},\mathtt{Y}) + (\bar{a}_{j,new} - \bar{a}_{j,old})h(\mathtt{X})h(\mathtt{Y})$, where h is a degree at most t polynomial that is 1 in 0 and 0 in points of corrupt players. This always works as there are at most t corrupt players. Now all virtual players are still in a consistent state, and we can execute the `open` protocol.

If the committer is corrupt, we simply execute the `open` protocol, if it fails, we make $\mathsf{F}_{\texttt{COM-SIMPLE}}$ fail as well.

PROOF A simulator for the case of static corruption is given as $\mathcal{S}_{\texttt{PERFECT-COM-SIMPLE}}$. We argue that this is a perfect simulator: all `commit` commands are simulated perfectly, this follows from Lemma 5.4 if the committer is honest and from that fact the we just follow the protocol if the committer is corrupt. Furthermore, each `open` command will reveal the shares the environment already knew in case the committer is corrupt (note that the existence of these shares for all honest players is guaranteed by Lemma 5.5). If the committer is honest, it will be a set of shares consistent with what the corrupt players have and the opened value. This holds in both real protocol and simulation.

It follows that the view of corrupt players is identically distributed in real protocol and in execution, so the only way the environment can distinguish is if the input/output behavior of honest players is different in simulation and in the real execution. This cannot happen for an honest committer, since the simulator gets the correct values to open and the open protocol always succeeds for an honest committer. It cannot happen for a corrupt committer either: if the open protocol fails, the simulator makes the functionality fail as well, and if it succeeds, the opened value must be the one that follows from earlier committed values and therefore equals the one output by the functionality. This is because Lemma 5.5 guarantees that all honest players have consistent shares of each committed value, therefore by our discussion on minimal distance decoding, an incorrect polynomial will disagree with at least $t+1$ honest players and the opening will fail.

Adaptive Corruption.

To simulate for adaptive corruption, we start $\mathcal{S}_{\texttt{PERFECT-COM-SIMPLE}}$ in the state where all players are honest. Recall that $\mathcal{S}_{\texttt{PERFECT-COM-SIMPLE}}$ works with virtual players $\bar{\mathsf{P}}_i$. When we get to a point where a new player P is corrupted, we corrupt P on $\mathsf{F}_{\texttt{COM-SIMPLE}}$ and get the set of values he committed to from its leakage port. Using this data we then adjust (if needed) the views of the virtual players so they are consistent with the new values we learned and everything the environment has seen so far. We give the view of $\bar{\mathsf{P}}$ to the environment, and continue the simulation following the algorithm of $\mathcal{S}_{\texttt{PERFECT-COM-SIMPLE}}$, giving it the adjusted state of the virtual players (except for that of P who is now corrupt).

We will see that the view of P that we give to the environment always has exactly the correct distribution, and furthermore the state of virtual players is distributed exactly as if we had run $\mathcal{S}_{\texttt{PERFECT-COM-SIMPLE}}$ with P being corrupt from the start, but where the environment let it behave honestly up to this point. Therefore it follows from the result for static corruption, that the simulation after the corruption is perfect.

We now describe the procedure for adjusting views: we will be given every value committed by P. For every such value a, the state of $\mathcal{S}_{\texttt{PERFECT-COM-SIMPLE}}$ contains a polynomial $f_{\bar{a}}(\mathtt{X},\mathtt{Y})$ that is part of its state for the virtual player $\bar{\mathsf{P}}$. We now replace $f_{\bar{a}}(\mathtt{X},\mathtt{Y})$ by $f_{\bar{a}}(\mathtt{X},\mathtt{Y}) + (a-\bar{a})h(\mathtt{X})h(\mathtt{Y})$, where $h()$ is a polynomial that is 1 in 0 and 0 in all points of players that were corrupted earlier, and we adjust the view of all virtual players accordingly. By the invariant of $\mathcal{S}_{\texttt{PERFECT-COM-SIMPLE}}$, $f_{\bar{a}}(\mathtt{X},\mathtt{Y})$ is a random polynomial consistent with $\bar{a}$ and the views of corrupt players. The adjustment we do will maintain this, but now with $\bar{a}$ replaced by a. $\square$

Agent $\mathrm{F}_{\mathtt{ICSIG}}$

Ideal functionality for IC signatures. Each instance of this functionality provides service to two special players P_D and P_{INT}. The functionality does a complete breakdown if both are corrupt or if inputs from honest players are not given as specified. Whenever a command is received, the functionality leaks the name of the command and whatever public input is given. Output from a command is sent once the round in which to deliver is specified on the influence port.

sign: This command is executed if in some round player P_D sends $(\mathtt{sign}, sid, a)$ and in addition all honest players send $(\mathtt{sign}, sid, ?)$. In this case $\mathsf{F}_{\mathtt{ICSIG}}$ records the pair (sid, a). Here, sid is just an identifier, and a is the value that is "signed".[a] The output to all parties is $(\mathtt{sign}, sid, \mathtt{success})$, in addition a is sent to P_{INT}.

reveal: This command is executed if in some round all honest players send $(\mathtt{reveal}, sid)$ and some (sid, a) is stored. The output to all parties is $(\mathtt{reveal}, sid, a)$.
If P_{INT} is corrupted and does not input $(\mathtt{reveal}, sid)$, then $(\mathtt{reveal}, sid, \mathtt{fail})$ is output to all parties.

designated reveal: This command is executed if in some round all honest players send $(\mathtt{reveal}, j, sid)$ and some (sid, a) is stored. The command behaves exactly as $\mathtt{reveal}$, except that only P_j gets output.

add: This command is executed if in some round all honest players send $(\mathtt{add}, sid_1, sid_2, sid_3)$ and some (sid_1, a_1) is stored and some (sid_2, a_2) is stored. As a result $\mathsf{F}_{\mathtt{ICSIG}}$ stores $(sid_3, a_1 + a_2)$. The output to all parties is $(\mathtt{add}, sid_1, sid_2, sid_3, \mathtt{success})$.

mult by constant: This command is executed if in some round all honest players send $(\mathtt{mult}, c, sid_2, sid_3)$ and $c \in \mathbb{F}$ and some (i, sid_2, a_2) is stored. As a result $\mathsf{F}_{\mathtt{ICSIG}}$ stores (i, sid_3, ca_2). The output to all parties is $(\mathtt{mult}, c, sid_2, sid_3, \mathtt{success})$.

[a] We require that all honest players agree to the fact that a value should be signed because an implementation will require the active participation of all honest players.

5.4.1 Commitments in Case of Honest Majority

In this section, we present a protocol that implements $\mathsf{F}_{\mathtt{COM\text{-}SIMPLE}}$ with statistical security for the case of honest majority, i.e., when $t < n/2$. We will use the same idea as before of committing by secret sharing the value to commit to. But the problem we face is that even if we force the committer to give us consistent shares, the resulting commitment will not be binding.

The problem is that there may only be $t + 1$ honest players. Even if they all have consistent shares, the adversary can easily cook up a different polynomial that is consistent with t of these players, and any value of the secret he desires. If he would open such a value using the `open` protocol we have seen, only one honest player would disagree, and this may as well be a corrupt player who is complaining for no good reason.

The problem could be solved if the unhappy honest player could *prove* that his

Protocol ICSign

initialize: This step is carried out before any commands are executed. For $i = 1, \ldots, n$, P_D chooses $\alpha_i \in \mathbb{F}$ a random and sends α_i to P_i, both players store the value.

sign: On input $(\texttt{sign}, sid, a)$ to P_D and $(\texttt{sign}, sid, ?)$ to other players, do the following:

For $i = 1, \ldots, n$, P_D chooses $\beta_{i,a} \in \mathbb{F}$ at random and sets $K_{i,a} = (\alpha_i, \beta_{i,a})$. He sends $\beta_{i,a}$ to P_i and $a, \{m_{i,a} = \mathrm{MAC}_{K_{i,a}}(a)\}_{i=1,\ldots,n}$ to P_{INT}.

We now check that P_D has sent correctly formed data. First, P_D chooses $a' \in \mathbb{F}$ and $\beta_{i,a'} \in \mathbb{F}$ at random, sets $K_{i,a'} = (\alpha_i, \beta_{i,a'})$ and sends $\beta_{i,a'}$ to P_i. He also sends $a', \{m_{i,a'} = \mathrm{MAC}_{K_{i,a'}}(a')\}_{i=1,\ldots,n}$ to P_{INT}.

P_{INT} chooses $e \in \mathbb{F}$ at random and broadcasts $e, a' + ea$.

Then do the following loop for $i = 1, \ldots, n$:

1. P_{INT} broadcasts $m_{i,a'} + em_{i,a} = \mathrm{MAC}_{K_{i,a'} + eK_{i,a}}(a' + ea)$.
2. P_D checks the broadcast message from P_{INT}. If it is correct, he broadcasts `accept`, else he broadcasts "P_{INT} is corrupt" and we exit the loop.
3. P_i checks that $m_{i,a'} + em_{i,a} = \mathrm{MAC}_{K_{i,a'} + eK_{i,a}}(a' + ea)$, and broadcasts `accept` or `reject` accordingly. P_D checks that P_i acted correctly, if so he broadcasts `accept` else he broadcasts "P_i is corrupt: $K_{i,a}$". This broadcasted key value will be used in the following and P_{INT} adjusts his values so it holds that $m_{i,a} = \mathrm{MAC}_{K_{i,a}}(a)$. If P_i said `reject` and P_D said `accept`, we exit the loop (this can only happen if P_D is corrupt).

If the above loop was terminated by an exit, P_D broadcasts $a, \{m_{i,a}\}_{i=1,\ldots,n}$. These values will be used in the following and each P_i adjusts his $\beta_{i,a}$-value so it holds that $m_{i,a} = \mathrm{MAC}_{K_{i,a}}(a)$.

P_{INT} outputs $(\texttt{sign}, sid, a, \texttt{success})$ where he uses the last value for a he received from P_D, and stores $a, \{m_{i,a}\}_{i=1,\ldots,n}$ under sid. Each P_i outputs $(\texttt{sign}, sid, \texttt{success})$ and stores $K_{i,a}$ under sid.

inconsistent share was in fact what he received from the committer. This would require something similar to a signature scheme.

IC Signatures

We therefore begin with an important tool, known as Information Checking or IC-signatures. The idea is to provide a functionality similar to digital signatures, but with information theoretic security. The basic version of this idea involves two players that we call $\mathsf{P}_D, \mathsf{P}_{INT}$ for Dealer and Intermediary. The basic idea is that P_D can give a secret message s to P_{INT}, and at any later time P_{INT} can choose to reveal s and prove to the other players that s really was the value he got from P_D. We specify what we want more formally as a functionality $\mathsf{F}_{\texttt{ICSIG}}$. The main properties we get are that if both $\mathsf{P}_D, \mathsf{P}_{INT}$ are honest, then s will always be successfully revealed, but no corrupt player knows s before it is revealed. If P_{INT} is corrupt (but P_D is honest), he can refuse to reveal s but cannot reveal a wrong value successfully. Finally, if P_D is corrupt (but P_{INT} is honest), it will still be possible for P_{INT} to convince the other players about the correct value of s.

In addition we want that linear operations can be done on signed values, so

Protocol ICSig (Continued)

(designated) reveal: On input $(\texttt{reveal}, sid)$ to all players, P_{INT} broadcasts $a, \{m_{i,a}\}_{i=1,\dots,n}$ as stored under sid. Each P_i retrieves $K_{i.a}$ stored under sid and checks that $m_{i,a} = \mathrm{MAC}_{K_{i,a}}(a)$. If this holds, he outputs $(\texttt{reveal}, sid, a)$ else he output $(\texttt{reveal}, sid, \texttt{fail})$.
If the command was designated, $(\texttt{reveal}, j, sid)$, P_{INT} only sends $a, m_{j,a}$ to P_j who checks the MAC as in the normal `reveal` command.

add: On input $(\texttt{add}, sid_1, sid_2, sid_3)$ to all players, P_{INT} retrieves the values $(a_1, m_{1,a_1}, \dots, m_{n,a_1}), (a_2, m_{1,a_2}, \dots, m_{n,a_2})$, that were stored under sid_1, sid_2 and stores $(a_1 + a_2, m_{1,a_1} + m_{1,a_2}, \dots, m_{n,a_1} + m_{n,a_2})$ under sid_3.
Each P_i retrieves the values $\beta_{i,a_1}, \beta_{i,a_2}$ stored under sid_1, sid_2 and stores $\beta_{i,a_1} + \beta_{i,a_2}$ under sid_3.
All players output $(\texttt{add}, sid_1, sid_3, sid_3, \texttt{success})$.

multiplication by constant: On input $(\texttt{mult}, c, sid_2, sid_3)$ to all players, P_{INT} retrieves the values $(a_2, m_{1,a_2}, \dots, m_{n,a_2})$, that were stored under sid_2 and stores $(c \cdot a_2, c \cdot m_{1,a_2}, \dots, c \cdot m_{n,a_2})$ under sid_3.
Each P_i retrieves the value β_{i,a_2} stored under sid_2 and stores $c \cdot \beta_{i,a_2}$ under sid_3.
All players output $(\texttt{mult}, c, sid_2, sid_3, \texttt{success})$.

that if P_D has given $a, b \in \mathbb{F}$ to P_{INT}, then P_{INT} can later convince everyone about the correct value of $ca + b$ for a publicly known constant c.

The protocol for implementing $\mathsf{F}_{\texttt{ICSIG}}$ is based on the well-known notion of unconditionally secure message authentication codes (MACs). Such a MAC uses a key K, a random pair $K = (\alpha, \beta) \in \mathbb{F}^2$, and the authentication code for a value $a \in \mathbb{F}$ is $\mathrm{MAC}_K(a) = \alpha a + \beta$.

We will apply the MACs by having P_D give a key $K_{i,a} = (\alpha_i, \beta_{i,a})$ to each P_i and a set of values $a, \mathrm{MAC}_{K_{i,a}}(a), i = 1, \dots, n$ to P_{INT}. The idea is to use the MACs to prevent P_{INT} from lying about a when he broadcasts it. Given a and $K_{i,a}$, each P_i can compute the MAC value he expects to see and compare to the MAC sent by P_{INT}.

It will be very important in the following that if we keep α constant over several different MAC keys, then one can add two MACs and get a valid authentication code for the sum of the two corresponding messages. More concretely, two keys $K = (\alpha, \beta), K' = (\alpha', \beta')$ are said to be *consistent* if $\alpha = \alpha'$. For consistent keys, we define $K + K' = (\alpha, \beta + \beta')$, it is then trivial to verify that $\mathrm{MAC}_K(a) + \mathrm{MAC}_{K'}(a') = \mathrm{MAC}_{K+K'}(a + a')$.

If P_{INT} is corrupt (but P_D is honest) it is also easy to see (as detailed in the proof below) that P_{INT} will be able to lie about the values he was given (or about any linear combination of them) with probability at most $1/|\mathbb{F}|$. Also, a key clearly reveals no information on the authenticated value. On the other hand, if P_D is corrupt (but P_{INT} is honest), he may not give consistent information to players, so the protocol must do an interactive verification when a value is signed, in order to check this.

Agent $\mathcal{S}_{\mathtt{ICSIG}}$

Simulator for $\pi_{\mathtt{ICSIG}}$ for the case of static corruption.

intialize In the preamble, the environment $\mathcal{Z}$ specifies the set C of actively corrupted players. Then the simulator sets up an internal copy of $\mathsf{F}_{\mathtt{SC}}$, where it corrupts players in C and connects the special ports of $\mathsf{F}_{\mathtt{SC}}$ to $\mathcal{Z}$. It also corrupts C on $\mathsf{F}_{\mathtt{ICSIG}}$. Finally, it sets up a copy of every honest player. These "virtual" players are called $\bar{\mathsf{P}}_i, \bar{\mathsf{P}}_D$ and $\bar{\mathsf{P}}_{INT}$; their internal variables will be called $\bar{a}, \bar{a'}$ etc. The basic idea is to let the virtual players execute the protocol with the corrupt players (controlled by $\mathcal{Z}$). Below, we describe in more detail how this is done for each command.
We consider three "modes": 1) both $\mathsf{P}_D, \mathsf{P}_{INT}$ are honest, 2) P_D is corrupt, and 3) P_{INT} is corrupt. If both $\mathsf{P}_D, \mathsf{P}_{INT}$ are corrupt, simulation becomes trivial as $\mathsf{F}_{\mathtt{ICSIG}}$ does a complete breakdown in that case.
In mode 1), we maintain an invariant: Note that every value b that is revealed in a `reveal` command is a linear combination of all signed values so far $a_1, \dots, a_t$: $b = \gamma_1 a_1 + \cdots + \gamma_t a_t$, where the γ_i's are public as they follow from the executed sequence of `add` and `mult` commands. The invariant now is: the set of "signed" values $\{\bar{a}_j\}$ held by $\bar{\mathsf{P}}_D$ is a random set of values consistent with every b that has been revealed, i.e., it holds that $b = \gamma_1 \bar{a}_1 + \ldots + \gamma_t \bar{a}_t$.

`sign` In mode 1), execute the `sign` protocol giving a random $\bar{a}$ as input to $\bar{\mathsf{P}}_D$. This maintains the invariant as no revealed value depends on the input we sign now. In mode 2), we get a value a that the corrupt P_D sends via $\mathsf{F}_{\mathtt{SC}}$. We give a as input to $\mathsf{F}_{\mathtt{ICSIG}}$ on behalf of P_D and then execute the `sign` protocol. In mode 3), we get a value a from leakage of $\mathsf{F}_{\mathtt{ICSIG}}$ (since P_{INT} is corrupt) and we execute the `sign` protocol giving a as input to $\bar{\mathsf{P}}_D$.

`add`, `mult` There is no communication to simulate, we just let the virtual players do the local computation prescribed in the protocol.

(designated) `reveal` For a public `reveal` in mode 1), we get the value z to be revealed from leakage of $\mathsf{F}_{\mathtt{ICSIG}}$. We now adjust the views of the virtual players so that it is consistent with the value z and hence the invariant is maintained. We do this by considering all equations of form $b = \gamma_1 a_1 + \cdots + \gamma_t a_t$ for revealed values b (including z), and set $(\bar{a}_1, \dots, \bar{a}_t)$ to be a random solution to this system of equations. Note that a solution must exist, namely $(\bar{a}_1, \dots, \bar{a}_t) = (a_1, \dots, a_t)$ the set of values actually input by $\mathcal{Z}$. If this requires a previous value of $\bar{a}_j$ to be changed, we compensate for this by adjusting the value of $\bar{a}'_j$ (and hence the MAC on $\bar{a}'_j$) such that all broadcasted values remain the same. Now all virtual players are still in a consistent state, and we can execute the `reveal` command. The same is done for a designated reveal where the receiver is corrupt. If the receiver is honest, the corrupt players see no communication, the simulator just executes the designated `reveal` internally to keep a consistent state.
In modes 2) and 3), we have executed the protocol with the correct signed values and we simply therefore simply execute the `reveal` or designated `reveal` protocol.

THEOREM 5.7 *$\pi_{\mathtt{ICSIG}} \diamond \mathsf{F}_{\mathtt{SC}}$ implements $\mathsf{F}_{\mathtt{ICSIG}}$ in* $\mathrm{Env}^{t,sync}$ *with statistical security for all $t < n$.*

PROOF As usual, we construct a simulator $\mathcal{S}_{\mathtt{ICSIG}}$ for the protocol. The simulator shown deals with static corruption, and considers separately the three cases where: both $\mathsf{P}_D, \mathsf{P}_{INT}$ are honest, P_D is corrupt and P_{INT} is corrupt. At the same time any number of other players may be corrupt. If P_D and P_{INT} are both corrupt, there is nothing to show as the functionality does a complete breakdown in that case. We show below how to deal with adaptive corruption, but first argue why the simulation is statistically close to a real execution:

Both $\mathsf{P}_D, \mathsf{P}_{INT}$ are honest.

In this case, $\mathcal{S}_{\texttt{ICSIG}}$ runs the protocol on behalf of the honest players using random dummy inputs. Note that this also involves generating keys to be held by each player P_i (whether he is corrupt or not). Now, by the random choice of a', the view of the `sign` protocol is independent of the signed values, so the simulation of this part is perfect. So is the simulation of the `add` and `mult` commands since no communication is done. Furthermore, when a value b is opened, the simulator adjusts the views of its virtual players so they are consistent with b. This includes computing from $K_{i,b}$ a MAC on b to show to each P_i. Since the key held by P_i has the correct distribution and the MAC follows deterministically from b and $K_{i,b}$, the joint distribution of everything seen by $\mathcal{Z}$ in the `reveal` command is correct.

P_{INT} is corrupt.

In this case, $\mathcal{S}_{\texttt{ICSIG}}$ knows each a to be signed from the start and now simulates by simply following the protocol. Therefore the only way in which the simulation can deviate from the real execution comes from the fact that in the simulation, when a value is revealed, honest players get the value from the functionality, which by definition always reveals the value sent by P_D, while this may not be the case in the real protocol. It *is* the case except with negligible probability, however: if P_{INT} is about to reveal a value a to honest player P_i, he knows the correct MAC $m_{i,a} = \alpha_i a + \beta_{i,a}$. This reveals no information on α_i by random choice of $\beta_{i,a}$, and neither does any other MACs known to P_{INT} by the same argument. Now, if P_{INT} produces $a' \neq a$ and $m_{i,a'}$ that P_i would accept, we know that $m_{i,a} = \alpha_i a + \beta_{i,a}$ and $m_{i,a'} = \alpha_i a' + \beta_{i,a}$. Subtracting one equation from the other we obtain $\alpha_i = (m_{i,a} - m_{i,a'})/(a - a')$, which makes sense because $a - a' \neq 0$. In other words P_{INT} must guess α_i to cheat and this can done with probability at most $1/|\mathbb{F}|$. If a MAC is rejected and the protocol continues, P_{INT} knows that a certain value is not the correct α_i. But in a polynomial time protocol, he will only be able to exclude a polynomial number of values, so the probability to cheat remains negligible. The simulation is therefore statistically close in this case.

P_D is corrupt.

In this case, $\mathcal{S}_{\texttt{ICSIG}}$ knows each a to be signed from the start and simulates by following the protocol. This means that the only way the simulation can deviate from the real protocol comes from the fact that in the simulation, the functionality on a `reveal` command always reveals the value it got from P_D to the honest players. In the protocol it may happen that the honest P_{INT} attempts to reveal the correct value but has it rejected by an honest P_i, however, this only happens with negligible probability: in the cases where in the `sign` phase, P_D broadcasts a and a set of MACs, or broadcasts the key for some P_i, the problem never occurs because all honest P_i are guaranteed to have data that are consistent with P_{INT}. Therefore, the only remaining case is where P_i says `accept` in the sign phase and P_D does not claim that P_i or P_{INT} are corrupt. Now, P_i only accepts if it holds that

$$m_{i,a'} + em_{i,a} = \mathrm{MAC}_{K_{i,a'}+eK_{i,a}}(a' + ea) = \alpha_i(a' + ea) + \beta_{i,a'} + e\beta_{i,a}.$$

If P_D gave an incorrect MAC to P_{INT} in the first place then $m_{i,a} \neq \alpha_i a + \beta_{i,a}$. This would mean that we could rearrange the above equation as follows

$$e = (\alpha_i a' + \beta_{i,a'} - m_{i,a'})/(m_{i,a} - \alpha_i a - \beta_{i,a})$$

and compute e from the values generated by P_D before e is chosen, in other words P_D must guess e to successfully distribute inconsistent data, so this happen with only negligible probability $1/|\mathbb{F}|$, and the simulation is statistically close also in this case.

Adaptive Corruption.

To construct a simulator for adaptive corruption, we start $\mathcal{S}_{\mathtt{ICSIG}}$ in the mode where all players are honest. Recall that $\mathcal{S}_{\mathtt{ICSIG}}$ works with virtual players $\bar{\mathsf{P}}_i, \bar{\mathsf{P}}_D$ and $\bar{\mathsf{P}}_{INT}$. When a new player P is corrupted, we corrupt P on $\mathsf{F}_{\mathtt{ICSIG}}$ and get some data from the leakage port of $\mathsf{F}_{\mathtt{ICSIG}}$. Using this data we then adjust (if needed) the views of the virtual players so they are consistent with everything the environment has seen so far, we give the view of $\bar{\mathsf{P}}$ to the environment, and continue the simulation following the algorithm of $\mathcal{S}_{\mathtt{ICSIG}}$. If $\mathsf{P} = \mathsf{P}_D$ or P_{INT}, we switch to the simulation mode where P_D or P_{INT} is corrupt. We will see that the view of P that we give to the environment always has exactly the correct distribution. Therefore we can argue, as for $\mathcal{S}_{\mathtt{ICSIG}}$, that the only case where simulation and real execution differ is if a corrupt P_D or P_{INT} succeeds in cheating which happens with negligible probability as we saw above.

We now describe the procedure for adjusting views: if P is not P_D or P_{INT}, we use the view of $\bar{\mathsf{P}}$ with no change. This works because the simulator gets no new input/output information when such a player is corrupted, and the view of such a player consists of keys (that we chose with the right distribution) and possibly some revealed values and MACs that were already broadcast.

If P is not P_D or P_{INT}, we are given all values that were privately revealed to this player in designated `reveal`. If $\mathsf{P} = \mathsf{P}_D$ or P_{INT}, we are given every signed value. In any case, we then run the procedure used by $\mathcal{S}_{\mathtt{ICSIG}}$ during the simulation of `reveal`, to adjust the views of $\bar{\mathsf{P}}_D$ and $\bar{\mathsf{P}}_{INT}$ to be consistent with each value we got from $\mathsf{F}_{\mathtt{ICSIG}}$ (by doing the same as if each value was revealed). This gives a correctly distributed history for every value we consider: given the correct set of a-values to sign (and random e-values), one can either choose the a'-values at random and compute the broadcasted values from this (this is what the real protocol does) – or one can choose the broadcasted values with the correct distribution and compute matching a'-values. This is what the simulation does. Both methods clearly lead to the same distribution. Note that all keys are chosen independently and the MAC's follow deterministically from keys and authenticated values. Keys and MACs therefore make no difference to this argument.

Finally, if both P_D and P_{INT} are corrupted, we do as just explained, give the views of $\bar{\mathsf{P}}_D$ and $\bar{\mathsf{P}}_{INT}$ to $\mathcal{Z}$ and let $\mathcal{Z}$ decide what happens hereafter (since $\mathsf{F}_{\texttt{ICSIG}}$ now breaks down). □

Protocol Statistical-Commit

1. On input $(\texttt{commit}, i, cid, a)$, P_i samples a bivariate symmetric polynomial $f_a(\mathtt{X}, \mathtt{Y})$ of degree at most t, such that $f_a(0,0) = a$.
   For each coefficient of the polynomial $f_k(\mathtt{X}) = f_a(\mathtt{X}, k)$, players call `sign` on $\mathsf{F}_{\texttt{ICSIG}}$ with $\mathsf{P}_D = \mathsf{P}_i, \mathsf{P}_{INT} = \mathsf{P}_k$ and the coefficient being the value to sign. This means that P_k learns all the coefficients, including $\beta_k = f_k(0)$.
2. Each P_j computes $\beta_{k,j} = f_j(k)$ and sends $\beta_{k,j}$ to each P_k.
3. Each P_k checks that $\beta_{k,j} = f_k(j)$ for $j = 1, \dots, n$. If so, he broadcasts `success`. Otherwise, he broadcasts $(\texttt{dispute}, k, j)$ for each inconsistency.
4. For each dispute reported in the previous step, P_i broadcasts the correct value of $\beta_{k,j}$.
5. If any P_k finds a disagreement between what P_i has broadcast and what he received privately from P_i, he knows P_i is corrupt and broadcasts $(\texttt{accuse}, k)$, as well as the offending value β'. This value is always a linear combination of values received directly from P_i.
6. For any accusation from P_k in the previous step, players call an appropriate sequence of `add` and `mult` commands to have $\mathsf{F}_{\texttt{ICSIG}}$ compute β' internally. Then they call `reveal` to make that value public. If the revealed value is indeed the β' broadcast by P_k and β' differs from the corresponding value broadcast by P_i, players output `fail` and the protocol ends here.
7. If we reach this step, no accusations were proved. Each P_k outputs `success` and stores (cid, i, β_k). In addition P_i stores the polynomial $g_a(\mathtt{X}) = f_a(\mathtt{X}, 0)$.

A Statistically Secure Commitment Scheme

The basic idea in the implementation of $\mathsf{F}_{\texttt{COM-SIMPLE}}$ for $t < n/2$ is to do essentially the same protocol as for $t < n/3$, but ask the committer to sign every value he sends to players. This actually makes the `commit` protocol simpler because accusations against the committer can be proved: the accuser can demonstrate that the message he claims to have received from the committer is correct, and everyone can then check that it indeed contradicts what the committer has broadcast.

Protocol Statistical-Commit and Protocol Com-Simple specify the solution in detail. The protocols assume access to $\mathsf{F}_{\texttt{ICSIG}}$.

The following theorem is easy to show, using the same principles we have seen many times by now. We leave the details to the reader.

Theorem 5.8 *$\pi_{\textit{COM-SIMPLE}} \diamond \pi_{\textit{ICSIG}} \diamond F_{SC}$ implements $F_{\textit{COM-SIMPLE}}$ in* $\mathrm{Env}^{t,sync}$ *with statistical security for all $t < n/2$.*

Protocol Com-Simple

commit: See Protocol Statistical-Commit.

public commit: On input $(\mathtt{pubcommit}, i, cid, a)$ a party P_k lets $a_k = a$ (this is a share on the polynomial $a(\mathtt{X}) = a$), outputs $(\mathtt{pubcommit}, i, cid, \mathtt{success})$ and stores (cid, i, a_k). Players call `sign` on $\mathsf{F}_{\mathtt{ICSIG}}$ with $\mathsf{P}_D = \mathsf{P}_i, \mathsf{P}_{INT} = \mathsf{P}_k$ and a as the value to sign (this is just for consistency so that every value stored after a commit is signed by the committer).[a]

open: On input $(\mathtt{open}, i, cid)$ where some $(cid, i, a(\mathtt{X}))$ is stored, the party P_i broadcasts $(cid, i, a(\mathtt{X}))$.
On input $(\mathtt{open}, i, cid)$ where some (cid, i, a_k) is stored each party $\mathsf{P}_k \neq \mathsf{P}_i$ broadcasts (cid, i, a_k). Players call `reveal` on $\mathsf{F}_{\mathtt{ICSIGN}}$ to confirm that a_k is correct. If the `reveal` fails or the revealed value is not a_k, P_k is *ignored.*
If $\deg(a(\mathtt{X})) \leq t$ and $a_k = a(k)$ for all parties that were not ignored then all parties output $(\mathtt{open}, i, cid, a(0))$. Otherwise they output $(\mathtt{open}, i, cid, \mathtt{fail})$.

designated open: As above, but the polynomial and the shares are only sent to P_j, and designated `reveal` is called on $\mathsf{F}_{\mathtt{ICSIG}}$. If P_j rejects the opening, it broadcasts a public complaint, and P_i must then do a normal opening. If that one fails too, all parties output `fail`.

add: On input $(\mathtt{add}, i, cid_1, cid_2, cid_3)$ where some $(i, cid_1, a(\mathtt{X}))$ and $(i, cid_2, b(\mathtt{X}))$ are stored the party P_i stores $(i, cid_3, a(\mathtt{X}) + b(\mathtt{X}))$ and outputs $(\mathtt{add}, i, cid_1, cid_2, cid_3, \mathtt{success})$.
On input $(\mathtt{add}, i, cid_1, cid_2, cid_3)$ where some (i, cid_1, a_k) and (i, cid_2, b_k) are stored a party $\mathsf{P}_k \neq \mathsf{P}_i$ stores $(i, cid_3, a_k + b_k)$. Players call `add` on $\mathsf{F}_{\mathtt{ICSIG}}$ to have it add a_k and b_k internally. P_k outputs $(\mathtt{add}, i, cid_1, cid_2, cid_3, \mathtt{success})$.

multiplication by constant: On input $(\mathtt{mult}, i, \alpha, cid_2, cid_3)$ where some $(i, cid_2, b(\mathtt{X}))$ is stored the party P_i stores $(i, cid_3, \alpha b(\mathtt{X}))$ and outputs $(\mathtt{mult}, i, \alpha, cid_2, cid_3, \mathtt{success})$.
On input $(\mathtt{mult}, i, \alpha, cid_2, cid_3)$ where some (i, cid_2, b_k) is stored a party $\mathsf{P}_k \neq \mathsf{P}_i$ stores $(i, cid_3, \alpha b_k)$. Players call `mult` on $\mathsf{F}_{\mathtt{ICSIG}}$ to have it multiply α and b_k internally. P_k outputs $(\mathtt{mult}, i, \alpha, cid_2, cid_3, \mathtt{success})$.

[a] Formally we would extend $\mathsf{F}_{\mathtt{ICSIG}}$ with a “public sign” command *a la* the **public commit** command of $\mathsf{F}_{\mathtt{COM}}$ and implement the signature simply by letting all parties store a. We leave the formal details as an exercise.

5.5 Final Results and Optimality of Corruption Bounds

In this section, we put together all the results we have seen in this chapter and show that the resulting bounds on the number of actively corrupted parties we can tolerate with our protocols are in fact optimal. We emphasize that all the negative results hold even if we only require security for static corruption, while the positive results hold for adaptive security.

Let $\mathsf{F}_{\mathtt{PPC}}$ be a functionality that provides a secure point-to-point channel between every pair of players. That is, it is defined exactly as $\mathsf{F}_{\mathtt{SC}}$, except that broadcast is not provided. A first important fact is that Lamport *et al.* [124] have shown a result which in our terminology is as follows:

Theorem 5.9 *There exists a protocol $\pi_{\mathtt{BROADCAST}}$ such that $\pi_{\mathtt{BROADCAST}} \diamond \mathsf{F}_{\mathtt{PPC}}$ implements $\mathsf{F}_{\mathtt{SC}}$ in* $\mathrm{Env}^{t,sync}$ *with perfect security for all $t < n/3$.*

They have also shown that the bound on t is optimal, i.e.,

THEOREM 5.10 *There exists no protocol* $\pi_{\textsf{BROADCAST}}$ *such that* $\pi_{\textsf{BROADCAST}} \diamond \mathsf{F}_{\textsf{PPC}}$ *implements* $\mathsf{F}_{\textsf{SC}}$ *in* $\text{Env}^{t,sync}$ *with perfect or statistical security if* $t \geq n/3$.

We do not give the proofs here, as the techniques are somewhat out of scope for this book. But the intuition is not hard to understand. Consider that case where $n = 3$, $t = 1$ and P_1 wants to broadcast a value x. Let us assume that P_2 is honest. Now, if P_1 is honest, P_2 must output x. On the other hand, if P_1 is corrupt and may send different values to different players, P_2 must agree with P_3 on some value. Of course, we can ask P_1 to send x to both P_2 and P_3. But now the only thing P_2 can do to ensure he agrees with P_3 is to ask him what he heard from P_1. If he is told a value $x' \neq x$, he knows of course that one of $\mathsf{P}_1, \mathsf{P}_3$ is corrupt, but there is no way to tell who is lying, and therefore no way to decide if the output should be x or x'. The proof of Theorem 5.10 is basically a formalization of this argument. On the other hand, if $n = 4$ and $t = 1$, there are 2 other players P_2 can ask what they heard from P_1, and if P_1 is honest we are guaranteed that the correct output can be found by taking majority decision among the 3 values P_2 has seen. This is the basic reason why a broadcast protocol as claimed in Theorem 5.9 can be built.

Putting all our results for the case of $t < n/3$ together using the UC theorem we get that Theorems 5.1, 5.2, 5.6 and 5.9 imply that any function can be computed with perfect, active security if less than $n/3$ players are corrupt, given access to secure point to point channels. More formally, we have:

THEOREM 5.11 $\pi^f_{\textsf{CEAS}} \diamond \pi_{\textsf{PTRANSFER,PMULT}} \diamond \pi_{\textsf{PCOM-SIMPLE}} \diamond \pi_{\textsf{BROADCAST}} \diamond \mathsf{F}_{\textsf{PPC}}$ *implements* $\mathsf{F}^f_{\textsf{SFE}}$ *in* $\text{Env}^{t,sync}$ *with perfect security for all* $t < n/3$.

We now argue that the bound $t < n/3$ is optimal. First, note that the problem of implementing broadcast can be phrased as a problem of computing a function securely, namely a function f that takes an input x from a single player and outputs x to all players. In this language, Theorem 5.10 says that if $t \geq n/3$, there exists functions (such as f) that cannot be computed given only $\mathsf{F}_{\textsf{PPC}}$, not even if we want only statistical rather than perfect security.

In other words, Theorem 5.11 is false for $t \geq n/3$ even if we only ask for statistical security. We shall see in a moment that if we assume broadcast is given (i.e., access to $\mathsf{F}_{\textsf{SC}}$ rather than $\mathsf{F}_{\textsf{PPC}}$) and only ask for statistical security we can tolerate $t < n/2$. So the obvious remaining question is what happens if we have access to $\mathsf{F}_{\textsf{SC}}$ but insist on perfect security. The answer turns out to be negative:

Let $\mathsf{F}_{\texttt{COMMIT}}$ be the functionality defined as $\mathsf{F}_{\texttt{COM-SIMPLE}}$, but with only the `commit` and `open` commands.

THEOREM 5.12 *There exists no protocol* $\pi_{\textsf{COMMIT}}$ *such that* $\pi_{\textsf{COMMIT}} \diamond \mathsf{F}_{\textsf{SC}}$ *implements* $\mathsf{F}_{\textsf{COMMIT}}$ *in* $\text{Env}^{t,sync}$ *with perfect security if* $t \geq n/3$.

PROOF We demonstrate that no such protocol exists for $n = 3$ and $t = 1$. Assume for contradiction that we have a protocol $\pi_{\texttt{COMMIT}}$. Suppose P_1 commits to a bit b. Let $V_i(b, R)$ be the view of the execution of the `commit` command seen by player P_i where we assume all players follow the protocol and R is the random coins used by all players. By perfect security, the distribution of $V_2(b, R)$ is independent of b (P_2 learns nothing about b before opening). It follows that given a sample $V_2(0, R)$, there must exist R' such that $V_2(0, R) = V_2(1, R')$. Now consider the following two cases:

Case 0: P_1 is corrupt. He commits to $b = 0$ following the protocol, so he sees $V_1(0, R)$ for some R. Now, he tries to guess R' such that $V_2(0, R) = V_2(1, R')$, and computes $V_1(1, R')$. He then executes the `open` command following the protocol, but pretending that his view of the `commit` command was $V_1(1, R')$.

Case 1: P_1 is honest, but P_3 is corrupt. P_1 commits to $b = 1$, and P_3 follows the protocol during the execution of the `commit` command. R' are the random coins used. Now P_3 tries to guess R as defined in Case 0, computes $V_3(0, R)$, and then executes the `open` command following the protocol, but pretending that his view of the `commit` command was $V_3(0, R)$.

Obviously both cases may happen with (possibly small but) non-zero probability. Furthermore the views that are input to the `open` command are exactly the same in the two cases, so the distribution of the bit that P_2 will output is the same in both cases. However, by perfect security, P_2 must *always* output 0 (or reject) in Case 0 and 1 in Case 1, which is clearly not possible. □

The reader might complain that what we have shown impossible here is a reactive functionality, and this is a more advanced tool than secure evaluation of a function which is what the positive results talk about. However, first the protocols we have shown can also be used to implement general reactive functionalities as discussed earlier. Second, Theorem 5.12 can also be shown for secure evaluation of a function, although this is a bit more cumbersome. Consider the function that takes inputs x from P_1, y from P_2 and outputs x, y to all players. If we require that the ideal functionality for computing this function does not output anything before its gets both inputs, we can show a result similar to Theorem 5.12 for this function. The point is that even if P_1 is corrupt, he must decide on x before he gets any output, and the same holds for P_2. Hence the protocol must commit P_1 to x but without revealing any information on x to the corrupt players if P_1 is honest. This is the same property we need for commitments, so the result can be shown along the same lines. We leave the details to the reader.

The final result in this chapter concerns the case where we assume access to $\mathsf{F}_{\texttt{SC}}$ and go for statistical security. In this case, putting previous results together using the UC theorem, we get that Theorems 5.1, 5.3 and 5.8 imply that any function can be computed with statistical, active security if less than $n/2$ players are corrupt, given access to secure point to point channels and broadcast. More formally, we have:

THEOREM 5.13 *$\pi^f_{\mathtt{CEAS-N/2}} \diamond \pi_{\mathtt{TRANSFER,MULT}} \diamond \pi_{\mathtt{COM-SIMPLE}} \diamond \mathsf{F}_{\mathtt{SC}}$ implements $\mathsf{F}^f_{\mathtt{SFE}}$ in $\mathrm{Env}^{t,sync}$ with statistical security for all $t < n/2$.*

Here, $\pi^f_{\mathtt{CEAS-N/2}}$ is $\pi^f_{\mathtt{CEAS}}$ modified to handle $t < n/2$ as discussed in Section 5.3. The bound $t < n/2$ is optimal: we have already seen in Chapter 3 that secure computation with information theoretic security is not even possible with passive corruption if $t \geq n/2$.

5.6 Notes

The history of Multiparty Computation starts in 1986 with the work of Yao [181] who suggested the famous Yao-garbling technique. This technique can actually not be found in [181], but was apparently mentioned in an oral presentation of that work. The garbling technique is described and proven secure in [131]. The garbling technique shows that any function can be securely evaluated with computational security by two players.

In 1987 Goldreich, Micali and Wigderson [103] showed the same result for any number of players and honest majority, based on existence of one-way trapdoor permutations. This result was also obtained independently, but later, by Chaum, Damgaard and van de Graaf [53], based on the quadratic residuosity assumption. It should be mentioned that Yao-garbling leads to constant-round protocols, which is not the case for [103, 53]. Constant round for an arbitrary number of players was obtained in 1990 by Beaver, Micali and Rogaway [10].

While all the work mentioned so far was based on computational assumptions the work that large parts of this book is based on was started in 1988 by Ben-Or, Goldwasser and Wigderson [15] and independently by Chaum, Crépeau and Damgård [51]. They showed that in the model with secure point-to-point channels, any function can be securely evaluated with unconditional security against t corrupted players, where $t < n/2$ for a passive adversary and $t < n/3$ for an active adversary. In 1989, Ben-Or and Rabin [153] showed that if broadcast is given for free and an small error probability is allowed, $t < n/2$ can be obtained even for an active adversary.

One can also consider so-called mixed adversaries that may corrupt some players passively and others actively. Also for this case one can show optimal bounds on the number of corruptions that can be tolerated, this was done in 1998 by Fitzi, Hirt and Maurer [93]. If t_p, t_a are the number of passively and actively corrupted players, respectively, then perfectly secure general multiparty computation is possible if and only if $2t_p + 3t_a < n$.[3]

Concerning the material in this book, the passively secure protocol we give in Chapter 3 is essentially the protocol from [15], except that the way we do multiplications is different. The approach we use, where players multiply shares locally and re-share the product, was suggested by Michael Rabin, shortly after [15] was published.

[3] The reader should note that the proceedings version of [93] states a different incorrect result, the authors proved the correct result later.

The actively secure protocol from Chapter 5 is again similar to [15] in spirit, but with some major differences in how multiplications are handled. We do verifiable secret sharing by having players commit to the shares they hold in the basic secret sharing scheme, and then we implement the commitments using secret sharing again. This approach is from [67]. It makes the presentation more modular and it also allows a simple generalization to linear secret sharing. The resulting protocol is, however, less efficient compared to [15] who do verifiable secret sharing directly, without passing through a construction of commitments. This is possible because the protocol for commitment is actually already a verifiable secret sharing when the underlying secret sharing scheme is Shamir's: if a sufficiently small number of incorrect shares are submitted when we try to reconstruct, then the secret can be efficiently computed. This is not true for linear secret sharing schemes in general, so in the later chapter on linear secret sharing, the modular approach is actually necessary.

6

MPC from General Linear Secret Sharing Schemes

Contents

6.1 Introduction

In this chapter we will show how the protocols we have seen so far can be generalized to be based on general linear secret sharing schemes. Doing this generalization has several advantages: first, it allows us to design protocols that protect against general adversary structures, a concept we explain below, and second, it allows us to consider protocols where the field that is used in defining the secret sharing scheme can be of size independent of the number of players. This turns out to be an advantage for the applications of multiparty computation we consider later, but cannot be done for the protocols we have seen so far: to use Shamir's scheme for n players based on polynomials as we have seen, the field must be of size at least $n + 1$.

For now, we will only consider the theory for linear secret sharing that we need to construct the protocols in this chapter. However, we note already now that to get full mileage from the more general applications of secret sharing and MPC, we need additional theory, in particular about the asymptotic behavior of secret sharing schemes. This material can be found in Chapter 11.

6.2 General Adversary Structures

The corruption tolerance of the protocols we have seen so far have been characterized by only a single number t, the maximal number of corruptions that can be tolerated. This is also known as the threshold-t model. One way to motivate this might be to think of corruption of a player as something that requires the adversary to invest some amount of resource, such as time or money. If the adversary has only bounded resources, there should be a limit on the corruptions he can do. However, characterizing this limit by only an upper bound implicitly assumes that all players are equally hard to corrupt. In reality this may be completely false: some players may have much better security than others, so a more realistic model might be that the adversary can corrupt a small number of well protected players or a large number of poorly protected players. However, the threshold model does not allow us to express this.

Another way to interpret corruptions is to see them as a result of some subset of players working together to cheat the others. Also from this point of view, it seems quite restrictive to assume that any subset of a given size might form such a collaboration. Which subsets we should actually worry about may depend on the composition of the set of players and their relation to each other, more than on the size of the subsets.

To solve these issues, we consider the more general concept of an adversary structure. This is a family $\mathcal{A}$ of subsets of the set of players $\mathsf{P} = \{\mathsf{P}_1, \dots, \mathsf{P}_n\}$. The idea is that this is a complete list of the subsets that the adversary is able to corrupt. In the threshold-t model, $\mathcal{A}$ would contain all subsets of size at most t. But in general any list of subsets is possible, with one constraint, however: we require that $\mathcal{A}$ be anti-monotone, namely $A \subseteq B$ and $B \in \mathcal{A}$ implies that $A \in \mathcal{A}$. The meaning of this is that if the adversary is powerful enough to corrupt set B, he can also choose to corrupt any smaller set. We characterize two important classes of adversary structures:

Definition 6.1 *An adversary structure $\mathcal{A}$ is said to be $Q2$ if for all $A_1, A_2 \in \mathcal{A}$ it holds that $A_1 \cup A_2 \neq \mathsf{P}$. It is said to be $Q3$ if for all $A_1, A_2, A_3 \in \mathcal{A}$ it holds that $A_1 \cup A_2 \cup A_3 \neq \mathsf{P}$.*

The $Q2$ and $Q3$ conditions are natural generalizations of the threshold-t model, for $t < n/2$ and $t < n/3$, respectively. For instance, the union of two sets that are both smaller than $n/2$ cannot be the entire player set P.

We can then define what it means that a protocol implements some functionality securely against an adversary structure $\mathcal{A}$. This is done in exactly the same way as we defined security for at most t corruptions, except that we now consider environments that corrupt only subsets in $\mathcal{A}$.

We have seen that multiparty computation with perfect, passive security in the threshold-t model is possible if and only if $t < n/2$ and with active security if and only if $t < n/3$. The following is then easy to prove:

Theorem 6.2 *For any adversary structure $\mathcal{A}$ that is not $Q2$, there are functions* *that cannot be securely computed with passive and perfect security against $\mathcal{A}$. Moreover, for any adversary structure $\mathcal{A}$ that is not $Q3$, there are functions that cannot be securely computed with active and perfect security against $\mathcal{A}$.*

To prove this, one notices that the corresponding proofs for $t \geq n/2$ and $t \geq n/3$ actually only use the fact the the underlying adversary structures are not $Q2$, respectively $Q3$. This is straightforward and is left to the reader, see for instance Section 3.4.

We shall see later that the converse is also true: multiparty computation with perfect, passive security against adversary structure $\mathcal{A}$ is possible if $\mathcal{A}$ is $Q2$ and with active security if $\mathcal{A}$ is $Q3$.

The following examples illustrate that we can achieve strictly more with general adversary structures that what can be done in the threshold case:

Example 6.1 Suppose we have 6 players, and consider the adversary structure containing the sets $\{\mathsf{P}_1\}, \{\mathsf{P}_2, \mathsf{P}_4\}, \{\mathsf{P}_2, \mathsf{P}_5, \mathsf{P}_6\}, \{\mathsf{P}_3, \mathsf{P}_5\}, \{\mathsf{P}_3, \mathsf{P}_6\}, \{\mathsf{P}_4, \mathsf{P}_5, \mathsf{P}_6\}$, and all smaller sets. It is trivial to see that this structure is $Q3$, so we can do multiparty computation with active security in this case. Note that if the threshold-t model with $t < n/3$ was all we could do, we could only tolerate corruption of a single player. $\triangle$

Example 6.2 Here is an infinite family of examples: suppose our player set is divided into two disjoint sets X and Y of m players each ($n = 2m$) where the players are on friendly terms within each group but tend to distrust players in the other group. Hence, a coalition of active cheaters might consist of almost all players from X or from Y, whereas a mixed coalition with players from both groups is likely to be quite small. Concretely, suppose we assume that a corrupted set can consist of at most $9m/10$ players from only X or only Y, *or* it can consist of less than $m/5$ players coming from both X and Y. This defines a $Q3$ adversary structure, and so multi-party computation with active security is possible in this scenario. Nevertheless, no threshold solution exists, since the largest coalitions of corrupt players have size more than $n/3$. The intuitive reason why multiparty computation is nevertheless possible is that although some corruptible sets are larger than $n/3$, we do not need to protect against corruption of *all* sets larger than $n/3$. $\triangle$

6.3 Linear Secret-Sharing

Let us consider how we can realize multiparty computation that is secure if the adversary may corrupt subsets of players in some adversary structure, that is not necessarily threshold. One might think that a solution could be to use Shamir secret sharing, generate more shares than there are players, and give a different number of shares to different players. At least this will imply that we can tolerate corruption of a larger number of those players that received only a small number of shares. This approach would allow us to handle some, but not all relevant adversary structures, and it will often be suboptimal in terms of efficiency.

A much better solution is to generalize Shamir's scheme further: Say we share the secret s among n players in the usual way using a polynomial $f_s(\mathtt{X})$. We compute shares by evaluating f_s in points $1, 2, \ldots, n$. This computation of the shares can be rephrased as follows: Consider a Van der Monde matrix M which by definition is a matrix where the rows are of form $(i^0, i^1, i^2, \ldots, i^t)$ for $i = 1, \ldots, n$. Now define a column vector $\mathbf{r}_s$ whose entries are the coefficients of f_s, with s (the degree-0 coefficient) as the first entry. Now, computing the product $M \cdot \mathbf{r}_s$ is clearly equivalent to evaluating f_s in points $1, \ldots, n$. Therefore we can rephrase Shamir's scheme as follows: choose a random vector with $t + 1$ entries subject to the secret s appearing as first coordinate. Compute $M\mathbf{r}_s$ and give the i-th entry of the result to player P_i.

A generalization now quite naturally suggests itself: first, why not consider other matrices than Van der Monde, in particular with more than n rows? Second, once we do this, we can assign more than one row to each player, so they may receive more than one value in the field. This generalization exactly leads to the linear secret sharing schemes we consider in this chapter.

Before we define these schemes formally, some notation: for consistency in the following, vectors are column vectors unless we state otherwise, and we will sometimes use $\cdot$ to denote matrix multiplication. Thus we will sometimes write the inner product of $\mathbf{a}, \mathbf{b}$ as $\mathbf{a}^{\mathsf{T}} \cdot \mathbf{b}$.

Definition 6.3 *A linear secret sharing scheme $\mathcal{S}$ over a field $\mathbb{F}$ for n players is defined by a matrix M and a function ϕ. M has $m \geq n$ rows and $t + 1$ columns, and ϕ is a labeling function $\phi : \{1, \ldots, m\} \mapsto \{1, \ldots, n\}$. We say M is the matrix for $\mathcal{S}$. We can apply ϕ to the rows of M in a natural way, and we say that player $P_{\phi(i)}$ owns the i-th row of M. For a subset A of the players, we let M_A be the matrix consisting of the rows owned by players in A.*

To secret-share $s \in \mathbb{F}$ using $\mathcal{S}$, one forms a column vector $\mathbf{r}_s$ with s as the first coordinate, while the other coordinates are $r_1, \ldots, r_t$ where each r_i is uniformly random in $\mathbb{F}$. Finally one computes the vector of shares $M\mathbf{r}_s$ and distributes them among players, by giving $(M\mathbf{r}_s)_i$ to player $P_{\phi(i)}$. Note that in this way, a player may get more than one value as his share. Note also that for a subset A of players, $M_A\mathbf{r}_s$ is the set of values received by players in A when s is shared.

Definition 6.4 *The adversary structure of $\mathcal{S}$ is a family of subsets of players. A set A is in the adversary structure if and only if the distribution of $M_A\mathbf{r}_s$ is independent of s. Such a set is called unqualified. The access structure of $\mathcal{S}$ is also a family of subsets of players. A set A is in the access structure if and only if s is uniquely determined from $M_A\mathbf{r}_s$. Such a set is called qualified.*

It follows from Theorem 6.6 below that every player subset is either qualified or unqualified.

We will let $\mathbf{m}_k$ denote the k-th row of M. When we need to talk about individual entries in M, it turns out to be convenient to number the columns from

0, and hence the k-th row of M can also be written as

$$\mathbf{m}_k = (M_{k,0}, M_{k,1}, \ldots, M_{k,t})$$

This also gives an alternative way to address single shares that will sometimes be convenient, namely we have:

$$(M \cdot \mathbf{r}_a)_k = \mathbf{m}_k \cdot \mathbf{r}_a$$

We will need the following basic fact from linear algebra, where we recall that $\mathrm{im}(N)$ for a matrix N is the set of all vectors of form $N\mathbf{v}$, or put differently; the set of vectors that are obtained as linear combinations of the columns of N.

LEMMA 6.5 *For any matrix M and vector $\mathbf{e}$ we have that $\mathbf{e} \notin \mathrm{im}(M^{\mathsf{T}})$ if and only if there exists $\mathbf{w}$ with $\mathbf{w} \in \ker(M)$ and $\mathbf{w}^{\mathsf{T}} \cdot \mathbf{e} \neq 0$.*

PROOF For a subspace V of any vector space over $\mathbb{F}$, $V^{\perp}$ denotes the orthogonal complement of V, the subspace of vectors that are orthogonal to all vectors in V. Now, notice that $\mathrm{im}(M^{\mathsf{T}})$ is the space spanned by the rows of M. Therefore, it is clear that $\ker(M) = \mathrm{im}(M^{\mathsf{T}})^{\perp}$, since computing $M \cdot \mathbf{a}$ for some $\mathbf{a}$ is equivalent to computing inner product of $\mathbf{a}$ with every row of M. From this and $V = (V^{\perp})^{\perp}$ for any subspace V it follows that $\ker(M)^{\perp} = \mathrm{im}(M^{\mathsf{T}})$. So $\mathbf{e} \notin \mathrm{im}(M^{\mathsf{T}})$, if and only if $\mathbf{e} \notin \ker(M)^{\perp}$, and this holds if and only if $\mathbf{w}$ as in the statement of lemma exists. □

THEOREM 6.6 *Let $\mathcal{S}$ be a linear secret sharing scheme and let M be the matrix for $\mathcal{S}$. A player subset A is qualified in $\mathcal{S}$ if and only if $\mathbf{e} = (1, 0, \ldots, 0)^{\mathsf{T}} \in \mathrm{im}(M_A^{\mathsf{T}})$, i.e., the subspace spanned by the rows of M_A contains $\mathbf{e}$. In this case there exists a* reconstruction vector *$\mathbf{u}$ such that $\mathbf{u}^{\mathsf{T}}(M_A \mathbf{r}_s) = s$ for all s. A is unqualified in $\mathcal{S}$ if and only if $\mathbf{e} \notin \mathrm{im}(M_A^{\mathsf{T}})$. In this case there exists a vector $\mathbf{w}$ that is orthogonal to all rows in M_A and has first coordinate 1.*

PROOF If $\mathbf{e} \in \mathrm{im}(M_A^{\mathsf{T}})$ there exists a vector $\mathbf{u}$ such that $M_A^{\mathsf{T}} \cdot \mathbf{u} = \mathbf{e}$, or $\mathbf{u}^T \cdot M_A = \mathbf{e}^{\mathsf{T}}$. But then we have

$$\mathbf{u}^{\mathsf{T}} \cdot (M_A \cdot \mathbf{r}_s) = (\mathbf{u}^{\mathsf{T}} \cdot M_A) \cdot \mathbf{r}_s = \mathbf{e}^{\mathsf{T}} \cdot \mathbf{r}_s = s \ .$$

If $\mathbf{e}$ is not in the span of the rows of M_A, then by Lemma 6.5, there exists a vector $\mathbf{w}$ that is orthogonal to all rows of M_A but not orthogonal to $\mathbf{e}$. This can be used as the $\mathbf{w}$ claimed in the theorem since we may assume without loss of generality that the inner product $\mathbf{w}^{\mathsf{T}} \cdot \mathbf{e}$ is 1. Let s, s' be any two possible values of the secret to be shared. For any $\mathbf{r}_s$, let $\mathbf{r}_{s'} = \mathbf{r}_s + (s' - s)\mathbf{w}$. Then the first coordinate of $\mathbf{r}_{s'}$ is s' and we have

$$M_A \cdot \mathbf{r}_{s'} = M_A \cdot \mathbf{r}_s \ .$$

That is, for each set of random coins used to share s there is exactly one set of coins for sharing s' such that the players in A receive the same shares. Since the randomness is chosen uniformly it follows that the distribution of shares received by A is the same for all values of the secret, and so A is in the adversary structure. □

In the following, we will refer to m, the number of rows in M as the size of the secret sharing scheme defined by M. The reason it makes sense to ignore t (the number of columns) in this connection is that we can assume without loss of generality that $t + 1 \leq m$: given any M we could always consider a new matrix M' containing only a maximal independent set of columns from M (including the first column). Because the column-rank equals the row-rank for any matrix, such a set will have size at most m. Moreover, it will lead to a secret sharing scheme with the same access structure. This follows from Theorem 6.6 which says that a set A is qualified exactly if the rows of M_A span the first basis vector. This happens if and only if the rows of M'_A span the first basis vector, because the columns in M_A can all be obtained as linear combinations of the columns in M'_A. In the following are will always assume that the columns of M are independent.

In the following we will reuse some notation from earlier chapters and write

$$M\mathbf{r}_s = [s; \mathbf{r}_s]_{\mathcal{S}} \ .$$

Using the standard entrywise addition and multiplication by a scalar the following vectors are well defined (for $\alpha, a, b \in \mathbb{F}$):

$$[a; \mathbf{r}_a]_{\mathcal{S}} + [b; \mathbf{r}_b]_{\mathcal{S}} \ , \quad \alpha[a; \mathbf{r}_a]_{\mathcal{S}} \ .$$

We now want to consider multiplication of shared values. To understand how this works, recall how we handled secure multiplication with Shamir's secret sharing scheme in previous chapters: say $[a; f_a]_t, [b; f_b]_t$ have been distributed and we want to compute ab in shared form. The main observation that we used for this was that if each player locally multiplies his share of a and of b, then the local products form a set of shares in ab, namely they form $[ab; f_a f_b]_{2t}$. This new sharing is of different type, however, it is formed by a polynomial of degree $2t$ and not t.

It turns out that something similar happens for general linear secret-sharing. Assume that $[a; \mathbf{r}_a]_{\mathcal{S}}, [b; \mathbf{r}_b]_{\mathcal{S}}$ have been distributed. We now want to define what it means for players to locally multiply their shares of a and of b. This gets more complicated than in the case of polynomials because a player may have more than one field value as his share of a (and of b). Therefore there may be several different pairwise products he can compute, and we need notation to capture all of these. We therefore define a special type of product, written

$$[a; \mathbf{r}_a]_{\mathcal{S}} \odot [b; \mathbf{r}_b]_{\mathcal{S}}$$

This vector is constructed so it contains all products of a share of a and a share of b that some player can compute locally. More formally

$$[a; \mathbf{r}_a]_{\mathcal{S}} \odot [b; \mathbf{r}_b]_{\mathcal{S}} = (([a; \mathbf{r}_a]_{\mathcal{S}})_{\{\mathsf{P}_1\}} \otimes ([b; \mathbf{r}_b]_{\mathcal{S}})_{\{\mathsf{P}_1\}}, \ldots, (([a; \mathbf{r}_a]_{\mathcal{S}})_{\{\mathsf{P}_n\}} \otimes ([b; \mathbf{r}_b]_{\mathcal{S}})_{\{\mathsf{P}_n\}}) \ ,$$

where $([a; \mathbf{r}_a]_{\mathcal{S}})_{\{\mathsf{P}_i\}}$ denotes the vector containing all shares of a given to P_i, and where for $\mathbf{a}, \mathbf{b} \in \mathbb{F}^\ell$, $\mathbf{a} \otimes \mathbf{b}$ is the tensor product of vectors, i.e., $\mathbf{a} \otimes \mathbf{b} \in \mathbb{F}^{\ell^2}$ and contains all (ordered) pairwise products of entries in the two vectors.

Note that if $n = m$ and every player owns one value we can set $\phi(i) = i$. In this case, $[a; \mathbf{r}_a]_\mathcal{S} \odot [b; \mathbf{r}_b]_\mathcal{S}$ is just the Schur product $[a; \mathbf{r}_a]_\mathcal{S} * [b; \mathbf{r}_b]_\mathcal{S}$. In the following, $\hat{m}$ will denote the length of $[a; \mathbf{r}_a]_\mathcal{S} \odot [b; \mathbf{r}_b]_\mathcal{S}$.

6.3.1 Multiplicative Secret-Sharing

As we hinted at above, if values a, b have been secret shared using $\mathcal{S}$, it turns out that the vector $[a; \mathbf{r}_a]_\mathcal{S} \odot [b; \mathbf{r}_b]_\mathcal{S}$ forms a set of shares of ab, albeit in a different linear secret-sharing scheme, which we will denote $\hat{\mathcal{S}}$. We specify $\hat{\mathcal{S}}$ by directly constructing its matrix $\hat{M}$: Consider the rows of M owned by P_i, i.e., $M_{\{\mathsf{P}_i\}}$. We now define $\hat{M}_{\{\mathsf{P}_i\}}$ to be a matrix consisting of all rows of form $\mathbf{u} \otimes \mathbf{v}$ where $\mathbf{u}$ and $\mathbf{v}$ are rows in $M_{\{\mathsf{P}_i\}}$. Then $\hat{M}$ is built by putting all the $\hat{M}_{\{\mathsf{P}_i\}}$ together in one matrix. The labeling function is defined such that P_i owns exactly the rows that came from $\hat{M}_{\{\mathsf{P}_i\}}$.

With this extension of the labeling function, we get the following simple fact that we will use later:

Lemma 6.7 *Let c_k be the k-th entry in $[a; \mathbf{r}_a]_\mathcal{S} \odot [b; \mathbf{r}_b]_\mathcal{S}$. Then there exist (fixed) indices u, v such that $c_k = a_u b_v$ where a_u is the u-th entry in $[a; \mathbf{r}_a]_\mathcal{S}$, b_v is the v-th entry in $[b; \mathbf{r}_b]_\mathcal{S}$ and $\phi(k) = \phi(u) = \phi(v)$.*

The main reason we introduce $\hat{\mathcal{S}}$ is that it can be used to implement multiplication of shared secrets:

Lemma 6.8 *For any linear secret sharing scheme $\mathcal{S}$ we have*

$$[a; \mathbf{r}_a]_\mathcal{S} \odot [b; \mathbf{r}_b]_\mathcal{S} = [ab; \mathbf{r}_a \otimes \mathbf{r}_b]_{\hat{\mathcal{S}}}$$

Proof Notice first that if $\mathbf{u}, \mathbf{v}$ are rows in $M_{\{\mathsf{P}_i\}}$, then $\mathbf{u} \cdot \mathbf{r}_a, \mathbf{v} \cdot \mathbf{r}_b$ are values P_i will receive when secrets a, b are shared, i.e., they are entries in $([a; \mathbf{r}_a]_\mathcal{S})_{\{\mathsf{P}_i\}}$ and $([b; \mathbf{r}_b]_\mathcal{S})_{\{\mathsf{P}_i\}}$, respectively.

Therefore, the entries in $([a; \mathbf{r}_a]_\mathcal{S})_{\{\mathsf{P}_i\}} \otimes ([b; \mathbf{r}_b]_\mathcal{S})_{\{\mathsf{P}_i\}}$ are of form

$$(\mathbf{u} \cdot \mathbf{r}_a) \cdot (\mathbf{v} \cdot \mathbf{r}_b) = (\mathbf{u} \otimes \mathbf{v}) \cdot (\mathbf{r}_a \otimes \mathbf{r}_b).$$

The lemma now follows by definition of $\hat{M}$. □

Note that the adversary structure of $\hat{\mathcal{S}}$ contains that of $\mathcal{S}$. This is because any player set A can locally compute their part of $[ab; \mathbf{r}_a \otimes \mathbf{r}_b]_{\hat{\mathcal{S}}}$ after secrets a, b have been shared. If players in A could reconstruct ab from this, this reveals information on a and b, which cannot be the case if A is in the adversary structure of $\mathcal{S}$.

Definition 6.9 *If the access structure of $\hat{\mathcal{S}}$ contains the set of all players, $\mathcal{S}$ is said to be* multiplicative. *If the access structure of $\hat{\mathcal{S}}$ contains the complement* *of every set in the adversary structure $\mathcal{A}$ of $\mathcal{S}$, then $\mathcal{S}$ is said to be* strongly multiplicative.

Note that if $\mathcal{S}$ is multiplicative then there exists a reconstruction vector $\mathbf{u}$ such that

$$\mathbf{u} \cdot ([a; \mathbf{r}_a]_{\mathcal{S}} \odot [b; \mathbf{r}_b]_{\mathcal{S}}) = \mathbf{u} \cdot [ab; \mathbf{r}_a \otimes \mathbf{r}_b]_{\hat{\mathcal{S}}} = ab \ .$$

A way to think about the strong multiplication property is as follows: if an adversary is able to corrupt a set of players in $\mathcal{A}$ (but not more than that), then the set of honest players is always the complement of a set in $\mathcal{A}$. And then strong multiplication says that the set of honest players can on their own reconstruct ab from $[ab; \mathbf{r}_a \otimes \mathbf{r}_b]_{\hat{\mathcal{S}}}$.

The reader should take a moment to understand that the multiplication property indeed generalizes in a natural way what we know about secret sharing using polynomials, as we promised above: the secret-sharing scheme $\mathcal{S}$ corresponds to sharing using polynomials of degree at most t, whereas $\hat{\mathcal{S}}$ corresponds to sharing using polynomials of degree at most $2t$. Likewise, $\mathbf{r}_s$ corresponds to the coefficients of a polynomial that evaluates to s in 0, and $\mathbf{r}_a \otimes \mathbf{r}_b$ corresponds to multiplication of polynomials.

Summarizing what we have seen, we have the following:

Lemma 6.10 *The following holds for any $a, b, \alpha \in \mathbb{F}$ and any multiplicative linear secret-sharing scheme $\mathcal{S}$:*

$$[a; \mathbf{r}_a]_{\mathcal{S}} + [b; \mathbf{r}_b]_{\mathcal{S}} = [a + b; \mathbf{r}_a + \mathbf{r}_b]_{\mathcal{S}} \ , \tag{6.1}$$

$$\alpha[a; \mathbf{r}_a]_{\mathcal{S}} = [\alpha a; \alpha\mathbf{r}_a]_{\mathcal{S}} \ , \tag{6.2}$$

$$[a; \mathbf{r}_a]_{\mathcal{S}} \odot [b; \mathbf{r}_b]_{\mathcal{S}} = [ab; \mathbf{r}_a \otimes \mathbf{r}_b]_{\hat{\mathcal{S}}} \ . \tag{6.3}$$

6.3.2 On Existence of Linear Secret Sharing Schemes

We start by the most basic facts on existence of linear secret sharing schemes:

Theorem 6.11

- *For every adversary structure $\mathcal{A}$, there exists a linear secret sharing-scheme $\mathcal{S}$ whose adversary structure is $\mathcal{A}$.*
- *For every $Q2$-structure $\mathcal{A}$, there exists a multiplicative linear secret sharing-scheme $\mathcal{S}$ whose adversary structure is $\mathcal{A}$.*
- *For every $Q3$-structure $\mathcal{A}$, there exists a strongly multiplicative linear secret sharing-scheme $\mathcal{S}$ whose adversary structure is $\mathcal{A}$.*

This theorem can be shown using just a single scheme, namely so-called replicated secret sharing which works as follows. Given adversary structure $\mathcal{A}$ and secret $s \in \mathbb{F}$ to share, do the following: for each set $A \in \mathcal{A}$, choose $r_A \in \mathbb{F}$ at random, subject to the constraint that $s = \sum_{A \in \mathcal{A}} r_A$. Then for all $A \in \mathcal{A}$, give r_A to all players *not* in A. We will call this scheme $\mathcal{R}_{\mathcal{A}}$. Note that it can be defined over any field, independently of the number of players.

Exercise 6.1 Show that $\mathcal{R}_{\mathcal{A}}$ is a linear secret sharing scheme with adversary structure $\mathcal{A}$.

Exercise 6.2 Show that $\mathcal{R}_{\mathcal{A}}$ is multiplicative if $\mathcal{A}$ is $Q2$.

Exercise 6.3 Show that $\mathcal{R}_{\mathcal{A}}$ is strongly multiplicative if $\mathcal{A}$ is $Q3$.

Of course, $\mathcal{R}_{\mathcal{A}}$ is in general very inefficient in the sense that most adversary structures contain a number of sets that is exponential in the number of players and so players will receive a very large number of shares. On the other hand, Shamir's scheme with threshold t is linear, and is easily seen to be multiplicative, respectively strongly multiplicative if $t < n/2$, respectively $t < n/3$. In this scheme players get only a single field element as their share, so we see that the share size does not have to depend on the size of the adversary structure, even if we also require (strong) multiplication.

It would therefore be great if we could characterize those access structures that admit efficient linear schemes, i.e., of size polynomial in the number of players. However, this question is completely open. But we can at least say that asking for multiplication does not incur a large cost over just asking for a linear scheme. This is formalized in Theorem 6.14 below. However, we first need a new notion:

Definition 6.12 *Let $\mathcal{F}$ be an access structure. Then the* **dual access structure** $\bar{\mathcal{F}}$ *is defined as follows: a set A is in $\bar{\mathcal{F}}$ if and only if the complement $\bar{A}$ is not in $\mathcal{F}$.*

For instance, if $\mathcal{F}$ contains all sets of size larger than t, then $\bar{\mathcal{F}}$ contains all sets of size larger than $n - t$.

Exercise 6.4 Let $\mathcal{S}$ be a secret sharing scheme with adversary structure $\mathcal{A}$ and access structure $\mathcal{F}$. Show that if $\mathcal{A}$ is $Q2$, then $\bar{\mathcal{F}} \subseteq \mathcal{F}$.

Lemma 6.13 *Given any linear secret sharing scheme $\mathcal{S}$ with access structure $\mathcal{F}$, one can construct a linear secret sharing scheme $\bar{\mathcal{S}}$ with access structure $\bar{\mathcal{F}}$ in time polynomial in the size of $\mathcal{S}$. Moreover, the matrices $M, \bar{M}$ of $\mathcal{S}$ and $\bar{\mathcal{S}}$ satisfy $M^\mathsf{T}\bar{M} = E$, where E is a matrix with 1 in the top-left corner and 0 elsewhere.*

Proof Let M be the matrix of $\mathcal{S}$, let $\mathbf{v}_0$ be a solution to $M^\mathsf{T} \cdot \mathbf{v}_0 = \mathbf{e}$, and let $\mathbf{v}_1, \ldots, \mathbf{v}_{m-(t+1)}$ be a basis for $\ker(M^\mathsf{T})$.

We now let the matrix $\bar{M}$ for $\bar{\mathcal{S}}$ be the matrix with columns $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{m-(t+1)}$, and with the same labeling function as for $\mathcal{S}$. We also define for later $\bar{\mathbf{e}} \in \mathbb{F}^{m-(t+1)+1}$ to be $(1, 0, \ldots, 0)^\mathsf{T}$. Then $M^\mathsf{T}\bar{M} = E$ follows immediately by construction of $\bar{M}$.

We now want to show that $\bar{\mathcal{F}}$ is indeed the access structure of $\bar{\mathcal{S}}$, i.e., $A \in \mathcal{F}$ if and only if $\bar{A}$ is not in the access structure of $\bar{\mathcal{S}}$.

If $A \in \mathcal{F}$, the rows of M_A span $\mathbf{e}$, or equivalently, there exists $\mathbf{v}$ such that $\mathbf{v}_{\bar{A}} = 0$ and $M^\mathsf{T} \cdot \mathbf{v} = \mathbf{e}$, where $\mathbf{v}_{\bar{A}}$ is the vector we obtain by selecting from $\mathbf{v}$ the coordinates owned by players in $\bar{A}$. By construction of $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{m-(t+1)}$ any such $\mathbf{v}$ can be written as $\mathbf{v_0}$ plus some linear combination of $\mathbf{v}_1, \ldots, \mathbf{v}_{m-(t+1)}$, or equivalently there exists a vector $\mathbf{w}$ with 1 as its first coordinate such that $\mathbf{v} = \bar{M} \cdot \mathbf{w}$. But then $\bar{M}_{\bar{A}} \cdot \mathbf{w} = \mathbf{v}_{\bar{A}} = 0$, and $\mathbf{w}^{\mathsf{T}} \cdot \bar{\mathbf{e}} = 1 \neq 0$. By Lemma 6.5, the existence of such a $\mathbf{w}$ shows that the rows of $\bar{M}_{\bar{A}}$ do not span $\bar{\mathbf{e}}$, and so $\bar{A}$ is not in the access structure of $\bar{\mathcal{S}}$.

Conversely, if $\bar{A}$ is not in the access structure of $\bar{\mathcal{S}}$, there exists $\mathbf{w}$ with $\bar{M}_{\bar{A}} \cdot \mathbf{w} = 0$ and $\mathbf{w}^{\mathsf{T}} \cdot \bar{\mathbf{e}} = 1$. If we set $\mathbf{v} = \bar{M} \cdot \mathbf{w}$ then $\mathbf{v}_{\bar{A}} = 0$, and so

$$M_A^{\mathsf{T}} \cdot \mathbf{v}_A = M^{\mathsf{T}} \cdot \mathbf{v} = M^{\mathsf{T}} \cdot \bar{M} \cdot \mathbf{w} = E \cdot \mathbf{w} = \mathbf{e} \ .$$

This shows that A is in $\mathcal{F}$. □

Theorem 6.14 *From every linear secret sharing scheme $\mathcal{S}$ with $Q2$-adversary structure $\mathcal{A}$, we can construct a multiplicative linear secret sharing scheme $\mathcal{S}'$ whose adversary structure is $\mathcal{A}$ in time polynomial in the size of $\mathcal{S}$. The size of $\mathcal{S}'$ is at most twice that of $\mathcal{S}$.*

Proof We begin by using Lemma 6.13 to construct $\bar{\mathcal{S}}$ with access structure $\bar{\mathcal{F}}$.

We can now start describing the desired new scheme $\mathcal{S}'$. As a first step consider a secret sharing scheme where we share a secret s according to both $\mathcal{S}$ and $\bar{\mathcal{S}}$. More formally, we generate vectors $\mathbf{r}_s$ and $\bar{\mathbf{r}}_s$, at random but with s as first coordinate, compute $M \cdot \mathbf{r}_s$, $\bar{M} \cdot \bar{\mathbf{r}}_s$ and distribute shares according to the labeling function. Now, from Exercise 6.4, we know that $\bar{\mathcal{F}} \subseteq \mathcal{F}$ and this implies that the access structure of our new scheme is $\mathcal{F}$.

Now assume we have shared secrets a, b according to this scheme. In particular this means that players hold $M \cdot \mathbf{r}_a$ and $\bar{M} \cdot \bar{\mathbf{r}}_b$. Note that because we use the same labeling function for M and $\bar{M}$, the entries in the vector $(M \cdot \mathbf{r}_a) * (\bar{M} \cdot \bar{\mathbf{r}}_b)$ can be computed locally by the players. We claim that the sum of all these entries is actually ab. Since the sum of the entries in the $*$-product is just the inner product, this can be seen from:

$$(M \cdot \mathbf{r}_a)^{\mathsf{T}} \cdot (\bar{M} \cdot \bar{\mathbf{r}}_b) = \mathbf{r}_a^{\mathsf{T}} \cdot (M^{\mathsf{T}} \cdot \bar{M}) \cdot \bar{\mathbf{r}}_b = \mathbf{r}_a^{\mathsf{T}} \cdot E \cdot \bar{\mathbf{r}}_b = ab \ . \qquad (6.4)$$

by Lemma 6.13

We can now finally specify $\mathcal{S}'$ by directly constructing its matrix M', which will have $2m$ rows and $2(t+1)$ columns. First build a matrix M'' filled in with M in the upper left corner, $\bar{M}$ in the lower right corner, and 0's elsewhere. Let $\mathbf{k}$ be the column in M'' that passes through the first column of $\bar{M}$. Add $\mathbf{k}$ to the first column of M'' and delete $\mathbf{k}$ from M''. Let M' be the result of this. The labeling of M' is carried over in the natural way from M and $\bar{M}$.

It is now clear that secret sharing according to $\mathcal{S}'$ will generate shares from both $\mathcal{S}$ and $\bar{\mathcal{S}}$ as described above and hence by (6.4), the set of all players will be qualified in $\hat{\mathcal{S}}'$, and this is exactly the requirement for $\mathcal{S}'$ to be multiplicative. □

We end this section by showing that multiplicative schemes can only exist for $Q2$ structures:

THEOREM 6.15 *Let $\mathcal{S}$ be a multiplicative linear secret sharing scheme. Then the adversary structure for $\mathcal{S}$ is $Q2$.*

PROOF Suppose for contradiction that the adversary structure $\mathcal{A}$ is not $Q2$. Then there are two sets $A_1, A_2 \in \mathcal{A}$ such that $A_1 \cup A_2 = \mathsf{P}$. By Lemma 6.5 there exists vectors $\mathbf{w}_1, \mathbf{w}_2$ with $\mathbf{w}_1 \in \ker(M_{A_1})$ and $\mathbf{w}_2 \in \ker(M_{A_2})$ and with non-zero first coordinates, that we can assume without loss of generality are both 1. We can think of $M \cdot \mathbf{w}_1$ and $M \cdot \mathbf{w}_2$ as two sets of shares of the secret 1. Hence by the multiplicative property we should be able to reconstruct the product 1 as a linear combination of the entries in $(M \cdot \mathbf{w}_1) \odot (M \cdot \mathbf{w}_2)$. But since $(M \cdot \mathbf{w}_1)_{A_1}$ and $(M \cdot \mathbf{w}_2)_{A_2}$ are both 0 and $A_1 \cup A_2 = \mathsf{P}$, $(M \cdot \mathbf{w}_1) \odot (M \cdot \mathbf{w}_2)$ is in fact the all-0 vector and we have a contradiction. $\square$

EXERCISE 6.5 Let $\mathcal{S}$ be a strongly multiplicative linear secret sharing scheme. Show that the adversary structure for $\mathcal{S}$ is $Q3$.

An obvious question is whether a result similar to Theorem 6.14 holds for strongly multiplicative secret sharing, i.e., from a linear secret sharing scheme for a $Q3$ adversary structure, can one construct a strongly multiplicative scheme of similar size? This question is wide open, it has resisted attempts to solve it for more than a decade, and no clear evidence is known either way as to what the answer might be.

6.4 A Passively Secure Protocol

In this section we show a natural generalization of the passively secure protocol, we have seen earlier, Protocol CEPS (CIRCUIT EVALUATION WITH PASSIVE SECURITY). We basically replace polynomials by linear secret sharing and do "the same thing". First a definition:

DEFINITION 6.16 *When we say that a player P_i distributes $[s; \mathbf{r}_s]_{\mathcal{S}}$, we means that he calculates $[s; \mathbf{r}_s]_{\mathcal{S}}$ as described above and distributes the shares to the players according to the labeling function ϕ. Whenever players have obtained shares of a value a, based on polynomial $\mathbf{r}_a$, we say that the players hold $[s; \mathbf{r}_s]_{\mathcal{S}}$.*

Suppose the players hold $[a; \mathbf{r}_a]_{\mathcal{S}}$, $[b; \mathbf{r}_b]_{\mathcal{S}}$. Then, when we say that the players compute $[a; \mathbf{r}_a]_{\mathcal{S}} + [b; \mathbf{r}_b]_{\mathcal{S}}$, this means that each player P_i computes the sum of corresponding shares of a and b that he holds, and by Lemma 6.10, this means that the players now hold $[a + b; \mathbf{r}_a + \mathbf{r}_b]_{\mathcal{S}}$. In a similar way we define what it means for the players to compute $[a; \mathbf{r}_a]_{\mathcal{S}} \odot [b; \mathbf{r}_b]_{\mathcal{S}}$ and $\alpha \cdot [a; \mathbf{r}_a]_{\mathcal{S}}$.

We can now describe the protocol for securely evaluating an arithmetic circuit using a general linear secret-sharing scheme $\mathcal{S}$, see Protocol GCEPS (GENERAL CIRCUIT EVALUATION WITH PASSIVE SECURITY). The only condition we need is that $\mathcal{S}$ is multiplicative.

We can then show the following result, where we let π^f_{GCEPS} denote the GCEPS-protocol instantiated for computing an arithmetic circuit for the function f, and

Protocol GCEPS (General Circuit Evaluation with Passive Security)

The protocol proceeds in three phases: the input sharing, computation and output reconstruction phases.

Input Sharing: Each player P_i holding input $x_i \in \mathbb{F}$ distributes $[x_i; \mathbf{r}_{x_i}]_{\mathcal{S}}$.

We then go through the circuit and process the gates one by one in the computational order. Just after the input sharing phase, we consider all input gates as being processed. We will maintain the following:

Invariant: Recall that computing with the circuit on inputs $x_1, \ldots, x_n$ assigns a unique value to every wire. Consider an input or an output wire for any gate, and let $a \in \mathbb{F}$ be the value assigned to this wire. Then, if the gate has been processed, the players hold $[a; \mathbf{r}_a]_{\mathcal{S}}$ for some vector $\mathbf{r}_a$.

We then continue with the last two phases of the protocol:

Computation Phase: Repeat the following until all gates have been processed (then go to the next phase): Consider the first gate in the computational order that has not been processed yet. According to the type of gate, do one of the following

Addition gate: The players hold $[a; \mathbf{r}_a]_{\mathcal{S}}$, $[b; \mathbf{r}_b]_{\mathcal{S}}$. for the two inputs a, b to the gate. The players compute $[a; \mathbf{r}_a]_{\mathcal{S}} + [b; \mathbf{r}_b]_{\mathcal{S}} = [a+b; \mathbf{r}_a + \mathbf{r}_b]_{\mathcal{S}}$.

Multiply-by-constant gate: The players hold $[a; \mathbf{r}_a]_{\mathcal{S}}$ for the inputs a to the gate. The players compute $\alpha[a; \mathbf{r}_a]_{\mathcal{S}} = [\alpha a; \alpha \mathbf{r}_a]_{\mathcal{S}}$.

Multiplication gate: The players hold $[a; \mathbf{r}_a]_{\mathcal{S}}$, $[b; \mathbf{r}_b]_{\mathcal{S}}$ for the two inputs a, b to the gate.

1. The players compute $[a; \mathbf{r}_a]_{\mathcal{S}} \odot [b; \mathbf{r}_b]_{\mathcal{S}} = [ab; \mathbf{r}_a \otimes \mathbf{r}_b]_{\hat{\mathcal{S}}}$
2. Define $\mathbf{s} \stackrel{\text{def}}{=} \mathbf{r}_a \otimes \mathbf{r}_b$. Then the parties hold $[ab; \mathbf{s}]_{\hat{\mathcal{S}}}$. Let $[ab; \mathbf{s}]_{\hat{\mathcal{S}}} = (c_1, \ldots, c_{\hat{m}})$. For each P_i and each c_j he owns, P_i distributes $[c_j; \mathbf{r}_{c_j}]_{\mathcal{S}}$.
3. Let $\mathbf{u} = (u_1, \ldots, u_{\hat{m}})$ be the reconstruction vector defined above and guaranteed by the fact that $\mathcal{S}$ is multiplicative. The players compute

$$\sum_j u_j [c_j; \mathbf{r}_{c_j}]_{\mathcal{S}} = \left[\sum_j u_j c_j; \sum_j u_j \mathbf{r}_{c_j}\right]_{\mathcal{S}} = \left[ab; \sum_j u_j \mathbf{r}_{c_j}\right]_{\mathcal{S}} .$$

Output Reconstruction: At this point all gates, including the output gates have been processed. So do the following for each output gate (labeled i): The players hold $[y; \mathbf{r}_y]_{\mathcal{S}}$ where y is the value assigned to the output gate. Each P_j securely sends every share he holds of y to P_i, who can then compute $y = \omega_P \cdot [y; \mathbf{r}_y]_{\mathcal{S}}$, where ω_P is the reconstruction vector for the set of all players P.

Env$^{\texttt{sync,A}}$ denote environments that corrupt subsets in adversary structure $\mathcal{A}$. We do not give the proof in detail here. It is very easy to derive from the corresponding proof for $\pi^f_{\texttt{CEPS}}$, found in Example 4.10. As in that proof, the simulator runs the protocol with the environment, while running internally copies of the honest players using dummy inputs. Since the environment only sees shares from a set of players in $\mathcal{A}$, this makes no difference. In the final output step, the simulator patches the sets of shares of outputs known by the environment to match the real outputs. This is easy by adding an appropriate multiple of the vector $\mathbf{w}$ constructed in the proof of Theorem 6.6. Likewise, adaptive security follows because correct views for newly corrupted honest players can be constructed by patching the dummy views in the same way.

THEOREM 6.17 *$\pi^f_{\mathsf{GCEPS}} \diamond \mathsf{F}_{\mathsf{SC}}$ implements $\mathsf{F}^f_{\mathsf{SFE}}$ in $\mathrm{Env}^{sync,\mathcal{A}}$ with perfect security, whenever the protocol is based on a multiplicative linear secret sharing scheme $\mathcal{S}$ with adversary structure $\mathcal{A}$.*

It is easy to see that to have security against $\mathrm{Env}^{\mathrm{sync},\mathcal{A}}$ we do not actually need to have a linear scheme $\mathcal{S}$ whose adversary structure exactly equals $\mathcal{A}$, it is enough if it contains $\mathcal{A}$. We still have to demand, however, that the adversary structure of $\mathcal{S}$ be $Q2$, since otherwise we cannot compute all functions securely, by Theorem 6.2. This implies that $\mathcal{A}$ must be $Q2$ as well. Therefore by invoking also Theorem 6.14, we get:

COROLLARY 6.18 *Any function can be computed with perfect and passive security against an environment corrupting subsets in adversary structure $\mathcal{A}$, provided $\mathcal{A}$ is $Q2$. The complexity of the protocol can be made polynomial in the size of the smallest linear secret-sharing scheme whose adversary structure is $Q2$ and contains $\mathcal{A}$.*

6.5 Actively Secure Protocols

In this section we show how to get actively secure protocols from general linear secret-sharing. We do not give all details, as many of them are trivial to derive from the corresponding results and protocols in Chapter 5.

We will first assume that we are given the functionality $\mathsf{F}_{\mathsf{COM}}$ for homomorphic commitment, show how to evaluate functions securely from this and then consider how to implement commitments based on linear secret sharing.

We first define some notation. Let $\mathcal{S}$ be a linear secret sharing scheme, and recall that for an appropriately chosen vector $\mathbf{r}_a$ with a the first coordinate, $g(\mathbf{r}_a) = M_A \cdot \mathbf{r}_s$ is the vector of shares of a. Also recall that ϕ is the labeling function that tells us which players receive the individual shares. Then we define

$$[\![a;\mathbf{r}_a]\!]_{\mathcal{S}} = (\langle g(\mathbf{r}_a)_1\rangle_{\phi(1)}, \ldots, \langle g(\mathbf{r}_a)_m\rangle_{\phi(m)}) \ .$$

This is just the natural generalization of the concept we defined earlier: the secret a has been shared and players are committed to the shares they hold. We can then also define

$$[\![a;\mathbf{r}_a]\!]_{\mathcal{S}} + [\![b;\mathbf{r}_b]\!]_{\mathcal{S}},\ \alpha[\![a;\mathbf{r}_a]\!]_{\mathcal{S}},\ [\![a;\mathbf{r}_a]\!]_{\mathcal{S}} \odot [\![b;\mathbf{r}_b]\!]_{\mathcal{S}}$$

similarly to Definition 6.16, but in addition to computing on shares locally, the corresponding commands from $\mathsf{F}_{\mathsf{COM}}$ are also invoked. Then the following lemma is easy to show:

Protocol GCEAS (Generalized Circuit Evaluation, Active Security)

The protocol assumes a strongly multiplicative linear secret sharing scheme $\mathcal{S}$ with adversary structure $\mathcal{A}$ and proceeds in three phases: the input sharing, computation and output reconstruction phases.

Input Sharing: Each player P_i holding input $x_i \in \mathbb{F}$ distributes $[\![x_i; \mathbf{r}_{x_i}]\!]_{\mathcal{S}}$. If this fails, P_i is corrupt, so we will instead assign 0 as default input for P_i, by distributing $[\![0; \mathbf{r}_0]\!]_{\mathcal{S}}$ for a fixed default value of $\mathbf{r}_0$, say the all-0 vector. This can be done by a number of public commit to 0's.

We then go through the circuit and process the gates one by one in the computational order. Just after the input sharing phase, we consider all input gates as being processed. We will maintain the following:

Invariant: Consider an input or an output wire for any gate, and let $a \in \mathbb{F}$ be the value assigned to this wire. Then, if the gate has been processed, the players hold $[\![a; \mathbf{r}_a]\!]_{\mathcal{S}}$.

We then continue with the last two phases of the protocol:

Computation Phase: Repeat the following until all gates have been processed (then go to the next phase): Consider the first gate in the computational order that has not been processed yet. According to the type of gate, do one of the following

Addition gate: The players hold $[\![a; \mathbf{r}_a]\!]_{\mathcal{S}}, [\![b; \mathbf{r}_b]\!]_{\mathcal{S}}$ for the two inputs a, b to the gate. The players compute $[\![a+b; \mathbf{r}_a + \mathbf{r}_b]\!]_{\mathcal{S}} = [\![a; \mathbf{r}_a]\!]_{\mathcal{S}} + [\![b; \mathbf{r}_b]\!]_{\mathcal{S}}$.

Multiply-by-constant gate: The players hold $[\![a; \mathbf{r}_a]\!]_{\mathcal{S}}$ for the input a to the gate. The players compute $[\![\alpha a; \alpha \mathbf{r}_a]\!]_{\mathcal{S}} = \alpha [\![a; \mathbf{r}_a]\!]_{\mathcal{S}}$.

Multiplication gate: The players hold $[\![a; \mathbf{r}_a]\!]_{\mathcal{S}}, [\![b; \mathbf{r}_b]\!]_{\mathcal{S}}$ for the two inputs a, b to the gate.

1. The players compute $[\![ab; \mathbf{r}_a \otimes \mathbf{r}_b]\!]_{\hat{\mathcal{S}}} = [\![a; \mathbf{r}_a]\!]_{\mathcal{S}} \odot [\![b; \mathbf{r}_b]\!]_{\mathcal{S}}$. Note that this involves each P_j committing to his local products and proving that this was done correctly. So this may fail for some subset of players $A \in \mathcal{A}$.
2. Define $\mathbf{s} \overset{\text{def}}{=} \mathbf{r}_a \otimes \mathbf{r}_b$. Then the parties hold $[\![ab; \mathbf{s}]\!]_{\hat{\mathcal{S}}}$. Let $[\![ab; \mathbf{s}]\!]_{\hat{\mathcal{S}}} = (c_1, \dots, c_{\hat{m}})$. For each P_i and each c_j he owns, he distributes $[\![c_j; \mathbf{r}_{c_j}]\!]_{\mathcal{S}}$ from $\langle c_j \rangle_i$. Also this step may fail for a subset $A \in \mathcal{A}$.
3. Let S be the indices of at the players for which the previous two steps did not fail. By strong multiplicativity of $\mathcal{S}$, there exists $\mathbf{r}_S$, a recombination vector for S, that is, a vector $\mathbf{r}_S = (r_1, \dots, r_{\hat{m}})$ that players in S can use to reconstruct secrets generated by $\hat{\mathcal{S}}$. In particular, we have $ab = \sum_{\phi(j) \in S} r_j c_j$. The players compute
$$\sum_{\phi(j)\in S} r_j [\![c_j; \mathbf{r}_{c_j}]\!]_{\mathcal{S}} = [\![\sum_{\phi(j)\in S} r_j c_j; \sum_{\phi(j)\in S} r_j \mathbf{r}_{c_j}]\!]_{\mathcal{S}} = [\![ab; \sum_{\phi(j)\in S} r_j \mathbf{r}_{c_j}]\!]_{\mathcal{S}} .$$

Output Reconstruction: At this point all gates, including the output gates have been processed. So do the following for each output gate (labeled i): The players hold $[\![y; f_y]\!]_{\mathcal{S}}$ where y is the value assigned to the output gate. Now we open $[\![y; f_y]\!]_{\mathcal{S}}$ towards P_i.

LEMMA 6.19 *The following holds for any $a, b, \alpha \in \mathbb{F}$ and any multiplicative linear secret-sharing scheme $\mathcal{S}$:*

$$[\![a; \mathbf{r}_a]\!]_{\mathcal{S}} + [\![b; \mathbf{r}_b]\!]_{\mathcal{S}} = [\![a + b; \mathbf{r}_a + \mathbf{r}_b]\!]_{\mathcal{S}} \ , \tag{6.5}$$

$$\alpha[\![a; \mathbf{r}_a]\!]_{\mathcal{S}} = [\![\alpha a; \alpha\mathbf{r}_a]\!]_{\mathcal{S}} \ , \tag{6.6}$$

$$[\![a; \mathbf{r}_a]\!]_{\mathcal{S}} \odot [\![b; \mathbf{r}_b]\!]_{\mathcal{S}} = [\![ab; \mathbf{r}_a \otimes \mathbf{r}_b]\!]_{\hat{\mathcal{S}}} \ . \tag{6.7}$$

Similarly to what we saw in Chapter 5, when we say that P_i distributes $[\![a; \mathbf{r}_a]\!]_{\mathcal{S}}$, we mean that P_i commits to all entries in $\mathbf{r}_a$, we then use the `add`, `mult` commands to compute commitments to the shares in a and finally use `transfer` commands to ensure that the party holding a share is committed to that share.

It is now straightforward to show

THEOREM 6.20 *$\pi^f_{\mathsf{GCEAS}} \diamond \mathsf{F}_{\mathsf{SC}} \diamond \mathsf{F}_{\mathsf{COM}}$ implements $\mathsf{F}^f_{\mathsf{SFE}}$ in $\mathrm{Env}^{\mathcal{A},sync}$ with perfect security for any f, whenever the protocol is based on a strongly multiplicative linear secret sharing scheme $\mathcal{S}$ with adversary structure $\mathcal{A}$.*

We note that it is possible to modify Protocol GCEAS to make it work also for a $Q2$ adversary structure and a multiplicative scheme $\mathcal{S}$. As in Chapter 5, the difficulty in this case is that if a player fails during processing of a multiplication gate, we cannot proceed since all players are needed to reconstruct in $\hat{\mathcal{S}}$ because we do not assume strong multiplication. We can handle such failures by going back to the start of the computation and open the input values of the players that have just been disqualified. Now we restart the protocol, while the honest players simulate the disqualified players. Note that we can use default random choices for these players, so there is no communication needed for the honest players to agree on the actions of the simulated ones. This allows the adversary to slow down the protocol by a factor at most linear in n. In doing this, it is important that we do not have to restart the protocol after some party received its output — this is ensured by starting the Output Reconstruction phase only after all multiplication gates have been handled.

Let $\pi^f_{\mathtt{GCEAS\text{-}Q2}}$ be $\pi^f_{\mathtt{GCEAS}}$ modified to handle $Q2$-structures as we just discussed. We then have the following theorem, the proof of which we leave to the reader:

THEOREM 6.21 *$\pi^f_{\mathsf{GCEAS\text{-}Q2}} \diamond \mathsf{F}_{\mathsf{SC}} \diamond \mathsf{F}_{\mathsf{COM}}$ implements $\mathsf{F}^f_{\mathsf{SFE}}$ in $\mathrm{Env}^{\mathcal{A},sync}$ with perfect security for any f when based on a multiplicative secret sharing scheme with adversary structure $\mathcal{A}$.*

6.5.1 Implementation of Homomorphic Commitment from Linear Secret Sharing

Let us first consider how we can implement $\mathsf{F}_{\mathtt{COM}}$ from $\mathsf{F}_{\mathtt{COM\text{-}SIMPLE}}$. Note first that from the results in Chapter 5, we already have

THEOREM 6.22 *There exists a protocol $\pi_{\mathsf{TRANSFER,MULT}}$ such that $\pi_{\mathsf{TRANSFER,MULT}} \diamond \mathsf{F}_{\mathsf{SC}} \diamond \mathsf{F}_{\mathsf{COM\text{-}SIMPLE}}$ implements $\mathsf{F}_{\mathsf{COM}}$ in $\mathrm{Env}^{t,sync}$ with statistical security for all $t < n$.*

Protocol GENERALIZED PERFECT COMMITMENT MULTIPLICATION

This protocol is based on a strongly multiplicative secret sharing scheme $\mathcal{S}$. If any command used during this protocol fails, all players output `fail`.

1. $\langle c \rangle_i \leftarrow ab$.
2. P_i chooses vectors $\mathbf{r}_a, \mathbf{r}_b$ with coordinates $a, \alpha_1, \dots, \alpha_t$ and $b, \beta_1, \dots, \beta_t$, for random α_j, β_j. He computes $\mathbf{r}_c = \mathbf{r}_a \otimes \mathbf{r}_b$, let $c, \gamma_1, \dots, \gamma_{(t+1)^2} - 1$ be the coordinates of $\mathbf{r}_c$. Then establish commitments as follows:
$$\langle \alpha_j \rangle_i \leftarrow \alpha_j \text{ for } j = 1, \dots, t$$
$$\langle \beta_j \rangle_i \leftarrow \beta_j \text{ for } j = 1, \dots, t$$
$$\langle \gamma_j \rangle_i \leftarrow \gamma_j \text{ for } j = 1, \dots, (t+1)^2 - 1$$
3. Recall that the matrices of $\mathcal{S}, \hat{\mathcal{S}}$ are called $M, \hat{M}$. Then, for $k = 1, \dots, m$ the `add` and `mult` commands are invoked to compute:
$$\langle a_k \rangle_i = M_{k,0} \langle a \rangle_i + M_{k,1} \langle \alpha_1 \rangle_i + \cdots + M_{k,t} \langle \alpha_t \rangle_i$$
$$\langle b_k \rangle_i = M_{k,0} \langle b \rangle_i + M_{k,1} \langle \beta_1 \rangle_i + \cdots + M_{k,t} \langle \beta_t \rangle_i$$
and for $k = 1, \dots, \hat{m}$ we compute:
$$\langle c_k \rangle_i = \hat{M}_{k,0} \langle b \rangle_i + \hat{M}_{k,1} \langle \gamma_1 \rangle_i + \ldots + \hat{M}_{k,(t+1)^2-1} \langle \gamma_{2t} \rangle_i \ .$$
Note that by this computation we have that a_k is the k'th share in a, i.e., $a_k = \mathbf{m}_k \cdot \mathbf{r}_a$, similarly b_k, c_k are the k'th shares in b and c, respectively. Then for all k, we execute $(\mathsf{P}_{\phi(k)}) a_k \leftarrow \langle a_k \rangle_i$, $(\mathsf{P}_{\phi(k)}) b_k \leftarrow \langle b_k \rangle_i$ and $(\mathsf{P}_{\phi(k)}) c_k \leftarrow \langle c_k \rangle_i$.
4. For $k = 1, \dots, \hat{m}$, do: $\mathsf{P}_{\phi(k)}$ checks that $a_u b_v = c_k$ for the appropriate u, v with $\phi(u) = \phi(v) = \phi(k)$ (see Lemma 6.7). He broadcasts `accept` if this is the case, else he broadcasts (`reject`, k).
5. For each $\mathsf{P}_{\phi(k)}$ who said (`reject`, k), do $a_u \leftarrow \langle a_u \rangle_i$, $b_v \leftarrow \langle b_v \rangle_i$ and $c_k \leftarrow \langle c_k \rangle_i$, all players check whether $a_u b_v = c_k$ holds. If this is not the case, all players output `fail`.
6. If we arrive here, no command failed during the protocol and all relations $a_u b_v = c_k$ that were checked, hold. All players output `success`.

This is just a restatement of part of Theorem 5.1. The result can be used in this context because $\pi_{\texttt{TRANSFER,MULT}}$ only uses commands from $\mathsf{F}_{\texttt{COM-SIMPLE}}$ and no secret-sharing, and because tolerating any number of corruptions less than n is of course good enough to handle any non-trivial adversary structure. Furthermore, we also have

THEOREM 6.23 *There exists a protocol $\pi_{GPTRANSFER,GPMULT}$ such that $\pi_{GPTRANSFER,GPMULT} \diamond F_{SC} \diamond F_{COM\text{-}SIMPLE}$ implements F_{COM} in* $\mathrm{Env}^{\mathcal{A},sync}$ *with perfect security when based on a strongly multiplicative secret sharing scheme $\mathcal{S}$ with adversary structure $\mathcal{A}$.*

The idea for constructing $\pi_{\texttt{GPTRANSFER,GPMULT}}$ is to start from $\pi_{\texttt{PTRANSFER,PMULT}}$, as presented in Section 5.2, replace secret-sharing based on polynomials by $\mathcal{S}$ and otherwise do “the same”. We show here how the multiplication proof can be done in this way. We argue informally why Protocol GENERALIZED PERFECT COM-

MITMENT MULTIPLICATION is secure and leave a formal simulation proof to the reader:

If the prover P_i is honest, the adversary gets no information on a, b: some values a_k, b_k, c_k are revealed to corrupt players, but the corrupted set is in the adversary structure so the a_k, b_k give no information and c_k always equals $a_u b_v$ fro the relevant values of k, u, v. Furthermore, all honest players will say `accept` after the check, so only values the adversary already knows are broadcast. If P_i is corrupt, note that the computation on commitments ensures that the sets $\{a_k\} = [a, \mathbf{r}_a]_{\mathcal{S}}, \{b_k\} = [b, \mathbf{r}_b]_{\mathcal{S}}, \{c_k\} = [c, \mathbf{r}_c]_{\hat{\mathcal{S}}}$ are consistent sets of shares computed under $\mathcal{S}, \mathcal{S}$ and $\hat{\mathcal{S}}$, respectively, and they therefore determine well defined secrets a, b, c. Furthermore the set of shares $[a, \mathbf{r}_a]_{\mathcal{S}} \odot [b, \mathbf{r}_b]_{\mathcal{S}}$ is a valid set of secrets under $\hat{\mathcal{S}}$ and determine the secret ab. By the checks done we know that if `success` is output the sets of shares $[a, \mathbf{r}_a]_{\mathcal{S}} \odot [b, \mathbf{r}_b]_{\mathcal{S}}$ and $[c, \mathbf{r}_c]_{\hat{\mathcal{S}}}$ agree in all entries owned by honest players. But the set of honest players is the complement of a set in the adversary structure of $\mathcal{S}$, it is therefore in the access structure of $\hat{\mathcal{S}}$ by the strong multiplication property, hence it is large enough to determine a secret in $\hat{\mathcal{S}}$ uniquely. It follows that $c = ab$ as desired.

EXERCISE 6.6 Construct the rest of protocol $\pi_{\texttt{GPTRANSFER,GPMULT}}$, that is, construct a perfectly secure protocol for transferring a commitment from one player to another based on a linear secret sharing scheme $\mathcal{S}$ and use this to prove Theorem 6.23.

We now turn to the question of how to implement $\mathsf{F}_{\texttt{COM-SIMPLE}}$ from linear secret sharing. It is of course not surprising that the solution is similar to what we have seen in Chapter 5, but some issues need to be addressed.

Note first that if players hold $[s, \mathbf{r}_s]_{\mathcal{S}}$, the adversary structure $\mathcal{A}$ is $Q3$, and players broadcast their shares, then the correct value of s is uniquely determined: we can in principle simply search for some $s', \mathbf{r}_{s'}$ such that $[s', \mathbf{r}_{s'}]_{\mathcal{S}}$ agrees with the set of shares broadcast except in some set of entries owned by some unqualified set $A' \in \mathcal{A}$. We claim that then s' always equals the correct secret s. The set A' may not be the set of players that are actually corrupted, but this does not matter: if we start from the set of all shares and then take away entries owned by corrupted players (say in set A) and also take away all entries in A', then $[s', \mathbf{r}_{s'}]_{\mathcal{S}}$ agrees with $[s, \mathbf{r}_s]_{\mathcal{S}}$ in the remaining entries since these all correspond to honest players. This set of entries is in the access structure of $\mathcal{S}$ by the $Q3$ property so $s = s'$ as desired. On the other hand, the search will always stop because $(s', \mathbf{r}_{s'}) = (s, \mathbf{r}_s)$ is of course a good solution. Note that if this is used as a commitment scheme, players would not have to do the search (this would be inefficient in most cases), we could just ask the committer to point out the right solution.

To make a perfect homomorphic commitment scheme, it is therefore sufficient to force a committer to hand out consistent shares of the value he has in mind, i.e., we want something of form $[a; \mathbf{r}_a]_{\mathcal{S}}$. This ensures privacy against corruption of any unqualified set, and if we require the committer to reveal $\mathbf{r}_a$ at opening time it follows from what we just saw that he cannot change to a different value of a.

To enforce consistency, we reuse the approach from earlier and have the committer secret share his value but then also secret-share the randomness he used for this. This introduces the possibility of other players cross-checking what they have, ultimately enforcing consistency. We can phrase this as a redundant version of the original linear secret sharing scheme, similar to the bivariate polynomials we saw earlier.

Therefore, to commit to a value $a \in \mathbb{F}$, based on secret sharing scheme $\mathcal{S}$, P_i chooses a random symmetric $(t+1) \times (t+1)$ matrix R_a with a in the upper left corner. Note that by symmetry, we have $R_a = R_a^\mathsf{T}$. Then, for each index k, he sends $\mathbf{u}_k = R_a \cdot \mathbf{m}_k^\mathsf{T}$ to $\mathsf{P}_{\phi(k)}$ (recall that $\mathbf{m}_k$ is a row vector taken from the matrix of $\mathcal{S}$).

Note that the product of the first row of R_a with $\mathbf{m}_k^\mathsf{T}$ is a normal share of a according to $\mathcal{S}$, so the share β_k of a is defined to be the first entry in $\mathbf{u}_k$. Note that for any pair of indices k, j we have (since R_a is symmetric):

$$\mathbf{m}_j \cdot \mathbf{u}_k = \mathbf{m}_j \cdot R_a \cdot \mathbf{m}_k^\mathsf{T} = (R_a \cdot \mathbf{m}_j^\mathsf{T})^\mathsf{T} \cdot \mathbf{m}_k^\mathsf{T} = \mathbf{m}_k \cdot \mathbf{u}_j$$

so if we define $\beta_{k,j} = \mathbf{m}_j \cdot \mathbf{u}_k = \mathbf{m}_k \cdot \mathbf{u}_j$, this is a value that both $\mathsf{P}_{\phi(k)}$ and $\mathsf{P}_{\phi(j)}$ can compute, so this can be used as a consistency check on the information P_i distributes. We have

Lemma 6.24 *If honest player P_i chooses R_a and sends $\mathbf{u}_k = R_a \cdot \mathbf{m}_k^\mathsf{T}$ to $\mathsf{P}_{\phi(k)}$ then the information given to any unqualified set A has distribution independent of a.*

Proof The information given to the set A can be written as $M_A \cdot R_a$. Since A is unqualified, there exists a vector $\mathbf{w}$ that has first coordinate 1 and is orthogonal to all rows in M_A. Now, if we interpret the tensor product $\mathbf{w} \otimes \mathbf{w}$ as a matrix in the natural way, this matrix is symmetric and has 1 in the upper left corner. Therefore, if we define $R_0 = R_a - a \cdot (\mathbf{w} \otimes \mathbf{w})$, then R_0 is symmetric and has 0 in the upper left corner. Furthermore $M_A \cdot R_a = M_A \cdot R_0$. So for any set of shares that A might receive, the number of random choices that could lead to these shares is the same for any value of a, namely the number of possible choices for $a = 0$. The lemma follows. □

Now, to show the security of Protocol Generalized Commit, we need two main lemmas:

Lemma 6.25 *If P_i remains honest throughout Protocol* Generalized Commit*, the view of any set of players $A \in \mathcal{A}$ players is independent of the committed value a, and all players who are honest at the end of the protocol will output the shares in a that P_i distributed.*

This follows immediately from Lemma 6.24 and the fact that if P_i remains honest, then the values that are broadcast are only values that players in A already know from the first step.

Protocol Generalized Commit

1. On input $(\texttt{commit}, i, cid, a)$, P_i chooses symmetric matrix R_a with a in the top left corner and sends $\mathbf{u}_k = R_a \cdot \mathbf{m}_k^\mathsf{T}$ to $\mathsf{P}_{\phi(k)}$. $\mathsf{P}_{\phi(k)}$ sets β_k to be the first entry in $\mathbf{u}_k$.
2. For each j, $\mathsf{P}_{\phi(j)}$ computes $\beta_{k,j} = \mathbf{m}_k \cdot \mathbf{u}_j$ and sends $\beta_{k,j}$ to $\mathsf{P}_{\phi(k)}$.
3. Each $\mathsf{P}_{\phi(k)}$ checks that $\beta_{k,j} = \mathbf{m}_j \cdot \mathbf{u}_k$. If so, it broadcasts `success`. Otherwise, it broadcasts $(\texttt{dispute}, k, j)$ for each inconsistency.
4. For each dispute reported in the previous step, P_i broadcasts the correct value of $\beta_{k,j}$.
5. If any $\mathsf{P}_{\phi(k)}$ finds a disagreement between what P_i has broadcast and what he received privately from P_i, he knows P_i is corrupt and broadcasts $(\texttt{accuse}, k)$.
6. For any accusation from $\mathsf{P}_{\phi(k)}$ in the previous step, P_i broadcasts all vectors he sent to $\mathsf{P}_{\phi(k)}$.
7. If any P_k finds a new disagreement between what P_i has now broadcast and what he received privately from P_i, he knows P_i is corrupt and broadcasts $(\texttt{accuse}, k)$.
8. If the information broadcast by P_i is not consistent, or if a set of players not in the adversary structure has accused P_i, players output `fail`.
   Otherwise, players who accused P_i and had new vectors broadcast will accept these. All others keep the vectors they received in the first step. Now, for each j, $\mathsf{P}_{\phi(j)}$ stores (cid, i, β_j). In addition P_i stores the first row of R_a. All players output `success`.

Lemma 6.26 *If the adversary structure is $Q3$, then no matter how corrupt players behave in Protocol* Generalized Commit, *players who are honest at the end of the protocol will all output* `fail` *or will output a set of shares in some value a' all consistent with a vector $\mathbf{r}_{a'}$.*

Proof The decision to output fail is based on public information, so it is clear that players who remain honest will either all output `fail` or all output some value, so assume the latter case occurs. If we take the set of all players and subtract the set A' of players who accused P_i and also subtract the set of players who are corrupt at the end of the protocol, then since A, A' are unqualified and the adversary structure is $Q3$, the remaining set S must be qualified. Note that S consists of honest players who did not accuse P_i. Let H be the set of all honest players.

We make the following *claim*: Consider any j with $\mathsf{P}_{\phi(j)} \in H$ and any k with $\mathsf{P}_{\phi(k)} \in S$. Then the two players agree on the $\beta_{k,j}$ they have in common, that is, $\mathbf{m}_k \cdot \mathbf{u}_j = \mathbf{m}_j \cdot \mathbf{u}_k$.

We argue this as follows: if $\mathsf{P}_{\phi(j)}$ had a new vector broadcast for him in Step 6, then the claim is true, since otherwise $\mathsf{P}_{\phi(k)}$ would have accused in Step 7. On the other hand, suppose no new values were broadcast for $\mathsf{P}_{\phi(j)}$. Then he keeps the values he was given in Step 1, as does $\mathsf{P}_{\phi(k)}$. It also means that neither $\mathsf{P}_{\phi(j)}$ nor $\mathsf{P}_{\phi(k)}$ accused at any time. But our claim holds for the values sent in Step 1, for if not, a $(\texttt{dispute}, k, j)$ would have been reported, and either $\mathsf{P}_{\phi(j)}$ or $\mathsf{P}_{\phi(k)}$ would have accused P_i in Step 5, and this did not happen.

Now form two matrices U_H, U_S where the columns consist of all $\mathbf{u}_k$-vectors held by players in H, respectively S. If M is the matrix of the secret sharing scheme $\mathcal{S}$, then $M_S \cdot U_S$ is a matrix containing all $\beta_{k,j}$-values that players in S can compute.

The claim above on agreement between players in S and all players in H can be compactly written as $M_H \cdot U_S = (M_S \cdot U_H)^\intercal$.

Let $\mathbf{v}$ be the reconstruction vector for S that we know exists by Theorem 6.6. We claim that the vector $\mathbf{r}_{s'} = U_S \cdot \mathbf{v}$ defines the sharing that P_i has effectively distributed. More precisely, as s' we can take the first entry in $\mathbf{r}_{s'}$ and furthermore, if we build a vector containing the shares output by players in H, this is indeed formed correctly from $\mathbf{r}_{s'}$, i.e., it equals $M_H \cdot \mathbf{r}_{s'}$. To show this, we can use the above claim to compute as follows:

$$M_H \cdot \mathbf{r}_{s'} = M_H \cdot U_S \cdot \mathbf{v} = (M_S \cdot U_H)^\intercal \cdot \mathbf{v} = U_H^\intercal \cdot M_S^\intercal \cdot \mathbf{v} = U_H^\intercal \cdot \mathbf{e}$$

where the last step follows from the property the reconstruction vector has (as can be seen in the proof of Theorem 6.6), and $\mathbf{e}$ is the first canonical basis vector. Clearly the entries in the vector $U_H^\intercal \cdot \mathbf{e}$ are the first entries in all the $\mathbf{u}_j$-vectors held by players in H, and these entries are exactly what they output as their shares by specification of the protocol. □

Based on what we have seen now it is straightforward to construct a complete protocol $\pi_{\texttt{GEN-PERFECT-COM-SIMPLE}}$ for implementing $\mathsf{F}_{\texttt{COM-SIMPLE}}$ based on a secret sharing scheme $\mathcal{S}$. This, as well as the simulation proof of security, follows the same recipe as we used for the proof of Theorem 5.6. We therefore have

THEOREM 6.27 *$\pi_{\texttt{GEN-PERFECT-COM-SIMPLE}} \diamond \mathsf{F}_{\texttt{SC}}$ implements $\mathsf{F}_{\texttt{COM-SIMPLE}}$ in $\mathrm{Env}^{\mathcal{A},sync}$ with perfect security when based on a linear secret sharing scheme $\mathcal{S}$ with $Q3$ adversary structure $\mathcal{A}$.*

EXERCISE 6.7 Construct the rest of protocol $\pi_{\texttt{GEN-PERFECT-COM-SIMPLE}}$ and show Theorem 6.27.

If we only go for statistical security in the end, and assume we have a broadcast functionality given, then assuming access to the $\mathsf{F}_{\texttt{ICSIG}}$ functionality from Section 5.4.1, we can implement $\mathsf{F}_{\texttt{COM-SIMPLE}}$ even for $Q2$ adversary structures. The resulting protocol, $\pi_{\texttt{GEN-COM-SIMPLE}}$ can be constructed in exactly the same way as we did $\pi_{\texttt{COM-SIMPLE}}$ for the threshold case earlier: to commit we do the same as in Protocol GENERALIZED COMMIT, except that the committer must sign everything he sends using $\mathsf{F}_{\texttt{ICSIG}}$. All accusations against the committer can now be proved because the accuser can demonstrate that he indeed received a particular message from the committer. This leads to:

THEOREM 6.28 *$\pi_{\texttt{GEN-COM-SIMPLE}} \diamond \pi_{\texttt{ICSIG}} \diamond \mathsf{F}_{\texttt{SC}}$ implements $\mathsf{F}_{\texttt{COM-SIMPLE}}$ in $\mathrm{Env}^{\mathcal{A},sync}$ with statistical security when based on a linear secret sharing scheme $\mathcal{S}$ with $Q2$ adversary structure $\mathcal{A}$.*

6.5.2 General Results on MPC from Linear Secret Sharing

If we combine everything we have seen so far, we obtain a number of results for general adversary structures that are natural generalizations of the results for threshold adversaries.

We first note that broadcast can be implemented securely from secure point-to-point channels, if the actively corrupted set of players is from a $Q3$ adversary structure [94]:

Theorem 6.29 *There exists a protocol* $\pi_{\texttt{GEN-BROADCAST}}$ *such that* $\pi_{\texttt{GEN-BROADCAST}} \diamond \mathsf{F}_{\texttt{PPC}}$ *implements* $\mathsf{F}_{\texttt{SC}}$ *in* $\mathrm{Env}^{\mathcal{A},sync}$ *with perfect security when* $\mathcal{A}$ *is* $Q3$*.*

Then, Theorems 6.23, 6.20, 6.27 and 6.29 imply that any function can be computed with perfect, active security against $Q3$ adversary structures, given access to secure point-to-point channels. More formally, we have:

Theorem 6.30 $\pi^f_{\texttt{GCEAS}} \diamond \pi_{\texttt{GPTRANSFER,GPMULT}} \diamond \pi_{\texttt{GEN-PERFECT-COM-SIMPLE}} \diamond \pi_{\texttt{GEN-BROADCAST}} \diamond \mathsf{F}_{\texttt{PPC}}$ *implements* $\mathsf{F}^f_{\texttt{SFE}}$ *in* $\mathrm{Env}^{\mathcal{A},sync}$ *with perfect security when based on a strongly multiplicative linear secret sharing scheme* $\mathcal{S}$ *with* $Q3$ *adversary structure* $\mathcal{A}$*.*

The final result in this chapter concerns the case where we assume access to $\mathsf{F}_{\texttt{SC}}$, i.e., we assume secure channels and broadcast and go for statistical security. In this case, putting previous results together, we get that Theorems 6.22, 6.21 and 6.28 imply that any function can be computed with statistical, active security if less than $n/2$ players are corrupt, given access to secure point-to-point channels and broadcast. More formally, we have:

Theorem 6.31 $\pi^f_{\texttt{GCEAS-Q2}} \diamond \pi_{\texttt{TRANSFER,MULT}} \diamond \pi_{\texttt{ICSIG}} \diamond \pi_{\texttt{GEN-COM-SIMPLE}} \diamond \mathsf{F}_{\texttt{SC}}$ *implements* $\mathsf{F}^f_{\texttt{SFE}}$ *in* $\mathrm{Env}^{\mathcal{A},sync}$ *with statistical security when based on a multiplicative linear secret sharing scheme* $\mathcal{S}$ *with* $Q2$ *adversary structure* $\mathcal{A}$*.*

These results show what can be done for active security with a given secret sharing scheme. We might also ask: what is the best protocol we can have for a given adversary structure $\mathcal{A}$?

One can first note that in general, it is clearly sufficient for security to come up with a secret sharing scheme $\mathcal{S}$ whose adversary structure *contains* $\mathcal{A}$. The protocol built from $\mathcal{S}$ would then protect against an even stronger adversary.

Now, for perfect security, we know that $\mathcal{A}$ must be $Q3$ to allow any function to be computed with perfect active security, and finally any $Q3$ structure admits a strongly multiplicative secret sharing scheme. Therefore we have:

Corollary 6.32 *Any function can be computed with perfect active security against an environment corrupting subsets in adversary structure* $\mathcal{A}$*, provided* $\mathcal{A}$ *is* $Q3$*, and we have access to* $\mathsf{F}_{\texttt{PPC}}$*. The complexity of the protocol can be made polynomial in the size of the smallest strongly multiplicative linear secret-sharing scheme whose adversary structure contains* $\mathcal{A}$*.*

For the case of statistical security, it is sufficient to have any linear scheme with a $Q2$-adversary structure containing $\mathcal{A}$, by Theorem 6.14. So for this case we get:

Corollary 6.33 *Any function can be computed with statistical active security against an environment corrupting subsets in adversary structure $\mathcal{A}$, provided $\mathcal{A}$ is $Q2$, and we have access to F_{SC}. The complexity of the protocol can be made polynomial in the size of the smallest linear secret-sharing scheme whose adversary structure contains $\mathcal{A}$.*

One can also note in this last corollary, that if $\mathcal{A}$ happens to be $Q3$ then we can replace access to F_{SC} by access to $\mathsf{F}_{\mathsf{PPC}}$ and simulate broadcast using Theorem 6.29. This gives a statistical version of the first corollary, and this can be significant for efficiency because the smallest linear scheme for an adversary structure may (for all we know) be much smaller than the best strongly multiplicative one.

6.6 Notes

Multiparty Computation with security against general adversary structures was first considered by Hirt and Maurer [110], who showed that the basic completeness theorems for unconditional security generalize from $t < n/2, t < n/3$ to $Q2$ and $Q3$ adversary structures, respectively.

In 2000, Cramer, Damgård and Maurer [67] showed that passive and active multiparty computation follows from multiplicative and strongly multiplicative linear secret sharing schemes, respectively, where the complexity of the protocol is polynomial in the size of the secret sharing scheme.

One may wonder if this result is the most general one possible, i.e., can one base unconditionally secure multiparty computation on *any* secret-sharing scheme? An answer to this was given by Cramer, Damgård and Dziembowski [65] who showed that, although verifiable secret sharing does follow by an efficient black-box reduction from any secret sharing scheme, no such reduction exists for multiparty computation, at least not one that would be efficient for any access structure. Therefore some extra property (such as linearity) must be assumed for a general reduction to exist.

The results on linear secret sharing in this chapter are from [67], except the efficient construction of secret sharing for the dual access structure which is by Serge Fehr (manuscript available from the author's web page). The model of linear secret sharing that we use in this chapter is essentially that of monotone span programs which were introduced by Karchmer and Wigderson [120]. We do not, however, introduce the full formalism of span programs, this is more relevant if one wants to see span programs as a model of computation, which is not our focus here.

7

Cryptographic MPC Protocols

Contents

7.1 Introduction

The main focus of this book is information theoretic solutions, and therefore we only cover the case of computational security very superficially. Basically, we survey some of the main results and give pointers for further reading.

7.2 The Case of Honest Majority

A main completeness result for computationally secure multiparty computation is that any function can be computed with computational security in a model where authenticated point to point channels are given. The adversary may corrupt $t < n/2$ players actively. Recall that computational security is defined just like statistical security, except that the simulator is only required to work for polynomial time bounded environments. More formally, the result is as follows:

THEOREM 7.1 *If non-committing encryption schemes exist, then there exists a protocol* π^f_{COMPSEC} *such that* $\pi^f_{\mathsf{COMPSEC}} \diamond \mathsf{F}_{\mathsf{AT}}$ *implements* $\mathsf{F}^f_{\mathsf{SFE}}$ *in* $\mathrm{Env}^{t,\mathtt{sync},\ \mathtt{poly}}$ *with computational security for all* $t < n/2$.

We will explain what non-committing encryption is in a moment, see also Section 4.3. For now we note that the assumption that $t < n/2$ is necessary to guarantee that the protocol terminates and gives output to honest players. In the 2-party case, for instance, it is clear that we cannot terminate if one of the players simply stops playing. We discuss below what can be done if we do not assume honest majority.

Note that we have already seen *statistically* secure protocols for the case $t < n/2$, but there we had to assume access to secure channels and broadcast (F_{SC}), so the above result says that we can make do with only authenticated channels (F_{AT}) if we settle for computational security.

The reader might complain about the fact that we assume authenticated channels – if we are willing to assume computationally secure cryptography, we can both public-key encrypt and authenticate messages with signatures, so can we not get a protocol that starts from nothing? This, however, is not possible, at least in most practical scenarios. One issue is that even in a two-player protocol, an honest player would need to make sure he is talking to the player he wants to do the protocol with and not a third party, otherwise the protocol would not be secure against an external attacker in the case where both players are honest, and this is a property one would often need. We can, however, minimize the use of authenticated channels, namely by using signatures. This will mean that we only need authenticated channels in an initial phase to communicate public keys.

To show Theorem 7.1, one notes that from Theorem 5.13 we already know that we can implement $\mathsf{F}_{\mathsf{SFE}}^{f}$ with statistical security if we are given access to F_{SC}. It will therefore be sufficient to implement F_{SC} given access to F_{AT}, i.e., implement secure communication based on authenticated communication. Signatures plus authenticated channels to communicate public keys is enough to get the broadcast we need to implement for F_{SC}. The only remaining thing is to also provide confidentiality. This can be done using encryption – recall that this is a running example in Chapter 4. But to get adaptive security one needs a solution that is secure even if both players are corrupted after some communication has happened. One can do this using so-called non-committing encryption, where the simulator is able to construct "ciphertexts" that are not the result of properly encrypting something, but can later be claimed to contain any plaintext the simulator desires. In this way, the simulator can handle the case where it must simulate a conversation between two honest players, and where one or both of these players is later corrupted. Non-committing encryption from one-way trapdoor permutations with certain extra properties was constructed in [40], with efficiency improvements under specific assumptions in [78]. Informally speaking, what one needs is that one can generate a permutation (a public key) obliviously, that is, while learning nothing about the corresponding trapdoor (secret key). This is also sometimes known as a simulatable trapdoor permutations (STP).

Even the best non-committing encryption schemes known are very inefficient compared to regular public-key encryption, the size of a ciphertext is a factor κ larger than the plaintext, where κ is the security parameter. However, if we are only after static security, any chosen-ciphertext secure (CCA-secure) cryptosystem can be used instead, and this will lead to a much more efficient protocol with current state of the art.

Note also that this approach gives a completely different proof of Theorem 7.1 than the first one given in [103], which only showed static security, assuming existence of one-way trapdoor permutations. The approach used there was to build a protocol for oblivious transfer (OT) from trapdoor permutations, and then base the rest of the protocol on OT and zero-knowledge proofs.

OT is a primitive that one may think of as an ideal functionality that gets two input bits b_0, b_1 from a sender and one input bit c from a receiver, and then returns b_c to the receiver. In fact, given only OT as a black box, one can evaluate any function securely for any number of players, this was shown in [122].

It is instructive to note the difference in efficiency between the two approaches we have outlined and where it comes from. The first approach combines a statistically secure protocol $\pi_{\texttt{STAT}}$ based on secret sharing with a computationally secure implementation of F_{SC}. Since $\pi_{\texttt{STAT}}$ only requires that we do linear algebra in relatively a small field, it can be implemented very efficienctly, in fact, the complexity can be close to linear in the security parameter. The implementation of F_{SC} does require public-key cryptography which is much less efficient, but since we only need the cryptography as a secure transport mechanism, we can combine the public-key systems with much more efficient symmetric cryptosystems, at least if we settle for static security. Such an implementation of F_{SC} corresponds closely to the kind of service offered on the Internet by the well known (and very efficient) SSL and TLS protocols. In contrast, the approach from [103] uses public-key techniques throughout to do the actual computation and is therefore less efficient when used directly. The reader should note, however, that with a more recent technique for implementing the OTs needed in [103], the difference in efficiency becomes much smaller. This technique is known as OT-extension [114, 145], and the idea is do a small number of OTs using expensive public-key cryptography and then extend these to a much larger number of OTs by cheaper techniques based on special types of hash functions.

However, regardless of the approach used, if we want adaptive security and resort to non-committing encryption, we will get an impractical protocol. It is therefore interesting to note that we can have adaptively secure solutions that are more efficient that if we are given public-key cryptosystems with stronger properties. For instance, some cryptosystems are additively homomorphic, that is, the plaintexts come from some ring, and given ciphertexts c_1, c_2 that are encryptions of m_1, m_2, and the public key (but not the secret key) one can easily compute c_3 that looks exactly like a fresh encryption of $m_1 + m_2$. It was shown in [64] that we can compute arithmetic circuits over the plaintext ring securely if the secret key is secret-shared among the players. This only gives static security, but in [79] it was shown that if we base the protocol on Paillier's public-key cryptosystem [149], adaptive security can be obtained at cost only a constant factor over the static solution.

Even stronger properties are offered by fully homomorphic encryption, where we can also compute a new ciphertext that contains the product of the plaintexts. The first such schemes were constructed in [102]. With such a scheme multiparty computation is easy in principle: players publish ciphertext (under a single public key) containing their input, everyone can now evaluate a ciphertext containing the desired output, and if we assume that the secret key has been secret shared among the players, then they can collaborate to decrypt the result. This solution is extremely efficient in terms of communication, in particular, we need only a constant number of rounds, and the amount of communication is independent of the circuit we compute. However, with current state of the art, the computational overhead is very large. At the time of writing, it is still unclear whether fully homomorphic encryption will be of real practical importance.

7.3 The Case of Dishonest Majority

When t is not assumed to be less than $n/2$, we have already seen that we cannot have information theoretic security at all, and we also cannot guarantee that the protocol terminates (unless we only go for passive security). This means that the best we can hope for is a computationally secure solution of a weaker version of $\mathsf{F}_{\mathtt{SFE}}^{f}$ where the environment gets the output first and can then tell the functionality whether the honest players should get the result or not. Let $\mathsf{F}_{\mathtt{SFE}}^{f,\mathtt{abort}}$ be this weaker variant.

Even if we only go for a UC-secure implementation of $\mathsf{F}_{\mathtt{SFE}}^{f,\mathtt{abort}}$, this turns out to be impossible is most cases, as shown in [41]. Basically, to do anything non-trivial in the UC model for dishonest majority, one needs a set-up assumption, some ideal functionality that is assumed available but is not implemented. Popular such assumptions are the common reference string model (CRS), where we assume a functionality that outputs a string to all players with a certain distribution, and the key-registration model [4] where it is assumed that parties have public keys and are guarenteed to know their secret keys. Of course, one can argue that this makes the model less realistic, but on the other hand, all results up to now have assumed at least access to $\mathsf{F}_{\mathtt{AT}}$, and this is also a type of set-up assumption, even if it is often not considered as such in the literature.

The basic reason why simulation arguments do not always go through in the plain UC model is that the simulator needs to be able to do "something" that an adversary could not do in a real attack. For example suppose a protocol instructs a player P_i to send a ciphertext and prove to another player P_j that he knows the plaintext, of course without revealing it. A simulator for this procedure is typically required to simulate what a corrupt P_j would see in this game. The simulation must be done without knowing the plaintext, this is exactly how we argue that the plaintext stays hidden from P_j. But on the other hand, assume that instead P_i is corrupt and sends a ciphertext where he does not know the plaintext. Now, why can he not do what the simulator does, and in this way cheat P_j? In the so-called stand-alone model where we do not consider composition of protocols, the answer usually is that the simulator is allowed to rewind its "opponent" (the adversary) to a previous state, and rewinding the other player is of course not something one could do in a real protocol execution.

However, rewinding does not work in the UC model: to have security under arbitrary concurrent composition, we cannot allow the simulator to rewind the adversary (the environment in this case). Therefore, we need to be able to give the simulator an edge in some other way. In the CRS model, the edge is that the simulator gets to choose the reference string – in fact it must do so since it has to simulate everything the environment sees in the protocol. The point is that it may be able to generate the CRS together with some trapdoor relating to the

CRS that a real life adversary does not get to see, and this is why the simulator can fake a convincing view of the protocol, whereas an adversary cannot do so.

Using such a set-up assumption, we can indeed do general multiparty computation also with dishonest majority. In [42] the following was shown:

THEOREM 7.2 *Assuming STPs exist, there exists a protocol* $\pi^f_{\textsf{COMPSEC-DM}}$ *in the CRS model, such that* $\pi^f_{\textsf{COMPSEC-DM}} \diamond \mathsf{F}_{\mathsf{AT}}$ *implements* $\mathsf{F}^{f,abort}_{\mathsf{SFE}}$ *in* $\mathrm{Env}^{t,sync,poly}$ *with computational security for all* $t \leq n$.

Based on the fact that information theoretic security is not possible for dishonest majority, one might think that information theoretic methods for protocol design are not useful in this setting. This is very far from true, however, and we will have more to say about this in Chapter 8.

8

Some Techniques for Efficiency Improvements

Contents

8.1 Introduction

In this chapter we cover some techniques for improving the efficiency of the protocols we have seen earlier in the book, but some of the techniques also apply to secure computing protocols in general.

8.2 Circuit Randomization

Recall the way we represented secret data when we constructed the first MPC protocol for passive security: for $a \in \mathbb{F}$, we defined the object $[a; f_a]_t$ to be the set of shares $f_a(1), \ldots, f_a(n)$ where $f_a(0) = a$ and the degree of f_a is at most t. At the same time it was understood that player P_i holds $f_a(i)$.

One of the most important properties of this way to represent data is that it is linear, that is, given representations of values a and b players can compute a representation of $a + b$ by only local computation, this is what we denoted by $[a; f_a]_t + [b; f_b]_t = [a + b; f_a + f_b]_t$ which for this particular representation means that each P_i locally computes $f_a(i) + f_b(i)$.

Of course, this linearity property is not only satisfied by this representation. The representation $[\![a; f_a]\!]_t$ we defined, based on homomorphic commitments to shares, is also linear in this sense.

A final example can be derived from additively homomorphic encryption: if we represent $a \in \mathbb{F}$ by $E_{pk}(a)$ where pk is a public key, and the corresponding secret key is shared among the players, then the additive homomorphic property exactly ensures that players can add ciphertexts and obtain an encryption of the sum, i.e., it holds that $E_{pk}(a) + E_{pk}(b) = E_{pk}(a + b)$, where the addition of ciphertexts is an operation that can be computed efficiently given only the public key.

In the following, for simplicity we will use the notation $[a]$ to denote any representation that is linear in the sense we just discussed. In doing so, we suppress the randomness that is usually used to form the parts held by the players (such as the polynomial f_a in $[a; f_a]_t$).

We assume some main properties of a linear representation, that we only define informally here. For any of the examples we mentioned, it is easy to see what they concretely mean is each of the cases.

Definition 8.1 *A linear representation $[\cdot]$ over a finite field $\mathbb{F}$ satisfies the following properties:*

1. *Any player P_i can collaborate with the other players to create $[r]$, where $r \in \mathbb{F}$ is chosen by P_i. If P_i is honest, the process reveals no information on r. $[r]$ will be correctly formed no matter whether P_i is honest or not.*
2. *If players hold representations $[a], [b]$ and a public constant α, they can locally compute new representations $[a] + [b] = [a + b]$ and $\alpha[a] = [\alpha a]$ of the same form as $[a]$ and $[b]$.*
3. *Given $[a], [b]$, the players can interact to compute a new representation $[a][b] = [ab]$ of the same form as $[a]$ and $[b]$ while revealing nothing new about a, b.*
4. *The players can collaborate to open any representation, in particular also representations of form $[a + b]$ or $[ab]$. This will reveal $a + b$, respectively ab but will reveal no other information on a, b.*

Following the pattern of several protocols we have already seen, it is clear how one could do secure function evaluation with a linear representation, namely players first create representations of their inputs, then they work their way through a circuit representing the desired function using the procedures for addition and multiplication, and finally we open the representations of the outputs.

There is, however, a way to push a lot of the work needed into a preprocessing phase, and this technique is known as circuit randomization. The idea is to preprocess a number of secure multiplications on random values and then later use these to multiply more efficiently the actual values occurring in the desired computation. This not only leads to a more efficient on-line phase, but can also lead to better efficiency overall, as it is often more efficient to generate a lot of multiplications on random values in parallel, than to perform the multiplications on real data as they are needed.

More concretely, we first create a number of so-called multiplication triples. Such a triple has form $[a], [b], [c]$ where a, b are uniformly random, unknown to all players and $c = ab$. One way to create such triples is to have each player P_i create $[a_i], [b_i]$ for random a_i, b_i, and the use the assumed operations to compute

$$[a] = [a_1] + \cdots + [a_n], \quad [b] = [b_1] + \cdots + [b_n] \text{ and finally } [c] = [ab] \ .$$

We note already now that this is not the most efficient method, we look at better solutions below. But in any case, it is clear that whatever protocol we use should create many triples in parallel. This opens the possibility of using techniques specialized for this purpose where the amortized cost per triple can be minimized, moreover this can usually be done in a constant number of rounds and hence be quite efficient.

We can then revisit the MPC protocol we sketched before: In a preprocessing phase, we would create a large number of multiplication triples. Once we know the function to compute and the inputs are fixed, we execute the on-line phase. Here we have players create representations of their inputs and then any linear computation on represented values can be done by only local operations. But for multiplication, we would make use of the preprocessing:

Assume we are given $[a], [b]$ and want to compute $[ab]$ securely. To do this, we take the next unused triple $[x], [y], [z]$ from the preprocessing. We then compute

$$[a] - [x] = [a - x], \ [b] - [y] = [b - y]$$

and open $e = a - x, d = b - y$. Note that because we assume x, y to be uniformly random in the view of the environment, this is secure to do: e, d will be uniformly random and hence easy to simulate. Moreover, it is easy to see that we have

$$ab = xy + eb + da - ed = z + eb + da - ed \ ,$$

and hence players can compute locally

$$[ab] = [z] + e[b] + d[a] - ed \ .$$

This will reduce the work we need in the on-line phase to opening two representations per secure multiplication plus some local computation, whereas doing $[x][y]$ directly would typically require more interaction. Since communication is usually more costly in practice than local computation, this gives a significant performance improvement for many concrete representations. Moreover, for the case of dishonest majority, where we know that we need to use public-key cryptography in some form, it turns out to be possible to push the use of such expensive cryptography into the preprocessing phase and use a more efficient representation in the on-line phase. We return to this below.

Protocol On-Line Phase summarizes the protocol we have just sketched. It is straightforward to formalize it, by first specifying the representation as an ideal functionality with operations for creation, arithmetic and opening. One can then show that the protocol implements $\mathsf{F}_{\mathsf{SFE}}^{f}$ when the circuit used computes f.

8.3 Hyper-invertible Matrices

A hyper-invertible matrix M over a finite field $\mathbb{F}$ is a matrix with the property that any square submatrix is invertible. More precisely,

Definition 8.2 *A matrix M is hyper-invertible if the following holds: Let R be a subset of the rows, and let M_R denote the sub-matrix of M consisting of rows in R. Likewise, let C be a subset of columns and let M^C denote the sub-matrix*

Protocol On-Line Phase

This protocol computes an arithmetic circuit securely over a field $\mathbb{F}$. It assumes a linear representation of values in the field, written $[a]$, with properties as described in the text. We assume also a preprocessing phase that creates a number of random multiplication triples of form $[a], [b], [c]$ with $ab = c$. We assume for simplicity that all outputs are public.

Input Sharing: Each player P_i holding input $x_i \in \mathbb{F}$ creates a representation $[x_i]$.

We then go through the circuit and process the gates one by one in the computational order. Just after the input sharing phase, we consider all input gates as being processed. We will maintain the following:

Invariant: Computing with the circuit on inputs $x_1, \ldots, x_n$ assigns a unique value to every wire. Consider an input or an output wire for any gate, and let $a \in \mathbb{F}$ be the value assigned to this wire. Then, if the gate has been processed, the players hold $[a]$.

Computation Phase: Repeat the following until all gates have been processed (then go to the next phase): Consider the first gate in the computational order that has not been processed yet. According to the type of gate, do one of the following

Addition gate: The players hold $[a], [b]$ for the two inputs a, b to the gate. The players compute $[a] + [b] = [a + b]$.

Multiply-by-constant gate: The players hold $[a]$ for the input a to the gate, and public constant α. The players compute $\alpha[a] = [\alpha a]$.

Multiplication gate: The players hold $[a], [b]$ for the two inputs a, b to the gate. Take the next unused triple $[x], [y], [z]$ from the preprocessing phase.

1. Players compute $[a] - [x] = [a - x]$, $[b] - [y] = [b - y]$, and open $e = a - x, d = b - y$.
2. Players compute locally $[ab] = [z] + e[b] + d[a] - ed$.

Output Reconstruction: At this point all gates, including the output gates have been processed. So for each output gate, the players hold $[y]$ where y is the value assigned to the output gate. We open $[y]$ to reveal y to all players.

consisting of columns in C. Then we require that M_R^C is invertible whenever $|R| = |C| > 0$.

The term *hyper-invertible* was coined in [11] in the context of MPC. Hyper-invertible matrices are known in several other places than MPC, and are also known as regular matrices and MDS matrices, and are equivalent to linear interpolation codes and MDS codes. See Chapter 11 for more on interpolation codes and MDS codes.

Hyper-invertible matrices have nice properties that make them very useful in MPC. We will look at this shortly, first we show that they actually exist, at least if the underlying field is large enough. Some notation that will be useful below: when C is a set of indices designating some of the entries in $\mathbf{x}$, we let $\mathbf{x}_C$ denote the vector containing only the coordinates designated by C.

THEOREM 8.3 *Let $\alpha_1, \dots, \alpha_n, \beta_1, \dots, \beta_m \in \mathbb{F}$ be arbitrary but distinct (so we assume $|\mathbb{F}| \geq n+m$). Consider the function that maps $(x_1, \dots, x_n)$ to $(y_1, \dots, y_m)$ as follows: first compute the unique polynomial f of degree at most $n-1$ such that $f(\alpha_i) = x_i, i = 1, \dots, n$. Then output $f(\beta_1), \dots, f(\beta_m)$. This is clearly a linear mapping. The matrix M of this mapping is hyper-invertible.*

PROOF First consider any input $\mathbf{x} = (x_1, \dots, x_n)$ and corresponding output $\mathbf{y} = (y_1, \dots, y_m)$. Then, given a total of n of the x_i's and y_i's, we can compute all the other x_i's and y_i's. This just follows from the fact that knowing the value f takes when evaluated on n distinct inputs is enough to compute f by Lagrange interpolation, and then we evaluate f in the other points to get the remaining x_i's and y_i's.

Now, assume we are given sets R, C designating rows and columns with $|C| = |R|$ and let $\bar{C}$ be the complement of C. Then clearly $|\bar{C}| + |R| = n$. Therefore it follows from what we just observed that given $\mathbf{x}_{\bar{C}}, \mathbf{y}_R$, we can compute $\mathbf{x}_C$.

We now use this to show that M_R^C is invertible. We do so by demonstrating how $\mathbf{x}_C$ can be computed from $\mathbf{y}_R = M_R^C \mathbf{x}_C$. Suppose we are given just $\mathbf{y}_R$. Then we set $\mathbf{x}_{\bar{C}}$ to be the all-0 vector. The above observation says we can now compute full vectors $\mathbf{x}, \mathbf{y}$ with $\mathbf{y} = M\mathbf{x}$. But this and $\mathbf{x}_{\bar{C}} = 0$ implies that in particular we have found $\mathbf{x}_R$ such that $\mathbf{y}_R = M_R^C \mathbf{x}_C$, as $\mathbf{y} = M\mathbf{x} = M^C\mathbf{x}_C + M^{\bar{C}}\mathbf{x}_{\bar{C}} = M^C\mathbf{x}_C$.
□

We now state two useful properties of hyper-invertible matrices:

LEMMA 8.4 *Let M be hyper-invertible, and let C be a subset of the columns, R a subset of the rows, with $|\bar{C}| = |R|$, that is, $|C| + |R| = n$. Consider any $\mathbf{x}, \mathbf{y}$ with $\mathbf{y} = M\mathbf{x}$. Then given $\mathbf{x}_C, \mathbf{y}_R$, one can efficiently compute $\mathbf{x}_{\bar{C}}, \mathbf{y}_{\bar{R}}$. Furthermore, the map $\mathbf{x}_C, \mathbf{y}_R \mapsto \mathbf{x}_{\bar{C}}, \mathbf{y}_{\bar{R}}$ is linear.*

PROOF It is easy to see that

$$\mathbf{y}_R = M_R\mathbf{x} = M_R^C\mathbf{x}_C + M_R^{\bar{C}}\mathbf{x}_{\bar{C}} \ .$$

Since M is hyper-invertible, $M_R^{\bar{C}}$ is invertible, so rearranging the above, we have

$$\mathbf{x}_{\bar{C}} = (M_R^{\bar{C}})^{-1}(\mathbf{y}_R - M_R^C\mathbf{x}_C) \ .$$

Now we have all of $\mathbf{x}$, so $\mathbf{y}$ is trivial to compute. It is clear that the map is linear.
□

LEMMA 8.5 *Let M be hyper-invertible, and consider the linear mapping $\phi_M : \mathbb{F}^n \to \mathbb{F}^m$ induced by M. Suppose we fix k of the input coordinates to arbitrary values. Then consider the linear mapping ϕ' naturally induced by ϕ_M from the remaining $n-k$ coordinates to any $n-k$ coordinates of the output (so we assume that $m \geq n-k$). Then ϕ' is a bijection.*

PROOF The $n-k$ input coordinates correspond naturally to a subset C of the columns of M, and the selection of output coordinates naturally to a subset R

of the rows. Given an input vector $\mathbf{z} \in \mathbb{F}^{n-k}$ for ϕ', we can construct a vector $\mathbf{x} \in \mathbb{F}^n$ such that $\mathbf{x}_C = \mathbf{z}$ and $\mathbf{x}_{\bar{C}}$ contains the k fixed coordinates. Then

$$\phi'(\mathbf{z}) = M_R\mathbf{x} = M_R^C\mathbf{x}_C + M_R^{\bar{C}}\mathbf{x}_{\bar{C}} \ .$$

Since $M_R^{\bar{C}}\mathbf{x}_{\bar{C}}$ is a fixed vector and M_R^C is invertible, this is clearly a bijection. □

Randomness Extraction and Efficient Secure Multiplication

We now look at how the above properties of hyper-invertible matrices can be used in multiparty computation. In particular we have seen the concept of a linear representation of secret data, and we have seen that, for instance, to create multiplication triples it is very useful to be able to create a representation of a random field element that is unknown to the environment. We also saw a trivial way to do this where each player creates a representation of his own random element and then we add all contributions. This works, but seems wasteful because each of the $n-t$ honest player contributes a random element but we end up with only 1 random representation.

Hyper-invertible matrices offer a better solution as shown in Protocol Extract Random Representations. The protocol uses an n by n hyper-invertible matrix M and creates first a set of representations $[x_i],\ i = 1, \dots, n$. Now note that since we can do linear computation on representations, we can consider these as a vector with entries $[x_i],\ i = 1, \dots, n$. Now multiply M on this vector, and obtain in a natural way a vector of representations $[y_i],\ i = 1, \dots, n$. More precisely $[y_i] = M_{i,1}[x_1] + \cdots + M_{i,n}[x_n]$. Note that, by our assumptions on linear representations, the $[y_i]$ can be computed by only local computation.

Protocol Extract Random Representations

This protocol produces a number of random representations based on contributions from each player.

1. Each player P_i creates $[x_i]$ where x_i is uniformly random.
2. The players apply matrix M to this vector $[x_1], \dots, [x_n]$ of representations (as explained in the text), to get via local computation representations $[y_1], \dots, [y_n]$.
3. Output the representations $[y_1], \dots, [y_{n-t}]$.

We claim that the y_i's contained in the output $[y_1], \dots, [y_{n-t}]$ are uniformly random in the view of the environment.

To see this, note that first if we let $\mathbf{x} \in \mathbb{F}^n$ be the vector with the x_i's as entries, and define a vector $\mathbf{y}$ from the y_i's in a similar way, then $\mathbf{y} = M\mathbf{x}$. Now, let C be the set of indices designating the $n-t$ honest players, and let R be the set of indices $1, \dots, n-t$. Now, by Lemma 8.5, for any set of values in $\mathbf{x}_{\bar{C}}$ chosen by the corrupt players, the values taken by $y_1, \dots, y_{n-t}$ are in 1-1 correspondence with the entries in $\mathbf{x}_C$. But these are uniformly random and independent of the view of the environment, since they were chosen by the honest players, and hence so are $y_1, \dots, y_{n-t}$.

In this way, the amortized cost of outputting a representation of a random value is, up to a constant factor, only that of having a single player create a representation, and hence a factor n more efficient than the naïve method.

It is instructive to consider that this idea can also be seen as follows: we are given $x_1, \ldots, x_n$, and we know that $n - t$ of these values are random, but not which ones. Now, using a hyper-invertible matrix, we can compute $n - t$ new values that are guaranteed to be random, so in other words, we can extract all the randomness that is available.

Exercise 8.1 In this exercise we assume passive corruption and we use representations of form $[a; f_a]_t$, where $t < n/2$.

Using Protocol Extract Random Representations as a starting point, construct a protocol that will produce as output a number of pairs $[r; f_r]_t, [r, g_r]_{2t}$, where r is random and unknown to all players and f_r, g_r are independent and randomly chosen polynomials.

Show that the total communication done in your protocol is $O(n)$ field elements per pair that is output.

The result of the exercise can be used to construct an efficient protocol that starts from $[a; f_a]_t, [b; f_b]_t$ and outputs $[ab, f_{ab}]_t$, still assuming passive (adaptive) corruption.

To see this, recall that players can compute using local computation $[ab; f_a f_b]_{2t}$. Now, if we assume we have a pair $[r; f_r]_t, [r, g_r]_{2t}$ available, then by further local computation, players can compute $[ab - r; g_r - f_a f_b]_{2t}$. This representation we can safely open because r and g_r are random, and so all players now know $ab - r$. We can then make a default representation of $ab - r$ using the polynomial that is constantly $ab - r : [ab - r; ab - r]_t$. Adding this to $[r, f_r]_t$ will produce $[ab, f_{ab}]_t$, where $f_{ab} = f_r + ab - r$.

An important detail about this protocol is that, to open $[ab - r; g_r - f_a f_b]_{2t}$, there is no reason to have all players send their share to all other players. It is sufficient to send to just one player, say P_1. It can then reconstruct $ab - r$ and send this value to all players. Note that this works since we assume passive corruption, so P_1 will follow the protocol, and that this only requires sending $O(n)$ field elements.

Together with the exercise, we see that we now have a protocol for doing many secure multiplications that costs communication $O(n)$ field elements per multiplication. In comparison, the first protocol we saw in Chapter 3 required $O(n^2)$ field elements.

Exercise 8.2 The efficient opening described above can be made active secure at almost the same price. We will only sketch how this is done. The parties start with t secret sharings which need to be opened, call the secret-shared values $(v_1, \ldots, v_t)$. Then they apply a linear error correcting code C over $\mathbb{F}$, producing secret sharings of the entries in the codeword $(w_1, \ldots, w_n) = C(v_1, \ldots, v_t)$, which is possible by just locally applying C to the shares of all values, as C is linear.

Then each $[w_i]$ is opened efficiently as above, with P_i playing the role of P_1. The corrupted P_i's might lie about their w_i, but the honest P_i's will distribute the correct w_i's. This ensures that $n - t$ of the values w_i received by all parties are correct. If $n < 3t$, then a linear interpolation code C (see Chapter 11) will allow to recover the t original values even in the presence of the t errors, see Section 11.8.2. Work out the exact details of this.

Checking consistency of secret sharings and other linear properties

In the first part of this subsection, we concentrate on linear representations based on secret sharing. The simplest form we have seen is $[a; f_a]_t$ where player P_j holds $f_a(j)$ and $f_a(0) = a$. Having player P_i create such a representation is simple if we settle for passive security: P_i just selects f_a and sends $f_a(j)$ to each P_j.

This does not work for active security, as we have no guarantee that what P_i sends is consistent with a polynomial of degree at most t. We have seen in Chapter 5 a protocol that can be used to force even a corrupt P_i to select consistent shares (Protocol Commit). This protocol has perfect security if $t < n/3$, but requires a lot of interaction and communicates much more data than the simple solution where P_i just sends one value to all players.

We now show in Protocol Check Consistent Secret-Shares that hyper-invertible matrices can be used to verify whether P_i has behaved correctly and this method will be much more efficient if we want to create many random shared values. We will assume for simplicity that $n = 3t + 1$.

In the following, we will say that a representation $[a; f_a]_t$ is well-formed if the values held by honest players are consistent with a polynomial of degree at most t. Note that we abuse notation slightly by writing $[a; f_a]_t$ with subscript t even if there is no guarantee that the degree of f_a is at most t until this has been checked.

Protocol Check Consistent Secret-Shares

This protocol checks that a set of shares of values distributed by player P_i are well-formed.

1. P_i distributes $[x_1; f_{x_1}]_t, \ldots, [x_n; f_{x_n}]_t$ for random $x_1, \ldots, x_n$.
2. The players apply matrix M to this vector of representations (in the same way as in the previous subsection), to get via local computation representations $[y_1; f_{y_1}]_t, \ldots, [y_n; f_{y_n}]_t$.
3. For $j = 1, \ldots, 2t$, all players send their share in $[x_j; f_{x_j}]_t$ and $[y_j; f_{y_j}]_t$ to P_j. Furthermore, for $j = 2t + 1..n$, all players send their share in $[x_j; f_{x_j}]_t$ to P_j. Player P_j checks that all sets of shares he receives are consistent with a polynomial of degree at most t, and broadcasts `accept` or `reject` accordingly.
4. If all players said `accept`, then output $[y_{2t+1}; f_{y_{2t+1}}]_t, \ldots, [y_n; f_{y_n}]_t$.

The claim is now that first, if all players say `accept` then $[y_{2t+1}; f_{y_{2t+1}}]_t, \ldots, [y_n; f_{y_n}]_t$ are well-formed even if P_i is corrupt, i.e., the degree of f_{x_j} for $j = 2t + 1, \ldots, n$

is at most t. And second, that if P_i is honest, then $y_{2t+1}, \ldots, y_n$ are uniformly random in the view of the environment.

To show the first claim, notice that if all players say `accept`, then of the $2n$ representations $[x_1; f_{x_1}]_t, \ldots, [x_n; f_{x_n}]_t, [y_1; f_{y_1}]_t, \ldots, [y_n; f_{y_n}]_t$, we know that at least n of them are well-formed, namely the $2t+1$ $[x_j; f_{x_j}]_t$'s that were checked by honest players and the t $[y_j; f_{y_j}]_t$'s that were checked by honest players. Then it follows from Lemma 8.4 that all the other representations can be computed from these n well-formed ones. In particular $[y_{2t+1}; f_{y_{2t+1}}]_t, \ldots, [y_n; f_{y_n}]_t$ can be computed as a linear combination of well-formed representations, and so it immediately follows that they are well-formed as well. This is because a linear combination of polynomials of degree at most t is again of degree at most t.

To show the second claim, note that the environment knows t of the values $x_1, \ldots, x_n,$, namely those for which the corresponding representations were "checked" by a corrupt player. It now follows from Lemma 8.5 that the remaining $n-t$ x_j's (which are unknown to the environment) are in 1-1 correspondence with any subset of $n-t$ of the y_j's. We form such a subset as the union of $y_{2t+1}, \ldots, y_n$ and t y_j's that were checked by honest players (at least t such y_j's must be available). All these y_j's are uniformly random in the view of the environment, so in particular this is true for $y_{2t+1}, \ldots, y_n$.

Note that if Protocol Check Consistent Secret-Shares has success, then it outputs a constant fraction of the representations that were created initially. Hence, up to a constant fraction, the cost of the protocol is the same as simply sending one share to each player per representation [1].

It is of course a natural question what we do if Protocol Check Consistent Secret-Shares is not successful? The protocol as it stands would just not generate any output. However, there are more productive ways of dealing with such a situation. The basic observation is that if some P_j says `reject`, then either P_i or P_j must be corrupt. One option is now to simply eliminate both players from the global protocol. Note that if we had less than one third corrupted players before the elimination, this is also the case after, so we can still hope to finish the computation eventually. Also note that such a problem can at most occur t times, so this upper bounds the extra work we do as a result of eliminations. See the notes below for pointers to literature on this issue.

It is not hard to see that the principle from Protocol Check Consistent Secret-Shares can be used to check other properties of values inside representations, as long as these properties are preserved under linear transformations. As an example, suppose we use representations of form $[a; f_a]_t$ and we want to check that a player has created a number of representations $[0; f_1]_t, \ldots, [0, f_n]_t$. This can be done with a simple variation of Protocol Check Consistent Secret-Shares, where players who verify representations should also verify that the value in the checked representation is 0. This works, exactly because the property of containing a 0 is preserved under linear mappings on representations.

[1] Here, we conveniently ignore the broadcasts towards the end of the protocol, and these will usually not come for free. However, if we created enough representations in parallel and only do one set of broadcasts, then the amortized cost of these will become insignificant.

EXERCISE 8.3 Specify the protocol sketched above and prove that if $n = 3t + 1$, then your protocol can output t random sharings of 0. Specify and prove a protocol for producing a number of pairs of representations of form $([a; f_a]_t, [a, g_a]_{2t})$, i.e., the same element is contained in both representations.

8.4 Packed Secret-Sharing.

In this section we present a technique that allows us to secret-share several field elements in one go, in such a way that each player only receives one field element as his share. We must pay for this by being able to tolerate a smaller number of corrupted players, but the idea can nevertheless be used in combination with other techniques to get better efficiency in many cases.

The idea is a simple twist on Shamir secret sharing. We will assume for simplicity of notation that the underlying field is $\mathbb{F} = \mathbb{Z}_p$ for a prime p, where $p > \ell + n$ and n is the number of players. This means that the numbers $-\ell+1, \dots, 0, 1, \dots, n$ can be interpreted as distinct field elements in a natural way. To share a vector of elements $\mathbf{s} = (s_1, \dots, s_\ell) \in \mathbb{F}^\ell$, tolerating up to t corrupted players, one selects a random polynomial $f_{\mathbf{s}}(\mathtt{X})$ of degree at most $d = \ell - 1 + t$ with the property that $f_{\mathbf{s}}(-j + 1) = s_j$ for $j = 1, \dots, \ell$. Then the share given to P_i is $f_{\mathbf{s}}(i)$. We can now easily prove:

LEMMA 8.6 *Let the secret vector* $\mathbf{s}$ *and random polynomial* $f_{\mathbf{s}}$ *be defined as above. Then, any subset of at most* t *shares has a distribution independent of* $\mathbf{s}$ *and from any set of at least* $\ell + t$ *shares, one can reconstruct* $\mathbf{s}$.

PROOF The last statement on reconstruction follows trivially from Lagrange interpolation. As for the first statement, assume without loss of generality that we consider the shares $f_{\mathbf{s}}(1), \dots, f_{\mathbf{s}}(t)$. Note that by Lagrange interpolation, we can construct a polynomial h of degree at most $d = \ell - 1 + t$ such that $h(1) = \dots = h(t) = 0$ and $h(-j + 1) = -s_j$ for $j = 1, \dots, \ell$. Hence for each polynomial $f_{\mathbf{s}}(\mathtt{X})$ for sharing $\mathbf{s}$, there exists exactly one polynomial, namely $f_{\mathbf{s}}(\mathtt{X}) + h(\mathtt{X})$ for sharing the vector $(0, .., 0)$, generating the same first t shares. Since polynomials are chosen uniformly, it follows that for every secret vector, the distribution of the t shares is the same, namely the one resulting from sharing the all-zero vector. □

We now show how to use packed secret-sharing to get passively secure computation on vectors, while only having players operate on single field elements. We will need to assume that the degree d of the polynomials is such that $2d < n$. We will also assume that t and ℓ are both in $\Theta(n)$, this is clearly possible while still ensuring the demand on d is satisfied.

We can first define a linear representation of vectors with ℓ coordinates, denoted

$$[\mathbf{a}; f_{\mathbf{a}}]_d = (f_{\mathbf{a}}(1), \dots, f_{\mathbf{a}}(n)) \ .$$

And if we define addition of representations by coordinate-wise addition as usual, one easily sees that we have

$$[\mathbf{a}; f_{\mathbf{a}}]_d + [\mathbf{b}; f_{\mathbf{b}}]_d = [\mathbf{a} + \mathbf{b}; f_{\mathbf{a}} + f_{\mathbf{b}}]_d \ .$$

In other words, if the n players hold representations of two vectors, we can add them securely by each player doing just one local addition.

In the same way, we define $[\mathbf{a}; f_{\mathbf{a}}]_d * [\mathbf{b}; f_{\mathbf{b}}]_d$ as the coordinate-wise product of the two sets of shares, and we have

$$[\mathbf{a}; f_{\mathbf{a}}]_d * [\mathbf{b}; f_{\mathbf{b}}]_d = [\mathbf{a} * \mathbf{b}; f_{\mathbf{a}} f_{\mathbf{b}}]_{2d} \ .$$

This means that if $2d < n$, we can do a multiplication protocol in much the same way as we have seen before, this can be seen in Protocol Packed Secure Multiplication.

The protocol assumes that auxiliary representations $[\mathbf{r}; f_{\mathbf{r}}]_{2d}, [\mathbf{r}; g_{\mathbf{r}}]_d$ for a random vector $\mathbf{r}$ are available. Such pairs can be constructed using the techniques we have seen in the previous section (see exercise below). Note that it is secure to send all shares of $\mathbf{a} * \mathbf{b} - \mathbf{r}$ to P_1 because $\mathbf{r}$ and $f_{\mathbf{r}}$ are random.

Protocol Packed Secure Multiplication

This protocol takes as input two representations $[\mathbf{a}; f_{\mathbf{a}}]_d$, $[\mathbf{b}; f_{\mathbf{b}}]_d$.
It is assumed that we have available a pair of auxiliary representations $[\mathbf{r}; f_{\mathbf{r}}]_{2d}, [\mathbf{r}; g_{\mathbf{r}}]_d$ for a random vector $\mathbf{r}$,

1. $[\mathbf{a}; f_{\mathbf{a}}]_d * [\mathbf{b}; f_{\mathbf{b}}]_d = [\mathbf{a} * \mathbf{b}; f_{\mathbf{a}} f_{\mathbf{b}}]_{2d}$ by local multiplication.
2. The difference $[\mathbf{a} * \mathbf{b}; f_{\mathbf{a}} f_{\mathbf{b}}]_{2d} - [\mathbf{r}; f_{\mathbf{r}}]_{2d}$ is computed by local computation and all shares are sent to player P_1.
3. P_1 reconstructs $\mathbf{a} * \mathbf{b} - \mathbf{r}$ from the shares received, then forms $[\mathbf{a} * \mathbf{b} - \mathbf{r}; h]_d$ and sends a share to each player.
4. Players compute by local operations $[\mathbf{a} * \mathbf{b} - \mathbf{r}; h]_d + [\mathbf{r}; g_{\mathbf{r}}]_d = [\mathbf{a} * \mathbf{b}; h + g_{\mathbf{r}}]$.

Exercise 8.4 Using the randomness extraction techniques from the previous section, build a protocol that outputs a set of pairs of form $[\mathbf{r}; f_{\mathbf{r}}]_{2d}, [\mathbf{r}; g_{\mathbf{r}}]_d$ for random $\mathbf{r}$ as required in the multiplication above. Assuming t and ℓ are $\Theta(n)$, show that your protocol uses communication of $O(n)$ field elements per pair produced.

Let us consider what these ideas actually achieve: we have seen that if $2d = 2(t + \ell - 1) < n$ then we can do secure addition and multiplication of ℓ-vectors using the same communication (and computational work) that we used in the previous subsection to do addition and multiplication on *single* field elements, namely we send $O(n)$ field elements per operation. So we get ℓ operations "for the price of one". Of course, this only works in a so-called "same instruction, multiple data" fashion (SIMD), i.e., we always have to do the *same* operation on all ℓ entries in a block. Moreover, the SIMD operations do not come for free, we have to accept a smaller threshold value for the number of corruptions we can tolerate, but we can still tolerate corruption of a constant fraction of the players.

An obvious application of this is to compute securely ℓ instances in parallel of the same arithmetic circuit C. Since we chose ℓ to be $\Theta(n)$, the communication we need per gate computed is a constant number of field elements.

It is natural to ask if we can use this approach to save on computation and work also when computing a *single* circuit securely, and whether we can get active security as well? The answer turns out to be yes in both cases, and this leads to protocols that are close to optimal, at least in some respects. The notes section below has more information on this.

8.5 Information Theoretic Protocols in the Preprocessing Model

In this section we show how some of the techniques we have seen can be used for the case of *dishonest majority*, i.e., we assume up to t active corruptions and $t \leq n-1$. We have already seen in Chapter 7 that for this case, we have to use public-key machinery which is typically quite expensive. In comparison, the information theoretically secure primitives we have seen are computationally simpler and more efficient.

It may therefore seem that we cannot have really efficient solutions for dishonest majority. This is not entirely true, however: what we can do is to assume a preprocessing phase (using public-key cryptography) where the function to compute and the inputs are not yet known. This phase will produce some "raw material" to be used in an on-line phase where the actual computation is done and where we can use the more efficient primitives. More concretely, we will use the pattern we saw in the section on circuit randomization, so the idea will be to define an appropriate linear representation and assume a preprocessing that outputs multiplication triples in this representation.

To define our linear representation, we will use again a technique we saw in the implementation of $\mathsf{F}_{\texttt{ICSIG}}$, namely to use information theoretic message authentication codes (MACs). This will lead to statistical security, and in general the protocols we will see have error probability $1/|\mathbb{F}|$. We assume for simplicity that the field is so large that this is negligible, but there are also solutions that work for small fields (see the Notes section below for details on this).

A key K for our MACs is a random pair $K = (\alpha, \beta) \in \mathbb{F}^2$, and the authentication code for a value $a \in \mathbb{F}$ is $\mathrm{MAC}_K(a) = \alpha a + \beta$.

We recall briefly how these MACs are to be used: One party P_i will hold $a, \mathrm{MAC}_K(a)$ and another party P_j holds K. The idea is to use the MAC to prevent P_i from lying about a when he is supposed to reveal it to P_j. It will be very important in the following that if we keep α constant over several different MAC keys, then one can add two MACs and get a valid authentication code for the sum of the two corresponding messages. More concretely, two keys $K = (\alpha, \beta), K' = (\alpha', \beta')$ are said to be *consistent* if $\alpha = \alpha'$. For consistent keys, we define $K + K' = (\alpha, \beta + \beta')$ so that it holds that $\mathrm{MAC}_K(a) + \mathrm{MAC}_{K'}(a') = \mathrm{MAC}_{K+K'}(a + a')$.

In the protocol below, we will give to P_i several different values $m_1, m_2, \ldots$ with corresponding MACs $\gamma_1, \gamma_2, \ldots$ computed using keys $K_i = (\alpha, \beta_i)$ that are random but consistent. It is then easy to see that if P_i claims a false value for any of the m_i's (or a linear combination of them) he can guess an acceptable MAC for such a value with probability at most $1/|\mathbb{F}|$. This follows by an argument similar to what we saw in the proof of Theorem 5.7.

To represent a value $a \in \mathbb{F}$, we will give a share a_i to each party P_i, where the a_i are randomly chosen, subject to $a = \sum_i a_i$. In addition, P_i will hold MAC keys $K^i_{a_1}, \ldots, K^i_{a_n}$. He will use key $K^i_{a_j}$ to check the share of P_j, if we decide to make a public. Finally, P_i also holds a set of authentication codes $\mathrm{MAC}_{K^j_{a_i}}(a_i)$. We will denote $\mathrm{MAC}_{K^j_{a_i}}(a_i)$ by $m_j(a_i)$ from now on. Party P_i will use $m_j(a_i)$ to convince P_j that a_i is correct, if we decide to make a public. Summing up, we have the following way of representing a:

$$[a] = (a_i, \{K^i_{a_j}, m_j(a_i)\}^n_{j=1})^n_{i=1}$$

where $a_i, \{K^i_{a_j}, m_j(a_i)\}^n_{j=1}$ is the information held privately by P_i.

We say that $[a] = (a_i, \{K^i_{a_j}, m_j(a_i)\}^n_{j=1})^n_{i=1}$ is *consistent*, with $a = \sum_i a_i$, if $m_j(a_i) = \mathrm{MAC}_{K^j_{a_i}}(a_i)$ for all i, j. Two representations

$$[a] = (a_i, \{K^i_{a_j}, m_j(a_i)\}^n_{j=1})^n_{i=1}, \quad [a'] = (a'_i, \{K^i_{a'_j}, m_j(a'_i)\}^n_{j=1})^n_{i=1}$$

are said to be *key-consistent* if they are both consistent, and if for all i, j the keys $K^i_{a_j}, K^i_{a'_j}$ are consistent. We will want *all* representations in the following to be key-consistent: this is ensured by letting P_i use the same α_j-value in keys towards P_j throughout. Therefore the notation $K^i_{a_j} = (\alpha^i_j, \beta^i_{a_j})$ makes sense and we can do linear computations with these representations, as detailed in Protocol Operations on $[\cdot]$-representations.

Protocol Operations on $[\cdot]$-representations

Opening: We can reliably open a consistent representation to P_j: each P_i sends $a_i, m_j(a_i)$ to P_j. P_j checks that $m_j(a_i) = \mathrm{MAC}_{K^j_{a_i}}(a_i)$ and broadcasts `accept` or `fail` accordingly. If all is OK, P_j computes $a = \sum_i a_i$, else we abort. We can modify this to opening a value $[a]$ to all parties, by opening as above to every P_j.

Addition: Given two key-consistent representations as above we get that

$$[a + a'] = (a_i + a'_i, \{K^i_{a_j} + K^i_{a'_j}, m_j(a_i) + m_j(a'_i)\}^n_{j=1})^n_{i=1}$$

is a consistent representation of $a+a'$. This new representation can be computed by only local operations.

Multiplication by constants: In a similar way, we can multiply a public constant δ "into" a representation. This is written $\delta[a]$ and is taken to mean that all parties multiply their shares, keys and MACs by δ. This gives a consistent representation $[\delta a]$.

Addition of constants: We can add a public constant δ into a representation. This is written $\delta + [a]$ and is taken to mean that P_1 will add δ to his share a_1. Also, each P_j will replace his key $K^j_{a_1} = (\alpha^j_1, \beta^j_{a_1})$ by $K^j_{a_1+\delta} = (\alpha^j_1, \beta^j_{a_1} - \delta\alpha^j_1)$. This will ensure that the MACs held by P_1 will now be valid for the new share $a_1 + \delta$, so we now have a consistent representation $[a + \delta]$.

We now specify what exactly we need the preprocessing phase to do for us, namely it must implement the ideal functionality $\mathsf{F}_{\texttt{TRIP}}$. For multiplication and input sharing later, we will need it to output both random single values $[a]$ and triples $[a], [b], [c]$ where a, b are random and $c = ab \mod p$. Also, all singles and triples produced must be key consistent, so that we can freely add them together.

The specification of the functionality follows an important general principle that is very useful whenever a functionality is supposed to output shares of secret data to the players: The environment is allowed to specify to the functionality all the data that the corrupted parties should hold, including all shares of secrets, keys and MACs. Then, the functionality chooses the secrets to be shared and constructs the data for honest parties so it is consistent with the secrets and the data specified by the environment.

This may at first sight seem overly complicated: why don't we just let the functionality choose all shares and output them to the players? However, it is important to understand that if we had specified the functionality this way, it could not be implemented! The point is that the simulator we need to construct to prove an implementation secure needs to show to the environment a simulated execution of the protocol we use to create the triples. The shares, keys and MACs for corrupt parties created here must be consistent with the data that $\mathsf{F}_{\texttt{TRIP}}$ creates for honest parties, since this will always be the case for a real execution. This consistency is only ensured if the functionality is willing to be consistent with what the simulator wants.

Note also that $\mathsf{F}_{\texttt{TRIP}}$ may get input `stop` from the environment and in this case will not generate any output. This is because we work with dishonest majority where termination cannot be guaranteed.

Since $\mathsf{F}_{\texttt{TRIP}}$ outputs random multiplication triples, we can securely compute multiplications exactly as we saw in the section on circuit randomization. This leads to Protocol On-Line Phase, MAC-based, which is essentially Protocol On-Line Phase, specialized for the MAC-based representation we use here. Note that since we allow a majority of the players to be corrupt, we cannot guarantee that the protocol terminates and gives output to the honest players. This means we can only hope to implement a weaker version of secure function evaluation $\mathsf{F}_{\texttt{SFE}}^{f,\texttt{abort}}$, where the functionality can be aborted before honest players get output (but where corrupted players may have received theirs).

Let $\pi_{\texttt{ON-LINE}}$ denote Protocol On-Line Phase, MAC-based. Then we have

Theorem 8.7 *$\pi_{\texttt{ON-LINE}} \diamond \mathsf{F}_{\texttt{TRIP}}$ implements $\mathsf{F}_{\texttt{SFE}}^{f,abort}$ in* $\mathrm{Env}^{t,sync,static}$ *with statistical security for all $t < n$.*

Proof We construct a simulator $\mathcal{S}_{\texttt{ON-LINE}}$ such that a poly-time environment $\mathcal{Z}$ cannot distinguish between $\pi_{\texttt{ON-LINE}} \diamond \mathsf{F}_{\texttt{TRIP}}$ and $\mathsf{F}_{\texttt{SFE}}^{f,\texttt{abort}} \diamond \mathcal{S}_{\texttt{ON-LINE}}$. We assume here static, active corruption. The simulator will internally run a copy of $\mathsf{F}_{\texttt{TRIP}}$ composed with $\pi_{\texttt{ON-LINE}}$ where it corrupts the parties specified by $\mathcal{Z}$. The simulator relays messages between parties/$\mathsf{F}_{\texttt{TRIP}}$ and $\mathcal{Z}$, such that $\mathcal{Z}$ will see the same interface as when interacting with a real protocol. During the run of the internal

Agent $F_{\texttt{TRIP}}$

Ideal functionality specifying the preprocessing needed for the $[\cdot]$-representation. Whenever a command is received, the functionality leaks the name of the command and whatever public input is given. Output from a command is sent once the round in which to deliver is specified on the influence port.

Initialize: On input $(\texttt{init}, \mathbb{F})$ from all parties the functionality stores a description of the field $\mathbb{F}$. For each corrupted party P_i the environment specifies values $\alpha^i_j, j = 1, \ldots, n$, except those α^i_j where both P_i and P_j are corrupt. For each honest P_i, the functionality chooses $\alpha^i_j, j = 1, \ldots, n$ at random.

Singles: On input $(\texttt{singles}, u)$ from all parties P_i, the functionality does the following, for $v = 1, \ldots, u$:

1. It waits to get from the environment either $\texttt{stop}$, or some data as specified below. In the first case it sends $\texttt{fail}$ to all honest parties and stops. In the second case, the environment specifies for each corrupt party P_i, a share a_i and n pairs of values $(m_j(a_i), \beta^i_{a_j}), j = 1, \ldots, n$, except those $(m_j(a_i), \beta^i_{a_j})$ where both P_i and P_j are corrupt.
2. The functionality chooses $a \in \mathbb{F}$ at random and creates the representation $[a]$ as follows:
   1. First it chooses random shares for the honest parties such that the sum of these and those specified by the environment is correct: Let A be the set of corrupt parties, then a_i is chosen at random for $\mathsf{P}_i \notin A$, subject to $a = \sum_i a_i$.
   2. For each honest P_i, and $j = 1, \ldots, n$, $\beta^i_{a_j}$ is chosen as follows: if P_j is honest, $\beta^i_{a_j}$ is chosen at random, otherwise it sets $\beta^i_{a_j} = m_i(a_j) - \alpha^i_j a_j$. Note that the environment already specified $m_i(a_j), a_j$, so what is done here is to construct the key to be held by P_i to be consistent with the share and MAC chosen by the environment.
   3. For all $i = 1, \ldots, n, j = 1, \ldots, n$ it sets $K^i_{a_j} = (\alpha^i_j, \beta^i_{a_j})$, and computes $m_j(a_i) = \text{MAC}_{K^j_{a_i}}(a_i)$.
   4. Now all data for $[a]$ is created. The functionality sends $a_i, \{K^i_{a_j}, m_j(a_i)\}_{j=1,\ldots,n}$ to each honest P_i (no need to send anything to corrupt parties, the environment already has the data).

Triples: On input $(\texttt{triples}, u)$ from all parties P_i, the functionality does the following, for $v = 1, \ldots, u$:

1. Step 1 is done as in **Singles**.
2. For each triple to create it chooses a, b at random and sets $c = ab$. Now it creates representations $[a], [b], [c]$, each as in Step 2 in "Singles".

protocol the simulator will keep copies of the shares, MACs and keys of both honest and corrupted parties and update them according to the execution.

The idea is now, that the simulator runs the protocol with the environment, where it plays the role of the honest parties. Since the inputs of the honest parties are not known to the simulator these will be random values (or zero). However, the environment will not be able to tell the difference.

In general, the openings in the protocol do not reveal any information about

Protocol On-Line Phase, MAC-based

The protocol assumes access to $\mathsf{F_{TRIP}}$, and we assume that a field $\mathbb{F}$ and a circuit C to compute securely are given at some point after $\mathsf{F_{TRIP}}$ has been invoked. Protocol Operations on $[\cdot]$-representations is used for linear operations on representations.

Initialize: The parties first invoke $\mathsf{F_{TRIP}}(\texttt{init}, \mathbb{F})$. Then, they invoke $\mathsf{F_{TRIP}}(\texttt{triples}, u)$ and $\mathsf{F_{TRIP}}(\texttt{singles}, u)$ to create enough singles and triples. Here, only an upper bound on the size of circuit to compute needs to be known.

At some later stage when C and the inputs are known, do the following:

Input Sharing: To share P_i's input $[x_i]$, P_i takes a single $[a]$ from the set of available ones. Then, the following is performed:

1. $[a]$ is opened to P_i.
2. P_i broadcasts $\delta = x_i - a$.
3. The parties compute $[x_i] = [a] + \delta$.

We then go through the circuit and process the gates one by one in the computational order. Just after the input sharing phase, we consider all input gates as being processed. We will maintain the following:

Invariant: Computing with the circuit on inputs $x_1, \ldots, x_n$ assigns a unique value to every wire. Consider an input or an output wire for any gate, and let $a \in \mathbb{F}$ be the value assigned to this wire. Then, if the gate has been processed, the players hold $[a]$.

Computation Phase: Repeat the following until all gates have been processed (then go to the next phase): Consider the first gate in the computational order that has not been processed yet. According to the type of gate, do one of the following

Addition gate: The players hold $[a], [b]$ for the two inputs a, b to the gate. The players compute $[a] + [b] = [a + b]$.

Multiply-by-constant gate: The players hold $[a]$ for the input a to the gate, and public constant α. The players compute $\alpha[a] = [\alpha a]$.

Multiplication gate: The players hold $[a], [b]$ for the two inputs a, b to the gate. Take the next unused triple $[x], [y], [z]$ from the preprocessing phase.

1. Players compute $[a] - [x] = [a - x]$, $[b] - [y] = [b - y]$, and open $e = a - x, d = b - y$.
2. Players can compute locally $[ab] = [z] + e[b] + d[a] - ed$.

Output Reconstruction: At this point all gates, including the output gates have been processed. So for each output gate, the players hold $[y]$ where y is the value assigned to the output gate. If P_i is to learn this value, we open $[y]$ to P_i.

the actual values in the computation. Whenever we open a value during **Input** or **Multiply** we mask it by subtracting with a new random value. Therefore, the distribution of the view of the corrupted parties is exactly the same in the simulation as in the real case. Then, the only method there is left for the environment to distinguish between the two cases, is to compare the protocol execution with the inputs and outputs of the honest parties and check for inconsistency.

Agent Simulator $\mathcal{S}_{\texttt{ON-LINE}}$

In the following, H and C represent the set of corrupted and honest parties, respectively.

Initialize: The simulator initializes a copy of $\mathsf{F}_{\texttt{TRIP}}$ with $\mathbb{F}$ as the field to use and creates the desired number of triples. Note that the simulator can read all data that corrupted parties specify to the copy of $\mathsf{F}_{\texttt{TRIP}}$.

Input: If $\mathsf{P}_i \in H$ the protocol is run honestly, but with dummy input, for example 0. If in Step 1 during input, the MACs are not correct, the protocol is aborted.
If $\mathsf{P}_i \in C$ the input step is done honestly and then the simulator waits for P_i to broadcast δ. Given this, the simulator can compute $x_i' = a + \delta$ since it knows (all the shares of) a. This is the supposed input of P_i, which the simulator now gives to the ideal functionality $\mathsf{F}_{\mathsf{SFE}}^{f,\texttt{abort}}$.

Add/Multiply-by-constant: The simulator runs the protocol honestly.

Multiply: The simulator runs the protocol honestly and, as before, aborts if some share from a corrupted party is not correct.

Output: If the receiving party $\mathsf{P}_i \in H$, the output step is run and the protocol is aborted if some share from a corrupted party is not correct. Otherwise the simulator tells $\mathsf{F}_{\mathsf{SFE}}^{f,\texttt{abort}}$ to give output to P_i.
If $\mathsf{P}_i \in C$ the simulator asks $\mathsf{F}_{\mathsf{SFE}}^{f,\texttt{abort}}$ for output Since P_i is corrupted the ideal functionality will provide the simulator with y, which is the output to P_i. Now it has to simulate shares y_j of honest parties such that they are consistent with y. This is done by changing one of the internal shares of an honest party. Let P_k be that party. The new share is now computed as $y_k' = y - \sum_{i \neq k} y_i$. Next, a valid MAC for y_k' is needed. This, the simulator can compute from scratch as $\mathrm{MAC}_{K_{y_k}^i}(y_k')$ since it knows from the beginning the keys of P_i. This enables it to compute $K_{y_k}^i$ by the computations on representations done during the protocol. Now the simulator sends the internal shares and corresponding MACs to P_i.

If the simulated protocol fails at some point because of a wrong MAC, the simulator aborts which is consistent with the internal state of the ideal functionality since, in this case, the simulator also makes the ideal functionality fail.

If the simulated protocol succeeds, the ideal functionality is always told to output the result of the function evaluation. This result is of course the correct evaluation of the input matching the shares that were read from the corrupted parties in the beginning. Therefore, if the corrupted parties during the protocol successfully cheat with their shares, this would not be consistent. However, as we have seen earlier, the probability of a party being able to claim a wrong value for a given MAC is $1/|\mathbb{F}|$. Therefore, since we abort as soon as a check fails, we conclude that if the protocol succeeds, the computation is correct except with probability $1/|\mathbb{F}|$. □

Exercise 8.5 Show that $\pi_{\texttt{ON-LINE}} \diamond \mathsf{F}_{\texttt{TRIP}}$ implements $\mathsf{F}_{\mathsf{SFE}}^{f,\texttt{abort}}$ in $\mathrm{Env}^{t,\texttt{sync}}$ with statistical security for all $t < n$, i.e. $\pi_{\texttt{ON-LINE}}$ is in fact adaptively secure.

By inspection of the protocol, one sees that the work invested by each player in $\pi_{\texttt{ON-LINE}}$ amounts to $O(n|C|)$ elementary field operations, where $|C|$ is the size of the circuit computed. This basically follows from the fact that the cost of doing the multiplications dominates, and for each of these, a player needs to check $O(n)$ MACs. Thus, for a constant number of players, the work one needs to invest to do the protocol is comparable to the work one would need to just compute the circuit in the clear with no security properties ensured. Using a more complicated protocol, one can improve this to $O(|C|)$ operations (see the Notes section for more on this).

Also note that $\pi_{\texttt{ON-LINE}}$ is only really efficient if the circuit C is defined over a large field $\mathbb{F}$, where large means that $1/|\mathbb{F}|$ is an acceptable error probability. If this is not the case, we could have more than one MAC on each value. This would make the error smaller, but the protocol would be less efficient. There are, however, solutions that are as efficient as $\pi_{\texttt{ON-LINE}}$ even for the field with 2 elements, i.e., for Boolean circuits (see the Notes section for more details).

Of course, in order to actually use a protocol like $\pi_{\texttt{ON-LINE}}$ in practice, one needs to also implement the preprocessing. This will require use of computationally secure cryptography (since otherwise we would contradict the fact that information theoretic security is not possible with dishonest majority), the details of this are not the main focus of this book. We just mention that it can be done quite efficiently using public-key cryptosystems with homomorphic properties. It is not hard to see why an additively homomorphic scheme would help here: suppose player P_i has a key pair (sk, pk) and holds the secret key sk while other players have the public key pk. Suppose further that the encryption function is linearly homomorphic over $\mathbb{F}$, i.e., from ciphertexts $E_{pk}(a), E_{pk}(b)$ and constant α one can efficiently compute a new ciphertext $E_{pk}(a + \alpha b)$ (where we suppress the randomness involved in the encryption for simplicity).

Now, if P_i has a message a and P_j holds a MAC key (α, β), then P_i can send $E_{pk}(a)$ to P_j who can then compute and return to P_i a new ciphertext $E_{pk}(\alpha a + \beta)$. Decrypting this, P_i will get a MAC on a under P_j's key. If P_j is able to do his operations such that the ciphertext he returns is a *random* ciphertext containing $\alpha a + \beta$, we are sure that P_i learns nothing except for the MAC and hence cannot beak the MAC scheme later. However, to have active security, we still need to ensure that players send well-formed ciphertexts computed according to the protocol. The notes section has pointers to literature with details on how this can be done.

8.6 Open Problems in Complexity of MPC

In this section we give a short and very informal overview of some things we know about the complexity of multiparty computation with unconditional security and mention some open problems.

We begin with the model that we consider in most of this book, namely we assume secure point-to-point channels (and broadcast in case more than $n/3$ players are corrupt) and less than $n/2$ corruptions. For this case, the ideas of circuit randomization, hyper-invertible matrices, packed secret sharing and more lead to protocols where the total work players need to invest to compute circuit C securely is essentially $|C|$ field operations (where we ignore some factors that depend logarithmically on n and $|C|$, the notes section has more details). Since computing C securely in particular means you compute it, we cannot hope to do better. It should be noted, however, that this result only holds in an asymptotic sense for a large number of players and large circuits, and the underlying protocols are not practical even if they are asymptotically efficient. The upper bound on the computational work also implies a similar bound on the communication complexity of the protocol. However, it is completely open whether we actually need to send $\Omega(|C|)$ field elements to compute C. For computational security, we know that fully homomorphic encryption (FHE) allows us to have protocols where the communication does not depend on $|C|$, but we do not know if something like "FHE for unconditional security" exists.

Another issue is the number of messages we need to send to compute C. Or, in case of synchronous networks, the number of rounds we need. All the general protocols we have seen require a number of rounds that is linear in the depth of C. We do not know if this is inherent, in fact, we do not even know which functions can be computed with unconditional security and a constant number of rounds. We do know large classes of functions that can indeed be computed with unconditional security and a constant number of rounds, but there is nothing to suggest that these functions are the only ones with this property.

If we go to the preprocessing model, we know protocols that require total computational work essentially $\Omega(n|C|)$ field operations. These protocols also require communication of $\Omega(n|C|)$ field elements and the players also need to store this many fields elements from the preprocessing. It can be shown that storage required is optimal, and the computational work per player is optimal, in the following sense: for any player P_i, there exists a circuit C for which secure computing of it in the preprocessing model would require P_i to do $\Omega(|C|)$ operations, but it is not guaranteed that other players would also have to work this hard when computing C. It is intuitively quite reasonable that each player needs to invest $\Omega(|C|)$ operations: the circuit needs to be computed, so it is clear that the *total* number of field operations done by the players should be $\Omega(|C|)$. However, it might be the case that only one player is honest and everybody else works against him, so it would be quite surprising if this player would not have to do work at least equivalent to computing the circuit on his own. We emphasize, however, that such a result is not known in full generality at the time of writing.

As for communication and rounds, also in the preprocessing model, it is completely open what the optimal complexities are.

8.7 Notes

The circuit randomization idea from the first section of the chapter was suggested by Beaver [8].

The hyper-invertible matrices and applications of these was suggested by Beerliová-Trubíniová and Hirt [11], improving on the use of so-called super-invertible matrices in [**?**].

Packed secret sharing was suggested by Franklin and Yung [95].

The work in [11] is one of many results in a line of research known as scalable MPC, where one tries to build protocols that scale well as the number of players increase. In [11] it is shown how circuit randomization and hyper-invertible matrices lead to perfectly secure evaluation of a circuit C for less than $n/3$ corrupted players and secure point-to-point channels with total communication complexity $O(n|C|)$ field elements plus an additive overhead that depends polynomially on the number of players and linearly on the depth of C, but not on the size of C. To reach this goal, [11] also does an in-depth treatment of how one deals with the case where Protocol CHECK CONSISTENT SECRET-SHARES is not successful.

Ben-Sasson *et al.* [17] show a similar result for less than $n/2$ corruptions when broadcast is also given for free. They show a protocol with statistical security (we cannot do better in this model) and communication complexity $O((n \log n + k/n^c)|C|)$ field elements where k is the security parameter and c is an arbitrary constant. Here, there is again an additive term that depends polynomially on n and k and the depth of C but not $|C|$.

A closely related line of research tries to minimize also the computational work as a function of the number of players. The latest work in this line is by Damgård *et al.* [76] who show a perfectly secure protocol for less than $(1/3 - \epsilon)n$ corruptions, where $\epsilon > 0$ is an arbitrary constant. In this protocol the total number of field operations players need to do is $|C| \log |C| \log^2 n$ plus an additive term that depends polynomially on n and the depth of C but only logarithmically on $|C|$. On one hand this is stronger than [11] because the dependency on n is reduced and it is the computational work that is optimized. On the other hand, the corruption tolerance is not quite optimal (because of the ϵ).

The protocol in [76] is based on packed secret sharing and therefore needs to work with only block-wise operations where the same operation is done on all positions in a block, but still one needs to compute an arbitrary Boolean circuit. This is handled using a technique by which arbitrary circuit can be restructured so it computes the same function but the new circuit only uses block-wise operations and a small number of intra-block permutation, and is a logarithmic factor larger. We have already seen passively secure protocols for packed secret sharing, active security is obtained using error correction which works if the number of honest players is large enough. The use of error correction and packed secret sharing means that the basic protocol π is only secure against a small (but constant) fraction of corrupted players. This is then boosted to the almost optimal threshold of $(1/3 - \epsilon)n$ using the *committees technique*. Here, we form a number of committees each consisting of a constant number of players. Each committee emulates a player in π. The emulation is done using an expensive protocol with optimal threshold (but this does not hurt overall asymptotic efficiency as committees have constant size). If at most $(1/3 - \epsilon)n$ players are corrupt, we can choose the committees in such a way that only a small constant fraction of them contain more than a third corrupted players. This means that the (emulated) execution of π will be secure.

The committees approach was suggested by Bracha [31] for the purpose of broadcast and was introduced in multiparty computation in [77].

We emphasize that all the results on scalable MPC that we mentioned hold asymptotically for a large number of players and even larger circuits. This is because the protocols typically use the approach we saw in Protocol Check Consistent Secret-Shares, where one runs a protocol that very efficiently handles a large number of secret-sharings and works correctly as long as players follow the protocol. On the other hand, honest players can verify that the protocol has been followed and complain if this is not the case. If complaints occur we need to resolve the conflicts and this usually requires doing something very expensive. This approach makes sense if the computation we do is large enough that it makes sense to do many secret-sharings in parallel, *and* if we can make sure that complaints will not occur very often, in which case the cost of handling them will be insignificant compared to the rest of the work.

The material in the section on preprocessing is from Bendlin *et al.* [21]. This paper also introduces the notion of semi-homomorphic public-key encryption and shows it is sufficient for implementing the preprocessing specified in $\mathsf{F}_{\mathtt{TRIP}}$. In a subsequent paper, Damgård *et al.*[80] show that the total computational complexity can be improved from quadratic in n to $O(n|C| + n^3)$ field operations, where the error probability is $1/|\mathbb{F}|$. The amount of data players have to store from the preprocessing as well as the communication complexity is also $O(n|C| + n^3)$ field elements. It is also shown in [80] that the required storage from the preprocessing is optimal and the computational work is optimal in the sense explained earlier: for any player P_i, there exists a circuit C for which secure computing of it in the preprocessing model would require P_i to do $\Omega(|C|)$ operations, but it is not guaranteed that other players would also have to work this hard. Finally, [80] shows how to implement the preprocessing more efficiently based on somewhat homomorphic encryption. Informally, this is a weaker form of fully homomorphic encryption where many additions but only a small bounded number of multiplications can be performed.

9

Applications of MPC

Contents

In this chapter we will look at two different applications of information theoretic MPC, a practical application and a theoretical application. The example of a practical application is the use of MPC to clear a commodity derivative market. The focus will be on the algorithmic tricks used to implement the auction efficiently. The theoretical application is the use of MPC to realize so-called zero-knowledge proofs. A zero-knowledge proof is a way for a prover to convince a verifier about the validity of a statement without leaking any information on why the statement is true. This can be seen as a multiparty computation problem with $n = 2$ parties. However, since the minimal requirement for information theoretic MPC is that less than $n/2$ parties are corrupted, information theoretic MPC does not seem to help in constructing zero-knowledge proofs. However, as we shall see, a technique sometimes called MPC in the head can be used to turn an efficient secure multiparty computation for a given relation into an efficient zero-knowledge proof for the same relation.

9.1 A Double Auction

In this section we look at a concrete application of MPC, with a main focus on the algorithmic tricks needed to efficiently do a secure auction. Along the way, we will look at how to efficiently and securely compare two integers secret shared among the parties.

9.1.1 Introduction

The algorithmic techniques we will look at are fairly general, but it is instructive to view them in a practical context. We will look at how they have been used to clear the Danish market for contracts on sugar beets from 2008 and until the time of writing. This was the first industrial application of MPC. More historical details on this can be found below and in the notes section.

In the economic field of mechanism design the concept of a trusted third party has been a central assumption since the 70's. Ever since the field was initiated it has grown in momentum and turned into a truly cross disciplinary field. Today, many practical mechanisms require a trusted third party. In particular, several research projects on combining mechanism design and MPC has concentrated on two types of applications:

- Various types of auctions. This is not limited to only standard highest bid auctions with sealed bids but also includes, for instance, variants with many sellers and buyers, so-called double auctions: essentially scenarios where one wants to find a fair market price for a commodity given the existing supply and demand in the market.
- Benchmarking, where several companies want to combine information on how their businesses are running, in order to compare themselves to best practice in the area. The benchmarking process is either used for learning, planning or motivation purposes. This of course has to be done while preserving confidentiality of companies' private data.

When looking at such applications, one finds that the computation needed is basically elementary arithmetic on integers of moderate size, typically around 32 bits. More concretely, quite a wide range of the cases require only addition, multiplication and comparison of integers. The known generic MPC protocols can usually handle addition and multiplication very efficiently, by using the field $\mathbb{F} = \mathbb{Z}_p$ for a prime p chosen large enough compared to the input numbers to avoid modular reductions. This gives integer addition and multiplication by doing addition and multiplication in $\mathbb{F}$.

This is efficient because each number is shared "in one piece", as opposed to, e.g., a bit-wise sharing. Unfortunately, this input representation also implies that comparison of shared values is much harder. A generic solution would express the comparison operation as an arithmetic circuit over $\mathbb{Z}_p$, but this would be far too large to give a practical solution, because the circuit would not have access to the binary representation of the inputs. Instead, special purpose techniques for comparison have been developed. We will have a closer look at these in Section 9.1.5.

9.1.2 A Practical Application Scenario

In this section we describe the history behind the practical case in which a secure auction was deployed in 2008, as mentioned above.

In Denmark, several thousand farmers produce sugar beets, which are sold to the company Danisco, which is the only sugar producing company on the Danish market. Farmers have contracts that give them production rights, that is, a contract entitles a farmer to produce a certain amount of beets per year. These contracts can be traded between farmers, but trading has historically been very limited and has been done only via bilateral negotiations. In the years up to 2008, however, the EU drastically reduced the support for sugar beet production. This and other factors meant that there was now an urgent need to reallocate contracts to those farmers where productions payed off best. It was realized that this was best done via a nation-wide exchange, a double auction.

Briefly, the goal is to find the so-called market clearing price, which is a price per unit of the commodity that is traded. In the specific case, the commodity is a contract for producing sugar beets and the unit is tons of beets. What happens is that each buyer specifies, for each potential price, how much he is willing to buy at that price, similarly all sellers say how much they are willing to sell at each price. All bids go to an auctioneer, who computes, for each price, the total supply and demand in the market. Since we can assume that supply grows and demand decreases with increasing price, there is a price where total supply equals total demand, at least approximately, and this is the price we are looking for. Finally, all bidders who specified a non-zero amount to trade at the market clearing price get to sell/buy the amount at this price.

This could in principle be implemented with a single trusted party as the auctioneer. However, in the given scenario, there are some additional security concerns implying that this is not a satisfactory solution: Bids clearly reveal information on a farmer's economic position and her productivity, and therefore farmers would be reluctant to accept Danisco acting as auctioneer, given its position in the market. Even if Danisco would never misuse its knowledge of the bids in future price negotiations, the mere fear of this happening could affect the way farmers bid and lead to a suboptimal result of the auction. On the other hand, contracts in some cases act as security for debt that farmers have to Danisco, and hence the farmers' organization DKS running the auction independently would not be acceptable for Danisco. Finally, the common solution of delegating the legal and practical responsibility by paying, e.g., a consultancy house to be the trusted auctioneer would have been a very expensive solution. It was therefore decided to implement an electronic double auction, where the role of the auctioneer would be played by a multiparty computation done by representatives for Danisco, DKS and the research project implementing the auction.

A three-party solution was selected, partly because it was natural in the given scenario, but also because it allowed using efficient information theoretic tools such as secret sharing, rather than more expensive cryptographic methods needed when there are only two parties.

9.1.3 The Auction System

In the system that was deployed, a web server was set up for receiving bids, and three servers were set up for doing the secure computation. Before the auction started, a public/private key pair was generated for each computation server, and a representative for each involved organization stored the private key on a USB stick, protected under a password.

Encrypt and Share Curve.

Each bidder logged into the web-server and an applet was downloaded to her PC together with the public keys of the computation servers. After the user typed in her bid, the applet secret shared the bids, creating one share for each server, and encrypted the shares under the respective server's public key. Finally the entire set of ciphertexts were stored in a database by the web-server.

As for security precautions on the client side, the system did not explicitly implement any security against cheating bidders, other than verifying their identity. The reason being that the method used for encrypting bids implicitly gives some protection: it is a variant of a technique called non-interactive VSS based on pseudo-random secret sharing. We will not look at the details of this method, but using it, an encrypted bid is either obviously malformed, or is guaranteed to produce consistently shared values. This means that the only cheating that is possible, is to submit bids that are not monotone, i.e., bids where, for instance, the amount you want to buy does not decrease with increasing price, as it should. It is easy to see that this cannot be to a bidders economic advantage.

Secure Computation.

After the deadline for the auction had passed, the servers were connected to the database and each other, and the market clearing price was securely computed, as well as the quantity each bidder would buy/sell at that price. The representative for each of the involved parties triggered the computation by inserting her USB stick and entering her password on her own machine.

The computation was based on standard Shamir secret sharing over $\mathbb{F} = \mathbb{Z}_p$, where p was a 64-bit prime.

The system worked with a set of 4000 possible values for the price, meaning that after the total supply and demand had been computed for all prices, the market clearing price could be found using binary search over 4000 values, which means about 12 secure comparisons.

9.1.4 Practical Evaluation

The bidding phase was done in the beginning of 2008. It ran smoothly, with only a small number of technical questions asked by users. The only issue observed was that the applet on some PC's took up to a minute to complete the encryption of the bids. It is not surprising that the applet needed a non-trivial amount of time, since each bid consisted of 4000 numbers that had to be handled individually. A total of 1200 bidders participated in the auction, each of these had the option of submitting a bid for selling, for buying, or both.

The actual secure computation was done January 14, 2008 and lasted about

30 minutes. Most of this time was spent on decrypting shares of the individual bids, which is not surprising, as the input to the computation consisted of about 9 million individual numbers. As a result of the auction, about 25.000 tons of production rights changed owner.

9.1.5 Implementation Details

In this section we describe the technical details of how the secure computation of the market clearing price was performed.

We assume that there are P different prices, $p_1, \ldots, p_P$, B buyers, S sellers, and n servers. We start at a point where each buyer $j = 1, \ldots, B$ for each price p_i shared an integer $d_{i,j}$ among the servers. We denote the sharing by $[d_{i,j}]$. The integer $d_{i,j}$ specifies how much the buyer is willing to buy if the price turns out to be p_i. Similarly, we assume that each seller $j = 1, \ldots, S$ for each price p_i shared an integer $s_{i,j}$ among the servers. We denote the sharing by $[s_{i,j}]$. The integer $s_{i,j}$ specifies how much the seller is willing to sell if the price turns out to be p_i.

Discrete Market Clearing Price

The goal, given by economic mechanism design, is now to find the price where most goods are moved. For this, the servers first run the following code:

1. For all prices p_i, compute sharings

$$[d_i] = \sum_{j=1}^{S} [d_{i,j}]$$

   and

$$[s_i] = \sum_{j=1}^{S} [s_{i,j}] \ ,$$

   by locally adding shares, i.e., use the protocol for adding shared values.
2. The output is a secret shared demand "curve" $[d_1], \ldots, [d_P]$ and a secret shared supply "curve" $[s_1], \ldots, [s_P]$.

If the demand d and supply s were continuous functions of the price p and were monotonously decreasing respectively monotonously increasing, then the price p_{MCP} where most goods would be traded is given by $d(p_{\text{MCP}}) = s(p_{\text{MCP}})$. Below this price the supply is smaller (or at least not larger) and above this price the demand is smaller (or at least not larger), which leads to a lower amount being traded (or at least not more).

We assume that the curves computed in shared form are monotone as specified above. They are, however, not continuous. We will therefore first compute

$$i_{\text{MCP}} = \max\{i : d_{i_{\text{MCP}}} > s_{i_{\text{MCP}}}\} \ .$$

We can assume that $d_1 > s_1$ (by, e.g., setting $p_1 = 0$), which ensures that i_{MCP} is well-defined.[1] It is then easy to see that the price trading most goods is $p_{i_{\text{MCP}}}$ or $p_{i_{\text{MCP}}+1}$. For our purpose here we simply define the discrete market clearing price to be $p_{i_{\text{MCP}}}$. If the price grid is fine enough we expect the amount of goods being traded at $p_{i_{\text{MCP}}}$ and $p_{i_{\text{MCP}}+1}$ to be the same for all practical purposes. In the practical application more involved choices were used to choose between the two possible clearing prices.

Binary Search

To find i_{MCP} the servers simply do a binary search on $i \in \{1, \dots, P\}$. They start with $i = \lceil P/2 \rceil$ and securely test whether $d_i > s_i$. If so, they go to a higher price; if not, they go to a lower price. This way they arrive at i_{MCP} using $\log_2(P)$ secure comparisons. By securely testing, we mean that they leak whether $d_i > s_i$, and nothing else.

Note that doing a binary search does not violate the security goal of leaking only i_{MCP}. Given i_{MCP} one knows that $d_i > s_i$ for all $i \leq i_{\text{MCP}}$ and $d_i \leq s_i$ for all $i > i_{\text{MCP}}$. Therefore the outcome of all the comparisons done during the computation can be simulated given just the result i_{MCP}.

The binary search was important for the feasibility of the system, as each secure comparison was relatively expensive, meaning that the difference between, e.g., 4000 comparison and $\lceil \log_2(4000) \rceil = 12$ comparisons would have been a difference between running for minutes and running for hours.

After i_{MCP} is found, the individual values $d_{i_{\text{MCP}},j}$ and $s_{i_{\text{MCP}},j}$ are reconstructed, and the goods are traded at price $p_{i_{\text{MCP}}}$ and in the revealed amounts.

9.1.6 Secure Comparison

The above approach leaves us only with the problem of taking two secret shared integers $[a]$ and $[b]$ and securely computing a bit $c \in \{0, 1\}$, where $c = 1$ if and only if $a > b$. We write

$$c = [a] \overset{?}{>} [b] \ .$$

Unfortunately there is no efficient algorithm for computing c from a and b using only addition and multiplication modulo p. Instead we will take an approach which involves first securely computing sharings of the individual bits of a and b, and then performing the comparison on the bit-wise representations.

Assume first that we have a protocol Bits which given a sharing $[a]$ securely computes sharings $[a_\ell], \dots, [a_0]$, where $\ell = \lfloor \log_2(p) \rfloor$ and $a_\ell \cdots a_0$ is the binary representation of a, with a_0 being the least significant bit. I.e., $a = \sum_{i=0}^{\ell} 2^i a_i$. Assume furthermore that we have a protocol MS1 which given sharings $[c_\ell], \dots, [c_0]$ of bits computes sharings $[d_\ell], \dots, [d_0]$ of bits, where $d_i = 1$ for the largest i for which $c_i = 1$ and where $d_i = 0$ for all other i — if all $c_i = 0$, then let all $d_i = 0$.

[1] Alternatively we can introduce some dummy price p_0 and define $s_0 = 0$ and d_0 to be the maximal amount and create some dummy sharings $[s_0]$ and $[d_0]$.

Protocol Compare

Input: Sharings $[a]$ and $[b]$.
Output: A sharing $[c]$ where $c = 1$ if $a > b$ and $c = 0$ otherwise.
Algorithm:

```
1. $([a_\ell], \ldots, [a_0]) = \textsc{Bits}([a])$.
2. $([b_\ell], \ldots, [b_0]) = \textsc{Bits}([b])$.
3. For $i = 0, \ldots, \ell$: $[c_i] = [a_i] + [b_i] - 2[a_i][b_i]$.
4. $([d_\ell], \ldots, [d_0]) = \mathrm{MS1}([c_\ell], \ldots, [c_0])$.
5. For $i = 0, \ldots, \ell$: $[e_i] = [a_i][d_i]$.
6. $[c] = \sum_{i=0}^{\ell} [e_i]$.
```

I.e., $[d_\ell], \ldots, [d_0]$ can be seen as a unary representation of the index i of the most significant 1 in c.

We assume that it never happens that $a = b$. I.e., $a > b$ or $b > a$. This can be guaranteed by introducing some dummy, different least significant bits. The comparison is then performed as in Protocol Compare. A computation like $[a_i] + [b_i] - 2[a_i][b_i]$ means that the parties first run the multiplication protocol to compute a sharing of $a_i b_i$. Then they run the multiplication protocol with the result of this and a dummy sharing of -2 to get a sharing of $-2a_i b_i$. Then they run the addition protocol twice to get a sharing of $a_i + b_i - 2a_i b_i$.

It is easy to see that $c_i \in \{0, 1\}$ and $c_i = 1$ if and only if $a_i \neq b_i$. So, the i for which $d_i = 1$ is the most significant bit position in which a and b differ. Therefore $a > b$ if and only if $a_i > b_i$ for the i where $d_i = 1$, and as $a_i \neq b_i$ we have that $a_i > b_i$ if and only if $a_i = 1$. So, the result is $c = a_i$ for the i where $d_i = 1$, which can be computed as $\sum_j a_j d_j$, as $d_j = 0$ for all other positions. So, the value of c is correct.

Clearly the protocol is secure, as no values are opened during the computation—all computations are done on secret shared values using secure sub-protocols. Note that we computed a sharing of c. When c is needed one can simply reconstruct $[c]$ towards all parties. If the comparison is done as a part of a larger secure computation it is, however, in some cases necessary to not leak c. We will see an example of that below.

We also use a version BitCompare which starts with bit-wise sharings of the numbers $([a_L], \ldots, [a_0])$ and $([b_L], \ldots, [b_0])$ and computes $[c]$.[2] Such a protocol is a special case of Protocol Compare, and does not need Bits as sub-protocol.

Since Protocol MS1 starts with a binary representation of the inputs, it is straight forward to construct a secure protocol with the desired functionality. A general approach is to start with a Boolean circuit for the desired functionality and then simulate this circuit using addition and multiplication. However, sometimes a carefully constructed protocol can do slightly better. One approach to computing a unary representation of the most significant 1 is given in Protocol MS1. Note that it only uses addition and multiplication of shared values as it

[2] We use L as bound as it might be the case that $L \neq \ell$.

```
Protocol MS1

Input: Sharings $[c_\ell], \dots, [c_0]$ of bits.
Output: Sharings $[d_\ell], \dots, [d_0]$, where $d_\ell \cdots d_0 = \mathrm{MS1}(c_\ell \cdots c_0)$.
Algorithm: 1. Let $[f_{\ell+1}]$ be a dummy sharing of 1.
           2. For $i = \ell, \dots, 0$: $[f_i] \leftarrow [f_{i+1}](1 - [c_i])$.
           3. For $i = \ell, \dots, 0$: $[d_i] \leftarrow [f_{i+1}] - [f_i]$.
```

should. As for the correctness, note that f_i is always a value from $\{0, 1\}$ and that it is 0 if and only if $f_{i+1} = 0$ or $c_i = 1$. In particular, when $f_j = 0$, then $f_k = 0$ for all $k < j$. Furthermore, since we start with $f_{\ell+1} = 1$ it follows that $f_i = 1$ for $i = \ell, \dots, i_0$, where i_0 is the smallest index such that all the bits $c_\ell, \dots, c_{i_1}$ are 0. As an example, if $c = c_7 \cdots c_0 = 00010010$, then $f_8 f_7 \cdots f_0 = 111100000$. Note then that $f_{i+1} - f_i$ is 0 if $f_{i+1} f_i = 11$ or $f_{i+1} f_i = 00$ and that $f_{i+1} - f_i$ is 1 if $f_{i+1} f_i = 10$. So, d_i will be 0 everywhere, except where the sequence 111100000 changes from 1 to 0. In our example where $f_8 f_7 \cdots f_0 = 111100000$ we get that $d_7 \cdots, d_0 = 00010000$. In general, d_i will be 1 exactly where the first 1 is found in c, as it should be.

Using similar techniques we can also develop an efficient and secure Protocol BitAdd, which takes bit-wise sharings $([a_L], \dots, [a_0])$ and $([b_L], \dots, [b_0])$ and produces a bit-wise sharing $([c_{L+1}], [c_L], \dots, [c_0])$ of $c = a + b$ (without any modular reduction). And we can produce a secure Protocol BitSub, which takes bit-wise sharings $([a_L], \dots, [a_0])$ and $([b_L], \dots, [b_0])$ of $a \geq b$ and produces a bit-wise sharing $([c_L], \dots, [c_0])$ of $c = a - b$. For these protocols we also use versions where some inputs might be known bits instead of shared bits, e.g., b_0 instead of $[b_0]$: simply first do a dummy sharing $[b_0]$ of b_0 and then run the corresponding protocol for shared values.

Exercise 9.1 Implement BitAdd and BitSub securely using in the order of ℓ secure multiplications. The number of secure additions is not important, as they are cheap in that they do not require communication. [Hint: Iterate from the least significant bit to the most significant one, and keep a carry bit. The challenge is to only use addition and multiplication.] How few multiplications can you do with?

We then turn our attention to Protocol Bits. At first this seems as a hard challenge, as it is not clear how to access an individual bit using only addition and multiplication, but as often is the case in MPC, a random self reduction will do the job. I.e., we generate a random solved instance of the problem, and then use it to solve the original problem. In doing that, we use the randomness of the solved instance to mask the original instance, which allows to reveal the masked values and do most of the hard operations on plaintext. This is a very common technique.

Assume that we have a secure Protocol RandomSolvedBits which outputs a random sharing $[r]$ of a random number r from $\mathbb{Z}_p$, along with sharings of the bits

```
Protocol Bits

Input: A sharing $[a]$ of $a \in \mathbb{Z}_p$.
Output: Sharings $[f_\ell], \ldots, [f_0]$ of the bits of $a$.
Algorithm: 1. $([r], [r_\ell], \ldots, [r_0]) \leftarrow$ RandomSolvedBits.
           2. $[c] = [a] - [r]$.
           3. $c \leftarrow$ Reconstruct$([c])$.
           4. Write $c$ in binary $c_\ell, \ldots, c_0$.
           5. $([d_{\ell+1}], \ldots, [d_0]) \leftarrow$ BitAdd$(([r_\ell], \ldots, [r_0]), (c_\ell, \ldots, c_0))$.
           6. Write $p$ in binary $(p_{\ell+1}, p_\ell, \ldots, p_0)$.
           7. $[e] =$ BitCompare$(([d_{\ell+1}], \ldots, [d_0]), (p_{\ell+1}, \ldots, p_0))$.
           8. For $i = 0, \ldots, \ell + 1$: $[ep_i] = [e][p_i]$.
           9. $([f_{\ell+1}], \ldots, [f_0]) \leftarrow$ BitSub$(([d_{\ell+1}], \ldots, [d_0]), ([ep_{\ell+1}], \ldots, [ep_0]))$.
```

$([r_\ell], \ldots, [r_0])$ of r. As we show later, producing such a random solved instance is fairly easy. The protocol then proceeds as described in Protocol Bits.

The protocol reconstructs only one value, namely c. Since $c = a - r \bmod p$ and r is an unknown uniformly random value in $\mathbb{Z}_p$ independent of a, the value c is a uniformly random number in $\mathbb{Z}_p$ independent of a, so it is secure to let all parties learn c. Therefore the protocol is secure. It takes a little more work to see that it is correct.

First note that $d = c + r$. Since $c = a - r \bmod p$ this means that $d = a$ or $d = a + p$. The later case happens when $r > a$ such that $a - r \bmod p$ gives a wrap around. We find out which case we are in by (securely) comparing d to p: the bit e is 1 if $d = a + p$ and it is 0 if $d = a$ — we can write this as $d = a + ep$. The next step then (securely) subtracts ep from d. In the for-loop we compute a bit-wise sharing of ep. Then we use a secure subtraction. It follows that $f_{\ell+1}, f_\ell, \ldots, f_0$ is the bit-wise representation of $d - ep = a$. Since a is an ℓ-bit number we in particular know that $f_{\ell+1} = 0$, so we only have to use $[f_\ell], \ldots, [f_0]$ as the result.

```
Protocol RandomSolvedBits

Output: Sharing $[r]$ of a random unknown field element $r \in \mathbb{Z}_p$.
Output: Sharings $[r_\ell] \ldots, [r_0]$ of the bits of $r$.
Algorithm: 1. For $i = 0, \ldots, \ell$: $[r_i] \leftarrow$ RandomBit().
           2. Write $p$ in binary $p_\ell, \ldots, p_0$.
           3. $[c] \leftarrow$ BitCompare$((p_\ell, \ldots, p_0), ([r_\ell], \ldots, [r_0]))$.
           4. $c \leftarrow$ Reconstruct$([c])$.
           5. If $c = 0$, go to Step 1
           6. $[r] \leftarrow \sum_{i=0}^{\ell} 2^i [r_i]$.
```

We are then stuck with Protocol RandomSolvedBits. For a last time we push some of the burden into the future, by assuming that we have a Protocol RandomBit, which generates a sharing of a random bit. Then a random solved instance is generated as in Protocol RandomSolvedBits. We essentially generate a random element $r \in \mathbb{Z}_p$ by using rejection sampling. Since $r < p$ with probability at least $\frac{1}{2}$, by the definition of ℓ, this protocol terminates after an expected two iterations, and on termination r is clearly a uniformly random integer from $[0, p)$, as desired.

Note that the reason why generating a random solved instance is easier than solving a given instance is that we do not solve the random instance — we generate the solution (the bits) and then generate the instance from the solution.

All that is left is then to implement Protocol RandomBit. One inefficient way of doing it is to let all parties generate a sharing of a random bit and then, e.g., take the secure XOR of these bits.[3] The reason why this is inefficient is that it would require $n-1$ multiplications for each of the generated bits. With three servers that is fine, but in general it can be rather inefficient. Furthermore, it is not actively secure as the servers might not all contribute bits, and as can be seen, all the above protocols are indeed active secure if the sub-protocols are active secure, so we should try to also make the generation of random bits active secure.

Protocol RandomBit

Output: A sharing $[r]$ of a uniformly random bit r.

Algorithm:
```
1. $[a] \leftarrow$ RandomFieldElement.
2. $[a^2] = [a][a]$.
3. $A \leftarrow$ Reconstruct($[a^2]$)
4. If $A = 0$, go to Step 1.
5. $b \leftarrow \sqrt{A} \bmod p$.
6. $[c] = (b^{-1} \bmod p)[a]$.
7. $[r] = 2^{-1}([c] + 1)$.
```

Our active secure protocol is based on the fact that squaring a non-zero element modulo an odd prime is a 2-to-1 mapping, and given $b = a^2$ one has no idea if the pre-image was a or $-a$. We will let the "sign" of such an a be our random bit. We use a sub-protocol RandomFieldElement which generates a sharing of a random field element. It is implemented by each server sharing a random field element and then adding all of them. Here there is no room to cheat: as long as just one server contributes a random field element, the result is random.

The protocol is given in Algorithm RandomBit. It rejects when $A = 0$ to ensure that we started with $a \in \mathbb{Z}_p^*$. When the loop terminates, then $A = a^2 \bmod p$ for a uniformly random $a \in \mathbb{Z}_p^*$. Since A only has two square roots, as we are computing modulo a prime, and since a is one of them, it follows that $b = a$ or $b = -a \bmod p$. Since $A = a^2 \bmod p$ and $A = (-a)^2 \bmod p$, the value $b = \sqrt{A} \bmod p$ is clearly independent of whether A was computed as $A = a^2 \bmod p$ or $A = (-a)^2 \bmod p$, no matter which algorithm is used for computing the square root. Since a was chosen uniformly at random it follows that $\Pr[b = a] = \frac{1}{2}$

[3] The XOR of bits a and b can be expressed via addition and multiplication as $a + b - 2ab$.

and $\Pr[b = -a \bmod p] = \frac{1}{2}$. If $b = a$, then $b^{-1}a \bmod p = 1$ and thus $r = 1$. If $b = -a \bmod p$, then $b^{-1}a \bmod p = -1 \bmod p$ and thus $r = 0$. So, $\Pr[r = 1] = \frac{1}{2}$ and $\Pr[r = 0] = \frac{1}{2}$. Since $[a]$ is not reconstructed, no party knows whether $r = 0$ or $r = 1$.

Conclusion

By inspection of the above protocols, and given a solution to Exercise 9.1, it can be seen that all protocols use a number of multiplications in the order of $\log_2(p)$, and the unmentioned constant is fairly small. This allows to perform comparisons of shared integers relatively efficiently. The ability to split a shared number has many other applications. One is to move a number x shared modulo one prime p into a sharing modulo another prime q. If q is smaller than p, this computes a modulo reduction of x modulo q. Many similar tricks exists, and are constantly being produced, to allow efficient secure computation of specific operations.

Exercise 9.2 Above we had to compute $\sqrt{A} \bmod p$. When p is a prime and A is a square modulo p this can be done efficiently. The algorithm depends on whether $p \bmod 4 = 1$ or $p \bmod 4 = 3$. Note: if $p > 2$, then p is odd, so it cannot be the case that $p \bmod 4 \in \{0, 2\}$. If $p \bmod 4 = 3$, then $\sqrt{A} \bmod p = A^{(p+1)/4} \bmod p$. I.e., if you let $b = A^{(p+1)/4} \bmod p$, then $b^2 \bmod p = A$. Notice that $A^{(p+1)/4}$ is well defined as $p \bmod 4 = 3$ implies that $p = 4m + 3$ for some integer. Therefore $p+1 = 4m+4 = 4(m+1)$ and $(p+1)/4 = m+1$ is an integer, as it should be for $A^{(p+1)/4}$ to be defined. Prove that $b^2 \bmod p = A$. [Hint: Use that $A = a^2 \bmod p$ for some a and that $p + 1 = (p - 1) + 2$, and a little theorem.]

Exercise 9.3 Assume that you are given a sharing $[a]$ of an element $a \neq 0$, and assume that you can generate a sharing $[r]$ of a uniformly random element $r \neq 0$. Argue that it is secure to compute $[ar] = [a][r]$ and reveal ar. Show how to securely compute $[a^{-1}]$ from $[a]$.

Exercise 9.4 In a Vickrey auction there is a single item, and a number of buyers B_i. Each buyer B_i bids a price p_i. The winner is the buyer with the highest bid, and the winner gets to buy the item, but at the second highest bid. Assume that all prices are different and describe a protocol which computes the winner and the price, and which leaks noting else. You are allowed to assume that you have access to some servers of which more than half are honest. Try to make the system as efficient as possible.

9.2 Zero Knowledge Proofs via MPC in the Head

In this section we explain how efficient multiparty computation can be used to construct efficient zero-knowledge proofs of knowledge. In a zero-knowledge proof of knowledge, a prover convinces a verifier that he knows a particular secret, without revealing anything about the secret. As an example, consider a prover P who sends his public key *pk* to a verifier V and claims that he knows the secret key sk corresponding to pk. Of course P could prove this by simply sending sk to V, but this would not be secure in most settings, so we seek a way to prove such a knowledge without leaking any information.

Generally, the problem is parametrized by a binary relation $R \subset \{0,1\}^* \times \{0,1\}^*$. For a pair $(x, w) \in R$ we call x an instance and we call w the witness. The prover knows $(x, w) \in R$ and send x to the verifier, claiming to know w such that $(x, w) \in R$. In the above example, we would have $x = pk$ and $w = sk$ and R would consist of the pairs (pk, sk) for which sk is the secret key corresponding to pk. We seek a way for P to substantiate his claim without leaking anything about w to V. Phrased as an ideal functionality, the problem can be captured using Agent $\texttt{ZKPOK}^R$.

Agent $\texttt{ZKPOK}^R$

The ideal functionality $\texttt{ZKPOK}^R$ for zero-knowledge is for two parties, a prover P and a verifier V. It is parametrized by a polynomial time binary relation $R \subset \{0,1\}^* \times \{0,1\}^*$.

Proof: On input (x, w) from P, where $(x, w) \in R$, output x on `leak`, and store $(\texttt{proved}, x)$.

Delivery: On input $(\texttt{deliver}, x)$ on `infl` when some $(\texttt{proved}, x)$ is stored, output x to V.

A related problem is that of implementing a "processing channel". Here a sender S sends some message x to a receiver R. However, instead of receiving x the receiver is to receive some processed version of x, where the processing is given by some function f. If security was not an issue, the sender could just send x and let R compute the processed value $y = f(x)$. However, we seek a way to let R learn y without learning anything extra about x. Phrased as an ideal functionality, the problem can be captured using Agent $\texttt{PC}^f$.

Agent $\texttt{PC}^f$

The ideal functionality $\texttt{PC}^f$ for a processing channel is for two parties, a sender S and a receiver R. It is parametrized by a polynomial time function $f : \{0,1\}^* \to \{0,1\}^*$.

Send: On input x from S, compute $y = f(x)$, output y on `leak` and store $(\texttt{sending}, y)$.

Delivery: On input $(\texttt{deliver}, y)$ on `infl` when some $(\texttt{sending}, y)$ is stored, output y to R.

For a given binary relation R, consider the function $f_R(x, w)$ where $f_R(x, w) = x$ if $(x, w) \in R$ and $f_R(x, w) = \perp$ if $(x, w) \notin R$, where $\perp \notin \{0,1\}^*$. It should be clear that if we can securely implement $\texttt{PC}^{f_R}$, then we can also securely implement $\texttt{ZKPOK}^R$. This, in fact, follows via the UC theorem, as it is trivial to securely implement $\texttt{ZKPOK}^R$ in the hybrid model with access to $\texttt{PC}^{f_R}$. The prover will input (x, w) to $\texttt{PC}^{f_R}$, acting as S with respect to $\texttt{PC}^{f_R}$. The verifier V will act as

R with respect to $\mathtt{PC}^{f_R}$. If it receives $x \neq \perp$ from $\mathtt{PC}^{f_R}$ it output x. If it receives $x = \perp$, it does nothing. Constructing a simulator is trivial.

We can therefore focus on implementing $\mathtt{PC}^{f_R}$ for the given R. We will, in fact, show something stronger. We will show how to securely implement $\mathtt{PC}^f$ for any poly-time function f.

The first step is to consider a related problem in a setting with n parties. For a given f, consider the corresponding function $g(x_1, x_2, \ldots, x_n) = (y_1, \ldots, y_n)$, where $y_1 = y_2 = \cdots = y_n = f(x_1)$. We know from Chapter 5 that there exists a perfectly secure protocol π implementing g given secure point-to-point channels and broadcast, tolerating $t < n/3$ corrupted parties. We are going to massage this π into a protocol for two parties which securely implements $\mathtt{PC}^f$.

Our first step will be to assume that π is in so-called canonical form which means it is defined as a function:

$$\pi : \mathbb{N} \times \mathbb{N} \times \{0,1\}^{n+1} \to \{0,1\}^{n+1} .$$

Concretely, this means we assume π proceeds in ℓ rounds, where in each round each party sends one secure message to each of the other parties and broadcasts one message. Then the execution of a given party in a given round is given by

$$(x^r_{i,1}, \ldots, x^r_{i,n}, x^r_i) = \pi(r, i, y^{r-1}_{1,i}, \ldots, y^{r-1}_{n,i}, \rho_i) ,$$

where r specifies the round number, $i \in \{1, \ldots, n\}$ specifies the index of the party which is being computed, $(y^{r-1}_{1,i}, \ldots, y^{r-1}_{n,i})$ are the incoming messages from $(\mathsf{P}_1, \ldots, \mathsf{P}_n)$ respectively, and ρ_i is a randomizer from which, e.g., shares are drawn. The output $x^r_{i,j}$ is the message sent securely from P_i to P_j in round r, and x^r_i is the message broadcast by P_i in round r. The party P_i can pass an internal state to the next round by "securely sending it to itself" as $x^r_{i,i}$. In detail an honest execution of a protocol π in canonical form runs as described in Fig. 9.1.

```
1. The input for $\mathsf{P}_i$ is $x_i$.
2. For $i = 1, \ldots, n$: Set $y^0_{i,i} = x_i$ and set $y^0_{i,j \neq i} = 0$. Sample uniformly random $\rho_i$.
3. For $r = 1, \ldots, \ell$:
   1. For $i = 1, \ldots, n$:
      1. Compute $(x^r_{i,1}, \ldots, x^r_{i,n}, x^r_i) = \pi(r, i, y^{r-1}_{1,i}, \ldots, y^{r-1}_{n,i}, \rho_i)$.
      2. For $j = 1, \ldots, n$: $y^r_{i,j} = (x^r_{i,j}, x^r_i)$.
4. The output of $\mathsf{P}_i$ is $y_i = y^\ell_{i,i}$.
```

Figure 9.1 Honest execution of canonical form protocol (ℓ, π)

A global transcript of such a protocol π consist of the values

$$G = (\{x_i\}^n_{i=1}, \{\rho_i\}^n_{i=1}, \{x^r_i\}^{n,\ell}_{i=1,r=1}, \{x^r_{i,j}\}^{n,n,\ell}_{i=1,j=1,r=1}) , \qquad (9.1)$$

i.e., it is all the inputs, all the random choices, all the broadcast messages and all the secret messages exchanged.

We can restrict a global transcript to a single party P_k, giving a local view

$$L_k = P_k(G) = (x_k, \rho_k, \{x_i^r, x_{i,k}^r\}_{i=1,i\neq k,r=1}^{n,\ell}) \ , \tag{9.2}$$

i.e., L_k consists of the input of P_k, the random choices of P_k, and all broadcast messages and all secret messages P_k received from other players. Note that from L_k, we can use π to compute (deterministically) all messages that P_k would send in an execution where its local view is L_k. Note also that the above way to generate a global transcript only makes sense if all parties follow the protocol. If some players are actively corrupted, the concept of a local view still makes sense and is well defined for honest parties, but the corrupted parties will of course not necessarily compute their messages according to π.

We say that two local views L_k, L_j are consistent if the messages P_k sends to P_j according to L_k are those that occur as received from P_k in L_j and vice versa. Accordingly, we define a predicate Consistent$(L_k, L_j) \in \{\top, \perp\}$, computed as in Algorithm CONSISTENT.

Algorithm CONSISTENT

1. The input is L_k, L_j, where $k \neq j$.
2. Use π to compute from L_k all messages P_k sends and extract those that P_j would see, namely $\{\tilde{x}_k^r, \tilde{x}_{j,k}^r\}_{r=1}^{\ell}$. If these messages differ from the corresponding messages $\{x_k^r, x_{j,k}^r\}_{r=1}^{\ell}$ occurring in L_j, output $\perp$ and stop.
3. Use π to compute from L_j all messages P_j sends and extract those that P_k would see, namely $\{\tilde{x}_j^r, \tilde{x}_{k,j}^r\}_{r=1}^{\ell}$. If these messages differ from the corresponding messages $\{x_j^r, x_{k,j}^r\}_{r=1}^{\ell}$ occurring in L_k, output $\perp$ and stop.
4. Output $\top$.

We now sketch how S and R will use protocol π: First S simulates a run of π on input $x_1 = x$ and give the output $y = f(x)$ of P_2 to R. In other words, S computes locally $(y, \ldots, y) = (y_1, \ldots, y_n) \leftarrow \pi(x, 0, \ldots, 0)$ and shows the outcome y to R. To convince R that y is the correct output, S will be asked to show the local view of a random selection of t of the parties in the protocol π. The only restriction is that R does not get to see the view of P_1. Note that it is secure to show the view of t parties to R as the protocol is secure against t corrupted parties, so seeing the view of t parties will not leak the input x of P_1.

To avoid that a cheating S modifies the views depending on who R wants to inspect, we first ask S to commit to all the local views in the protocol. Then R picks who to inspect, and S must open the corresponding commitments. If the views of the t inspected parties are all consistent, then except with very small probability the views of at least $n-t$ of the parties were all consistent, as we shall see below. But if this is the case, then the trace of the protocol that has been committed is one that could actually occur in real life, where at most t parties were corrupt. This means that all the consistent parties must have computed the correct output, as the protocol can tolerate up to t corrupted parties! The details

Protocol Zero-Knowledge Proof of Knowledge via MPC in the Head

The protocol is for two parties S and R. It runs in the hybrid model with access to an ideal functionality COM for commitment to strings.

1. First S runs π on input $x_1 = x$:
   1. Run π on $(x_1, \ldots, x_n) = (x, 0, \ldots, 0)$ as in Fig. 9.1, defining a global transcript G as in Eq. 9.1.
   2. Commit to $\mathsf{P}_i(G)$ for $i = 1 \ldots n$, using one call to the ideal functionality for each of the n local views. This means the local views can be individually opened.
2. R waits until the commitment functionality reports that all commitments were done.
3. Then R asks to see the local views of t random parties:
   1. Pick a uniformly random subset $E \subset \{1, \ldots, n\}$ with $|E| = t$ and $1 \notin E$.
   2. Send E to S
4. S reveals the local views of the t chosen parties:
   1. For $i \in E$, open the commitment to $P_i(G)$.
5. Then R checks that the revealed values are consistent with the parties P_i having followed the protocol π: For any pair $k, j \in E$, let L_k, L_j be the views opened in the previous step. If Consistent$(L_k, L_j) = \perp$ then terminate the protocol.
6. Check that all inspected players output the same value y, i.e., for all $i \in E$, check that the output value $y_{i,i}^{\ell}$ in L_i equals y.
7. If all the above checks go through, then R outputs y.

are given in Protocol Zero-Knowledge Proof of Knowledge via MPC in the Head. For the protocol to be secure, we need that the underlying protocol is private and robust as defined now.

9.2.1 Perfect Privacy of MPC

We define privacy simply by specializing the definition from Chapter 3.

Definition 9.1 (Perfect Privacy) *We say that π is a perfectly t-private implementation of g if there exists a poly-time function* Sim *such that the following condition holds for all input vectors $(x_1, \ldots, x_n)$. Compute $(y_1, \ldots, y_n) = g(x_1, \ldots, x_n)$ and execute π on inputs $(x_1, \ldots, x_n)$ as in Fig. 9.1. Then for all $C \subset \{1, \ldots, n\}$ with $|C| \leq t$ it holds that*

$$\mathrm{Sim}(\{x_j, y_j\}_{i \in C}) \overset{\text{perf}}{\equiv} \{P_j(G)\}_{j \in C} \ .$$

It follows in general that a protocol which is perfectly secure in the sense the we proved π to be in Chapter 5 is also t-private in the above sense for $t < n/3$.

9.2.2 Perfect Robustness of MPC

We say that a protocol π is perfectly t-robust if it always computes the correct result even if t parties deviated from the protocol. The correct result is defined by requiring that from the view of just the honest parties we can extract a well-defined input for also the corrupt parties, and the correct result is then defined based on these inputs. Note that this notion of robustness is not sufficient for or equivalent to security in general, we will have more to say about this below. The notion will, however, suffice for our application here.

Recall that the output of a party is just the message it sends to itself in the last round. We write

$$\text{Output}(L_i) = y^{\ell}_{i,i} \ .$$

where L_i is the local view of an honest party P_i. We use $\text{Input}(L_i)$ to denote the input of P_i.

Definition 9.2 (Perfect Robustness) *We say that π is a perfectly t-robust implementation of g if there exists a poly-time function* Inputs *returning values of the form $(x_1, \ldots, x_n)$ satisfying the following: Let $L_H = \{L_i | \mathsf{P}_i \in H\}$ be any set of local views that may result from an execution of π where a set H of at least $n - t$ parties are honest. Set*

$$(x_1, \ldots, x_n) = \text{Inputs}(L_H) \ ,$$
$$(y_1, \ldots, y_n) = g(x_1, \ldots, x_n) \ .$$

Then the following two conditions hold:

1. $x_i = \text{Input}(L_i)$ *for* $\mathsf{P}_i \in H$*,*
2. $\text{Output}(L_i) = y_i$ *for* $\mathsf{P}_i \in H$*,*

As mentioned, this notion of robustness tailored to our application in this section, and is in fact incomparable to standard notions: On one hand, it is not sufficient for security in general, as it might allow the extracted inputs of the corrupted parties to depend on the inputs of the honest parties. On the other hand, it does not follow in general that a protocol which is perfectly secure in the sense we proved for π Chapter 5 is also t-robust in the above sense.

The reason is that the above definition asks that the inputs of inconsistent parties can be computed from just the transcript of the consistent parties. When proving security in the sense that we did in Chapter 5 the simulator potentially has more power, in that it is the simulator itself which *produces* the transcript of the consistent parties—it might even simulate in a way which does not involve explicitly defining these transcripts.

However, it can easily be verified that the protocol π produced in Chapter 5 is indeed t-robust in the above sense, for $t < n/3$. This is because the way we extracted an input x_i for a corrupted P_i in Chapter 5 was to take the shares of x_i seen by the honest parties and from these we interpolated x_i. For this we needed only $n - t$ shares, so this way to extract inputs is exactly of the form required by the above definition. Then these inputs were submitted to the ideal functionality, which computed $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, where x_i for the honest P_i was the value sent directly to the ideal functionality by the environment. Then the ideal functionality sent y_i to each honest P_i.

9.2.3 Analysis

We now proceed to prove that Protocol Zero-Knowledge Proof of Knowledge via MPC in the Head is secure. We will only prove static security, but the protocol is in fact adaptive secure. Simulating the case where both parties are honest is simple: if we run the protocol over a secure channel there is no view to simulate. It therefore suffices to describe the simulators for the case where S respectively R are a corrupted. To simulate when S is corrupted, we use $\bar{\mathsf{S}}$. To simulate when R is corrupted, we use $\bar{\mathsf{R}}$.

Agent $\bar{\mathrm{R}}$

To simulate against a corrupted receiver, run as follows:

1. The simulator receives $y = f(x)$ from $\mathtt{PC}^f$, but does not know x.
2. The simulator (who is simulating $\mathtt{COM}$) commits to n dummy values on $\mathtt{COM}$.
3. Receive E from R and check that $|E| = t$ and $1 \notin E$.
4. Then simulate the local transcripts to be revealed:
   1. For $i \in E$, let $x_i = 0$.
   2. Let $y_1 = y$.
   3. For $i \in E \setminus \{1\}$, let $y_i = 0$.
   4. Use the simulator Sim from Definition 9.1 to sample

$$\{P_j(G)\}_{j\in E} \leftarrow \mathrm{Sim}(\{x_j, y_j\}_{i\in E}) \ .$$

5. For $j \in E$, change the values stored in $\mathtt{COM}$ such that the commitments are consistent with the values computed in the above step for $j \in E$. This goes unnoticed by the environment as it is the simulator who simulates $\mathtt{COM}$ and so far no commitments were opened.
6. For $j \in E$, open the commitments to the $P_j(G)$-values we just prepared.

It easily follows from t-privacy that the view produced by $\bar{\mathsf{R}}$ is identical to the view in the protocol.

To analyze $\bar{\mathsf{S}}$, consider first the case where the set T of players constructed by $\bar{\mathsf{S}}$ has size less than $n - t$. We claim that R would abort with overwhelming probability in such a case. To see this, note that at least $t/2$ pairs of players have been deleted from the full set of players to obtain T. If any such pair is in the set of t players checked by R, (s)he will abort.

Let t' be the number of players not in T, then we know that $t' \geq t$. let us assume that $t/n = c$ for a constant c. Then $t'/n \geq c$ so we expect that at least ct among the t players checked by R are not in T. In fact, by a standard Chernoff bound, except with negligible probability (as a function of n), there will be at least $c't$ such players for $c' < c$. Now if selecting the first $c't/2$ players not in T misses all inconsistent pairs, this means that the "partners" of these first players have not been selected. But then, each time a new player is selected after this point, an inconsistent pair is found with at least constant probability $c't/(2n) = c'/(2c)$. Overall, R finds an inconsistent pair except with negligible

Agent $\bar{\mathsf{S}}$

To simulate against a corrupted sender, run as follows:

1. Simply simulate the protocol honestly, by running as the honest R in Protocol Zero-Knowledge Proof of Knowledge via MPC in the Head. This is possible as R has no input.
2. The only challenge is that if during the above simulation R produces an output y. In that case the simulator needs to make the ideal functionality $\mathtt{PC}^f$ output y on behalf of R. This can only be done by inputting some $x \in f^{-1}(y)$ to $\mathtt{PC}^f$ on behalf of the corrupted S. This x is computed as follows:
   1. If R outputs y in the protocol, then S already committed to a set of local views. Inspect $\mathtt{COM}$ to extract these values $L_1, \dots, L_n$ (this is possible as it is the simulator who simulates $\mathtt{COM}$).
   2. Construct a set T of players as follows: initially let T be the set of all players. Now, if there exists a pair of players $\mathsf{P}_i, \mathsf{P}_j \in T$ for which L_i, L_j are not consistent, i.e., $\text{Consistent}(L_i, L_j) = \perp$, then remove P_i and P_j from T. Continue this way until no more pairs can be removed.
   3. If $|T| < n - t$, set $x = 0$. Otherwise, compute $(x_1, \dots, x_n) = \text{Inputs}(\{L_i\}_{i \in T})$, and use $x = x_1$.

probability in n. Therefore when T is smaller than $n - t$, it does not matter that $\bar{\mathsf{S}}$ does not compute the correct x.

Then we consider the case where T has size at least $n - t$. Since the local views of players in T are all pairwise consistent, this collection of local views could have resulted from a real execution of π, indeed, consider an environment that corrupts the players not in T and instructs them to send exactly those messages to players in T than occur in their local views. It therefore follows from t-robustness that evaluating the Inputs-function results in an input x for P_1 such that all honest players output $y = f(x)$. Since R inspects t players he will inspect at least one honest player except with negligible probability and will therefore output y or reject. Hence, in this case, $\bar{\mathsf{S}}$ indeed computes the correct value of x.

We have proved:

Theorem 9.3 *Consider functionality $\mathtt{PC}^f$ and construct a function g with n outputs from f as described above. Suppose protocol π is a perfect t-private and t-robust implementation of g, where $n = ct$ for a constant c. Then Protocol* Zero-Knowledge Proof of Knowledge via MPC in the Head *composed with functionality $\mathtt{COM}$ implements $\mathtt{PC}^f$ with statistical security.*

Implementation and Efficiency Considerations

We note first that protocols that can be used as π in the above Theorem do exist. Indeed, we have seen protocols that can evaluate any function with perfect active security assuming $t < n/3$. Such a protocol more than satisfies the conditions. However, it is clear that the resulting two-player protocol becomes more efficient the more efficient π is.

To get a protocol that implements $\mathtt{PC}^f$ from scratch, we could attempt to implement COM, but this would require the implementation to be UC-secure and hence cannot be done without some setup assumption (see Chapter 7). An approach leading to better efficiency that does not require setup assumptions is to use a non-UC commitment scheme and prove directly that that the composed protocol is a zero-knowledge proof that the prover knows x such that $y = f(x)$. The standard definitions of this (that are out of scope for this book) do not require universal composability, so in a sense this allows to trade composability for efficiency.

This is important, as the main motivation for the MPC-in-the-head approach is indeed efficiency – it was already well known that very general Zero-Knowledge proofs follow from commitment schemes, but with MPC-in-the-head we can make the proofs smaller. This improvement comes from several sources and we sketch here what the main ideas are:

- In Protocol Zero-Knowledge Proof of Knowledge via MPC in the Head, only n commitments need to be produced. Now, except for a small additive overhead that depends on the security parameter, commitments can be implemented such that a commitment has size equal to the string committed to. Therefore the communication needed in the 2-player protocol is essentially the communication complexity of π.
- The communication complexity of π can be optimized using some of the techniques we saw in Chapter 8. In particular, we saw that using packed secret-sharing, we can implement many parallel instances of the same computation almost "for the price of one". This is very useful in connection with zero-knowledge: here we need to evaluate the function $f_R(x, w)$ for NP relation R as defined above. Imagine that we write down a Boolean circuit that evaluates f_R. Now, the prover (who knows x, w) can compute the inputs and outputs for that occur for every gate in the circuit when we input x, w. She can then supply these as extra input to the computation. Now, all π needs to do is to check that the claimed output from every gate is the correct function of its inputs, and this can be done for all gates in parallel.
- Packed secret-sharing can be done based on Shamir's scheme as we have seen, but the price is that the size of field we use must grow linearly with the number of players n as each player must have his own evaluation point. As a result we get an overhead factor of $\log(n)$. This can be avoided using asymptotically good strongly multiplicative secret sharing schemes, see Chapter 12. The overhead disappears because those schemes can be constructed using a constant size field.

Using all the above ideas, one can get zero-knowledge proofs for NP-relations R with communication complexity linear in a circuit computing f_R. It is even possible to construct two-party protocols for secure function evaluation using the MPC-in-the-head approach (see the Notes section for details). This further emphasizes the interesting fact that it is important to study secure protocols for many players, even if the final goal is two-party protocols!

9.3 Notes

The sugar beets double auction was developed as part of two research projects SCET (Secure Computing, Economy and Trust) and SIMAP (Secure Information Management and Processing) carried out at Aarhus University. These projects aimed at improving the efficiency of SMC, with an explicit focus on a range of economic applications, which were believed to be particularly interesting for practical use. At the time of writing the auction is still run once a year by the spin-off company Partisia Market Design.

The final implementation of the auction was reported on in [29]. The algorithmic ideas behind the secure comparison that we describe here are based on [75].

The general idea of designing 2-party protocols from multiparty computation using the MPC-in-the-head approach is from [116], and the construction we give here is also from that paper. The proof is slightly different as we prove the 2-party protocol to be a proof of knowledge where the original proof only shows it is a proof of language membership. In [117] it is shown how design secure 2-party computation from multiparty protocols.

Part II

Secret Sharing

10

Algebraic Preliminaries

Contents

10.1 Introduction

Throughout, the reader is assumed to already have a good (working) knowledge of basic undergraduate algebra. [1] For ease of reference, we shall first recall some central definitions and useful elementary results concerning groups, rings, fields, modules, and vector spaces. For additional basic theory, full details and more, please refer e.g. to Serge Lang's *Algebra* [126]. [2]

Furthermore, at some points we shall need some results from field theory and algebraic number theory as well as from multi-linear algebra, specifically concerning cyclotomic number fields and tensor products. In each of these cases we shall give a brief introduction and state the required results. For full details and more, please refer to [126, 127, 125]. Our exposition of these topics (loosely) follows Lang [126].

Finally, we shall need results from the theory of algebraic function fields over finite fields, i.e., algebraic curves over finite fields.[3] This is deferred to Section 12.7. Specifically, the focus there will be on families of curves with asymptotically many rational points. We will give a bird's eye introduction to this topic that is self-contained in that it assumes as background only the material on basic algebra covered earlier on. For a full treatment of the basic definitions and theory of algebraic function fields, as well as such results as the ones referred to above, we refer to Henning Stichtenoth's *Algebraic Function Fields and Codes* [169]. [4] Except when stated differently, our exposition follows [169], specifically parts of Chapters 1, 3, 5, and 7. Proofs are mostly beyond the scope in this introduction. Yet we shall state all results needed for our purposes.

Basic definitions and results that are covered in full detail by most standard introductory textbooks are mentioned without reference (but sometimes a proof is given). In case we suspect that a result we quote is not so easy to look up elsewhere in full detail (e.g., because it is usually disguised as a special case of more advanced theory or hidden in exercises), we give an explicit reference to a suitable text.

[1] In particular, our goal here is to refresh knowledge, to quickly expand it in some more advanced directions relevant to our main cryptographic topic and to issue an invitation to more in-depth study from specialized sources, for instance those referenced here

[2] Lang's classical book [126] is a standard reference work for basic algebra. See also e.g. [127], which is of introductory nature.

[3] More precisely, projective smooth absolutely irreducible algebraic curves over finite fields.

[4] Stichtenoth's book is one of a few that makes a number of important concepts and (modern) developments on algebraic function fields over finite fields rather quickly accessible to readers with a firm command of basic commutative algebra (as covered for instance by parts of Lang's book [126]). Some prior exposure to Galois theory and algebraic number theory is useful but not strictly necessary. It also features an in-depth treatment of asymptotically good towers of algebraic function fields over finite fields. See also Niederreiter and Xing [144], Moreno [141] and Tsfasman, Vladuts and Nogin [173].

In a few places in the text we will give appropriate references when discussing certain specific (advanced) algorithmic aspects. For an introduction to computational aspects of number theory and algebra, the reader is referred to Shoup [164] and for an introduction to computational algebraic number theory, the reader is referred to Cohen [59].

10.2 Groups, Rings, Fields

10.2.1 Monoids

A *monoid* is a set G together with a map $\bullet : G \times G \longrightarrow G$, a *law of composition*, such that

1. for all $a, b, c \in G$, it holds that $a \bullet (b \bullet c) = (a \bullet b) \bullet c$ (*associativity*).
2. there is $e \in G$ such that, for all $a \in G$, it holds that $e \bullet a = a \bullet e = a$ (*neutral element*).

The neutral element e (sometimes denoted e_G) is unique: if e' is another neutral element, then $e = e \bullet e' = e'$. A monoid is *commutative* if $a \bullet b = b \bullet a$ for all $a, b \in G$.

A *submonoid* of G is a subset G' such that the neutral element e is contained in G' and G' is closed under the composition. The restricted composition makes it a monoid. In what follows we write ab instead of $a \bullet b$.

Let G, H be monoids. A *morphism* from G to H is a function $\phi : G \longrightarrow H$ such that $\phi(e_G) = e_H$ and $\phi(ab) = \phi(a)\phi(b)$ for all $a, b \in G$. Note that the composition on the l.h.s. (resp., r.h.s.) corresponds to the one in G (resp., in H). If $G = H$, we speak of an *endomorphism.*

A morphism $\phi : G \to H$ of monoids is an *isomorphism* if there is a morphism $\phi' : H \to G$ such that $\phi' \circ \phi$ is the identity on G and $\phi \circ \phi'$ is the identity on H. Here $\circ$ denotes composition as functions. A morphism is an isomorphism if and only if it is bijective [5]. If $G = H$, we speak of an *automorphism.*

10.2.2 Groups

A *group* is a monoid G such that, for each $a \in G$, there is $a' \in G$ with $aa' = a'a = e_G$ (an *inverse*). For each $a \in G$ its inverse is unique. A *subgroup* is a subset H such that it contains the neutral element, it is closed under the composition, and it is closed under taking inverses. The restricted composition makes H a group. Equivalently, H is a submonoid that is closed under taking inverses.

An *abelian group* is a group whose composition is commutative (i.e., as a monoid it is commutative). If G is abelian, it is common to use additive notation (i.e., $+$) for its composition. The neutral element is then denoted as 0_G (or simply as 0) and the inverse of $a \in G$ as $-a$. In particular, for all $a, b \in G$, it holds that

[5] i.e., both injective and surjective.

$a+b=b+a$ and $a-a:=a+(-a)=0$. If G is a nonabelian group, it is common to use multiplicative notation (i.e., $\cdot$) for its composition. The neutral element is then denoted as 1_G (or simply as 1) and the inverse of $a \in G$ as a^{-1} (or $1/a$). If it is not specified whether G is abelian or not, the notation pertaining to the nonabelian case is used.

A group G is *finite* if $|G| < \infty$. If G is a finite group and $x \in G$, then $\operatorname{ord}(x)$ is the smallest positive integer m such that $x^m = 1$. This is well-defined. In fact, it holds that $\operatorname{ord}(x)$ divides $|G|$. Moreover, if $p > 0$ is a prime number such that p divides $|G|$, then there is $x \in G$ with $\operatorname{ord}(x) = p$. A group is *cyclic* if there is $x \in G$ such that, for each $y \in G$, it holds that $x^d = y$ for some integer d. In that case, the element x is a *generator*.

For example, the integers $\mathbb{Z}$ under addition form a group, as well as the set $\{-1, 1\} \subset \mathbb{Z}$ under multiplication. Also, $\mathbb{Z}/n\mathbb{Z}$, the integers modulo n (n an integer), forms a group under addition. [6] If G is a monoid, then the set consisting of its automorphisms is a group under functional composition and its neutral element is the identity morphism.

A *morphism of groups* is just a morphism of the corresponding monoids. However, it is no longer necessary to require $\phi(1_G) = 1_H$ of a group morphism $\phi : G \longrightarrow H$, as this is now implied: $\phi(1_G) = \phi(1_G 1_G) = \phi(1_G)\phi(1_G)$ and left-multiplication by $\phi(1_G)^{-1} \in H$ on the l.h.s. as well as on the r.h.s. gives the desired result.

10.2.3 Rings

A *ring* is a set R with a law of addition $+ : R \times R \longrightarrow R$ and a law of multiplication $\cdot : R \times R \longrightarrow R$ such that the following holds.

1. It is an abelian group with respect to addition.
2. It is a monoid with respect to multiplication, i.e., multiplication is associative and there is a neutral element. [7]
3. $a(b+c) = ab + ac$ and $(a+b)c = ac + bc$ for all $a, b, c \in R$ (*distributivity*).

The neutral element for multiplication is denoted 1_R (or simply 1). Note that multiplication is not necessarily commutative. Moreover, it is not required that $0 \neq 1$. However, it holds that $0 = 1$ if and only if R is the *trivial ring*, i.e., R consists of a single element 0 $(= 1)$ only. [8]

Let R be a ring. The ring R is *commutative* if $ab = ba$ for all $a, b \in R$ (i.e., the multiplicative monoid is commutative). A *subring* is a subset $R' \subset R$ that is closed under addition and closed under multiplication such that the restricted compositions give an additive subgroup and a multiplicative submonoid, respectively.

[6] If $n = 0$ this is $\mathbb{Z}$ itself and if $n = 1$, this is the trivial group.

[7] As to multiplication in R, it is often more convenient to write ab instead of $a \cdot b$.

[8] In the forward direction: if $0 = 1$, then $0 = 0a = 1a = a$ for all $a \in R$. The statement in the other direction is trivial.

Under the restricted operations, a subring is a ring. Note that, by definition, $1 \in R'$ since R' is, in particular, a multiplicative submonoid of R. If R' is a subring of R, we also say equivalently that R is an *extension ring* of R'. An element $u \in R$ is a *unit* if there is $u' \in R$ such that $uu' = u'u = 1$. The units in a ring form a group under multiplication.

A *morphism of rings* is a function that is both a morphism of the corresponding addition-monoid and a morphism of the corresponding multiplication-monoid. Equivalently, $\phi : R \longrightarrow R'$ is a ring morphism if it holds that $\phi(a+b) = \phi(a)+\phi(b)$ and $\phi(ab) = \phi(a)\phi(b)$ for all $a, b \in R$, and if $\phi(1_R) = 1_{R'}$. As noted before, it is not necessary to require $\phi(0_R) = 0_{R'}$ separately.

For example, there is a unique ring morphism ϕ from $\mathbb{Z}$ to R. Let $a \in \mathbb{Z}$. If $a \geq 0$, define $\phi(a) = 1_R + \cdots + 1_R$ (a times). By convention, the empty sum yields 0_R and thus $\phi(0) = 0_R$. If $a < 0$, define $\phi(a) = (-1_R) + \cdots + (-1_R)$ ($-a$ times). This is a ring morphism. By definition, any ring morphism from $\mathbb{Z}$ to R maps 0 to 0_R and 1 to 1_R and this uniquely determines the morphism.

Throughout this chapter, let A be a commutative ring.

An *ideal* of A is a subgroup I under addition that is closed under multiplication by scalars from A, i.e., if $a \in A$ and $x \in I$, then $ax \in I$. Note that A is an ideal of A, as is the *null-ideal* consisting of 0 only. These are special cases of *principal ideals*. An ideal I of A is principal if there is $x \in A$ such that $I = xA := \{xa | a \in A\}$, i.e., it consists of all A-multiples of x. We also sometimes use the notation (x) instead of xA. In particular, $A = (1)$ and the null-ideal corresponds to (0). As another example, the ideal (2) of $\mathbb{Z}$ consists of all even integers.

Let I, J be ideals of A. Their *sum* is defined as

$$I + J = \{x + y \mid x \in I, y \in J\} \subset A,$$

which is an ideal of A. Their *product* is defined as

$$IJ = \{x_1y_1 + \cdots + x_my_m \mid m \geq 1, x_1, \ldots, x_m \in I, y_1, \ldots, y_m \in J\} \subset A,$$

which is also an ideal of A. Equivalently, IJ is the ideal generated by the products xy with $x \in I$ and $y \in J$. These definitions extend trivially to the case of more than two ideals.

The ideal I is a *prime ideal* if $I \neq A$ and if for all $x, y \in A$ it holds that $xy \in I$ implies $x \in I$ or $y \in I$. The ideal I is a *maximal* ideal if $I \neq A$ and if there is no ideal J of A with $I \subsetneq J \subsetneq A$. If I is a maximal ideal, then I is also a prime ideal. By an application of *Zorn's Lemma*, each nontrivial ring contains a maximal ideal. In fact, the stronger result holds that each proper ideal in a nontrivial ring is contained in some maximal ideal.

A *zero divisor* is an element $z \in A$ with $z \neq 0$ such that $zz' = 0$ for some $z' \in A$ with $z' \neq 0$. An element $x \in A$ is a *nilpotent* if there exists a positive integer m such that $x^m = 0$. The nilpotent elements form an ideal, the *nilradical*. The nilradical is the intersection of all prime ideals of A. [9]

The ring A is a *domain* if $0 \neq 1$ and if it has no zero-divisors, or equivalently, if the ideal (0) is a prime ideal. If A is a domain, $x, y \in A$ and $xy \neq 0$, then y *divides* x, denoted $y|x$, if there is $z \in A$ such that $yz = x$. Furthermore, if A is a domain then $x \in A$ ($x \neq 0$) is *irreducible* if it is not a unit and for all $y, z \in A$ it holds that if $x = yz$ then y is a unit or z is a unit. The ring A is *factorial* if it is a domain and it has "unique factorization". This means that if $x \in A$ is nonzero, then there exist irreducible elements $\pi_1, \ldots, \pi_\ell \in A$ and a unit $u \in A$ such that

$$x = u\pi_1 \cdots \pi_\ell.$$

This factorization is unique in the following sense. If $x = u'\pi'_1 \cdots \pi'_{\ell'}$ is another such factorization, then $\ell = \ell'$ and there are units $u_1, \ldots, u_\ell \in A$ such that, after applying a permutation, it holds that $\pi'_i = u_i\pi_i$ for $i = 1, \ldots, \ell$. If A is factorial, then an irreducible element is called a *prime*.

The ring A is a *principal ideal domain (PID)* if it is a domain and if each of its ideals is principal. Each PID is factorial. An example of a PID is the ring of integers $\mathbb{Z}$. All nonzero prime ideals in a PID are maximal and a nonzero ideal $J = (x)$ with $x \in A$ is a prime ideal if and only if $x \in A$ is a prime element. For example, if $A = \mathbb{Z}$, then the nonzero prime ideals (= maximal ideals) are the ones of the form (p) where p is prime.

By $A[X]$ we denote the commutative ring of polynomials with coefficients in A. The usual definition of the degree of a polynomial applies, but recall that the degree of the zero-polynomial $f = 0$ is $-\infty$ by definition. For any integer $m \geq 1$, by extension, $A[X_1, \ldots, X_m]$ denotes the commutative ring of m-variate polynomials with coefficients in A.

Finally, A is *Noetherian* if each of its ideals is finitely generated. In other words if I is an ideal of A, then there exist some $c_1, \ldots, c_k \in I$ such that each element of I can be expressed as an A-linear combination of $c_1, \ldots, c_k$.

10.2.4 Fields

Let A be a commutative ring. If $0 \neq 1$ and each of its nonzero elements is a unit, then A is a *field*. Note that a field has no zero divisors. The only ideals of a field are (0) (a maximal ideal) and (1), i.e., the whole field. A *subfield* is a subring that is a field. A *finite field* is one with finite cardinality (i.e., the set has finite cardinality). Recall that for each positive integer q of the form $q = p^m$, where $p \geq 2$ is a prime number and $m \geq 1$ is an integer, there exists a finite field of cardinality q. Any two given finite fields of the same cardinality are isomorphic. [10] For all other finite cardinalities q, there is no finite field with that cardinality. We typically denote a finite field with cardinality q by $\mathbb{F}_q$.

[9] See e.g. [126], Ch. X, par. 2, p. 417.

[10] However, the choice of an isomorphism between them is *not unique*.

Suppose K is a field and consider the intersection of all of its subfields, which is a field as well. If this is the field of rational numbers $\mathbb{Q}$, we say that the *characteristic* of K is 0. If it is a finite field $\mathbb{F}_p$ for some prime number $p > 0$, we say that the characteristic of K is p. This covers all the possibilities. Indeed, if $n \cdot 1 = 0$ for some integer $n \geq 2$, then the smallest such integer must be a prime number p; otherwise there would be zero divisors. Hence, $\mathbb{F}_p$ is the "smallest" subfield in this case. Otherwise, if there is no such an integer n, then the field contains $\mathbb{Z}$ and hence $\mathbb{Q}$ is the "smallest" subfield.

In the other direction, if K, L are subfields of some given field Ω, then KL denotes their *compositum*, i.e., the intersection of all subfields of Ω having both K and L as a subfield. [11]

It should be noted that a field K is a PID, [12] as its only ideals are (0) and (1). Moreover, $K[X]$ is a PID. The primes of $K[X]$ are the irreducible polynomials, i.e., the polynomials $P(X) \in K[X]$ of degree ≥ 1 such that for all $f(X), g(X) \in K[X]$ with $P(X) = f(X)g(X)$, it holds that $f(X)$ is constant or $g(X)$ is constant.

10.2.5 The Field of Fractions and Localization

If A is a domain, $Q(A)$ denotes the *field of fractions* of A, i.e., "the smallest field that contains A". Consider all pairs (a, b) with $a, b \in A$ and $b \neq 0$. Declare

$$(a, b) \sim (c, d) \text{ if } ad = bc.$$

This is an equivalence relation in the usual sense. The class of (a, b) is denoted a/b. The field of fractions $Q(A)$ is the set consisting of all these equivalence classes a/b, with addition and multiplication defined by $a/b \cdot c/d = ac/bd$ and $a/b + c/d = (ad + bc)/bd$. We write $1 := 1/1$ and $0 := 0/1$.

If P is a prime ideal of the domain A, the *localization of A at P* is the subring A_P of $Q(A)$ consisting of those elements $a/b \in Q(A)$ such that $b \notin P$. [13] This is an example of a *local ring*, a commutative ring with a unique maximal ideal, namely the ideal consisting of all $a/b \in Q(A)$ with $a \in P$ and $b \notin P$. Note that each nonzero element not contained in the maximal ideal is a unit, i.e., the units in A_P are those elements $a/b \in Q(A)$ such that $a, b \notin P$.

For example, let $p \in \mathbb{Z}$ be a prime number. Then $\mathbb{Z}_{(p)}$ is the subring of $\mathbb{Q}$ consisting of those $a/b \in \mathbb{Q}$ such that p does not divide b. Its unique maximal ideal consists of all elements of the form $a/b \in Q(A)$ such that p divides a and p does not divide b. The units in $\mathbb{Z}_{(p)}$ are those elements $a/b \in \mathbb{Q}$ such that p divides neither b nor a. Note that, moreover, $\mathbb{Z}_{(p)}$ is a PID: if I is a nonzero ideal,

[11] From a more constructive perspective, the compositum consists of all elements $x \cdot y^{-1}$ where both x and y are finite sums of terms of the form ab with $a \in K$ and $b \in L$ and where $y \neq 0$.

[12] But the implied unique factorization doesn't mean much in the case of a field, as all its nonzero elements are units.

[13] Since A is a domain, the ideal $P = (0)$ is prime. Thus, the localization at (0) is well-defined and it coincides with $Q(A)$.

then $I = (p^m)$ for some nonnegative integer m. As an aside, each nonzero element $z \in \mathbb{Q}$ can be uniquely expressed as $z = up^m$ with $m \in \mathbb{Z}$ and u a unit in $\mathbb{Z}_{(p)}$. Localization can also be defined when A is not a domain, but we will not need it and do not treat it here.

10.3 Modules and Vector Spaces

We now give the definition of modules and vector spaces.

10.3.1 Basic Definitions

We give a concise definition of a module that avoids a "long list" of axioms. It gives the user a good vantage point, especially in situations where showing the module property would otherwise come across as "complicated" as a result of the "gravitational pull" of such listings.

First we define the endomorphism ring of an abelian group G. The set of endomorphisms $\mathrm{End}(G)$ has a natural ring structure, as follows. Its 0-element corresponds to the function that sends each element of G to 0_G and its 1-element is the identity function on G. Let $\phi, \phi' \in \mathrm{End}(G)$. The endomorphism $\phi + \phi' \in \mathrm{End}(G)$ is defined by $(\phi+\phi')(a) := \phi(a)+\phi'(a)$ for all $a \in G$. The endomorphism $\phi \cdot \phi' \in \mathrm{End}(G)$ is defined by $(\phi \cdot \phi')(a) := \phi(\phi'(a))$ for each $a \in G$. Thus, multiplication is given by functional composition. Note that $-\phi \in \mathrm{End}(G)$, the additive inverse of ϕ, is given by the rule $(-\phi)(a) := \phi(-a) = -\phi(a)$ for each $a \in G$. It is easy to check that this gives indeed a ring structure on $\mathrm{End}(G)$.

An *R-module M* consists of an abelian group M, a ring R, and a ring morphism

$$\psi : R \longrightarrow \mathrm{End}(M).$$

In particular, the (left-) multiplication rm of a group element $m \in M$ by a scalar $r \in R$ is defined as the application to m of the endomorphism corresponding to r. Formally, $rm := \psi_r(m)$, where $\psi_r := \psi(r) \in \mathrm{End}(M)$. A *submodule* is a subgroup M' under addition that is closed under (left-) multiplication by scalars from R. Define

$$\psi' : R \longrightarrow \mathrm{End}(M')$$

such that, for each $r \in R$, its image ψ'_r under this map equals the restriction of ψ_r to M'. This is indeed an element of $\mathrm{End}(M')$; since M' is a submodule, it holds that $\psi_r(M') \subset M'$, for each $r \in R$. Under this restriction M' is a module. A *morphism* $\phi : M \longrightarrow N$ of R-modules is a morphism of the additive groups that satisfies $\phi(rm) = r\phi(m)$ for all $r \in R$, $m \in M$.

For example, let G be an abelian group. Then G is a $\mathbb{Z}$-module in a natural (and unique) way. As remarked earlier, for any given ring R, there is a unique ring morphism from $\mathbb{Z}$ to R. Now simply take $R = \mathrm{End}(G)$; the element $\lambda \in \mathbb{Z}$ is sent to the multiplication-by-λ-map on G.

As another example, if R' is a subring of a ring R, then R is an R'-module. Write R^+ for the additive group of R and simply define the ring morphism $\phi : R' \longrightarrow \mathrm{End}(R^+)$ by sending $r' \in R'$ to the multiplication-by-r'-map, which is clearly an endomorphism of R^+. In particular, R is an R-module. Finally, an ideal I of a commutative ring A is nothing else than a submodule of the A-module A.

If A is a commutative ring and M is an A-module, then M is *Noetherian* if each of its submodules is generated by a finite number of elements. In other words, if N is a submodule then there are some elements $c_1, \ldots, c_k \in N$ such that each element of N can be written as an A-linear combination of $c_1, \ldots, c_k$. So A is Noetherian as a ring if and only if A is Noetherian as an A-module.

If K is a field and V is a K-module, we speak of a *K-vector space V*. Also, in the case of K-vector spaces, we speak of *K-vector space morphisms* (also known as *linear maps*).

We will also use the notion of the *dual* of a vector space. Let K be a field and let V be a finite dimensional K-vector space. Then V^* is the K-vector space consisting of all K-vector space morphisms $\phi : V \longrightarrow K$ (the space of K-linear forms on V). It is isomorphic to V. For instance, suppose $V = K^n$ for some positive integer n. Let $\langle \cdot, \cdot \rangle$ denote the standard inner product on K^n, i.e.,

$$\langle \mathbf{x}, \mathbf{y} \rangle = x_1 y_1 + \ldots + x_n y_n$$

for all $\mathbf{x} = (x_i), \mathbf{y} = (y_i) \in K^n$. [14] Then an isomorphism between V and V^* is given by the map

$$\phi : V \longrightarrow V^*$$

$$\mathbf{a} \mapsto \langle \mathbf{a}, \cdot \rangle,$$

i.e., the vector $\mathbf{a} \in V$ is mapped to the K-linear form that consists of taking the inner product with $\mathbf{a}$.

10.3.2 Free Modules and Bases

Let R be a ring and let M be an R-module. Suppose $S \subset M$ is a subset. Then S is a *basis* if

1. S generates M over R, i.e., each element of M can be written as an R-linear combination of a finite number of elements of S.
2. S is R-linearly independent, i.e., if some R-linear combination of finitely many distinct elements of S yields 0, then each of the R-coefficients equals 0.

We allow $S = \emptyset$ as "the basis of the trivial module." This simplifies certain statements where otherwise an exception for the trivial module is made. Note that if $S \neq \emptyset$ is a basis, then M is not the trivial module if R is not the trivial ring.

[14] Note that if K has positive characteristic then $\langle \mathbf{x}, \mathbf{x} \rangle = 0$ for $\mathbf{x} \in K^n$ with $\mathbf{x} \neq \mathbf{0}$ is possible.

A *free R-module* M is one that admits a basis. Note that if M is free and a basis S is given, then each element of M is uniquely expressed as an R-linear combination of finitely many elements of S.

10.3.3 Free Modules over PIDs

Let A be a nontrivial commutative ring. Then the cardinality of any two bases of a free A-module M is the same. As an aside, this is easy to reduce to the case of vector spaces, for which this is a well-known fact. We argue this below. For quotients, please refer to Section 10.4. Choose some maximal ideal P of A. Let PM denote the submodule of M generated by the terms of the form rm with $r \in P$ and $m \in M$. Observe that M/PM is an A/P-vector space and that a basis of the A-module M, after reduction modulo PM, is a basis of the A/P-vector space M/PM; the claim follows.

The *rank* of M is the cardinality of any basis. If the rank is infinite, then a basis involves infinitely many elements. Yet each element of M is expressed as an A-linear combination of finitely many of them. If $A = K$ for some field K, we speak of the *dimension* of the K-vector space M. It is denoted by $\dim_K M$.

If A is a PID, more is true. If M' is a submodule of a free A-module M, then it is free as well and its rank is at most that of M. If $A = K$ for some field K, $M' \subsetneq M$ is a subspace and the dimension of M is finite, then the dimension of M' is strictly smaller than that of M. If A is a PID that is not a field, the rank of M' may equal that of M, yet $M' \subsetneq M$. For example, $\mathbb{Z}^n$ has rank n as a free $\mathbb{Z}$-module and $2 \cdot \mathbb{Z}^n$ is a free submodule of rank n as well, yet $2 \cdot \mathbb{Z}^n \subsetneq \mathbb{Z}^n$. Notable examples of PID's include, $A = \mathbb{Z}, K[X], K$ (K a field). It is customary to speak of dimension if the PID in question is a field and to speak of rank if it is not a field. If the PID is unspecified, we will speak of rank.

As an example of a vector space, if n is a positive integer then K^n is a K-vector space of dimension n, with the usual vector addition and scalar multiplication. As another example, if a ring R contains a field K as a subring, then R is a K-vector space. If L is a field that contains K as a subfield, then we say that L is an *extension field* of K. The *degree* of the extension, denoted $[L : K]$, is the dimension of L as a K-vector space.

The main result on free modules of finite rank over PID's is as follows. [15]

THEOREM 10.1 *Let A be a PID. Let M be a free A-module of rank $n \geq 1$ and let M' be a submodule. Then M' is a free A-submodule and it has rank ℓ with $0 \leq \ell \leq n$. Moreover, if M' is nontrivial then there is an A-basis $e_1, \ldots, e_n$ of M and there are nonzero elements $a_1, \ldots, a_\ell \in A$ such that*

- *$a_1e_1, \ldots, a_\ell e_\ell$ is an A-basis of M'.*
- *$a_i \mid a_{i+1}$ $(i = 1, \ldots, \ell - 1)$, for $i = 1, \ldots, \ell - 1$.*

[15] For a proof, see e.g. [126], pp. 153–154.

The ideals $(a_1), \ldots, (a_\ell)$ *of* A *are uniquely determined by* M *and* M'.

10.4 Quotients

If $\phi : X \longrightarrow Y$ is a morphism of groups (rings/modules/vector spaces), the *image* of ϕ, denoted imϕ, is the set $\phi(X) \subset Y$. The *kernel* of ϕ, denoted kerϕ, is the set $\phi^{-1}(0_Y) \subset X$, i.e., the set of elements in X that are sent to 0_Y by ϕ. A morphism $\phi : X \longrightarrow Y$ is an isomorphism if there is a morphism $\phi' : Y \longrightarrow X$ such that $\phi' \circ \phi$ is the identity morphism on X and $\phi \circ \phi'$ is the identity morphism on Y. In each of the mentioned cases, a morphism is an isomorphism if and only if it is bijective. The notation $X \simeq Y$ means that X and Y are isomorphic, i.e., there is an isomorphism $X \longrightarrow Y$.

We now define quotient groups and quotient rings. Let G be an abelian group and let H be a subgroup. Then G/H denotes the *quotient group* consisting of the classes $\overline{a} := \{a + H\}$ with $a \in G$. This is an abelian group with addition defined by $\overline{a} + \overline{b} = \overline{a + b}$ for all $a, b \in G$. The definition of this operation does not depend on the choice of the class representatives a, b. We shall not need this quotient notion in the case of nonabelian groups or noncommutative rings. [16] Recall that if $\phi : G \longrightarrow G'$ is a group morphism, then kerϕ is a subgroup [17] of G and imϕ is a subgroup of G'. Moreover, the assignment $a + \ker\phi \mapsto \phi(a)$ defines a group isomorphism $G/\ker\phi \longrightarrow \mathrm{im}\phi$. In particular, if ϕ is surjective, then $G/\ker\phi \simeq G'$ as groups.

As before, let A be a commutative ring. Let I be an ideal of A. Considering A as an abelian group and I as a subgroup, the quotient A/I is formed, which is an abelian group by the discussion above. By virtue of the fact that I is an ideal, it holds that $\overline{a}\overline{b} = \overline{ab}$ for all $a, b \in A$, independently of the class representatives a, b. This turns A/I into a commutative ring. Recall that A/I is a domain if and only if I is a prime ideal and A/I is a field if and only if I is a maximal ideal.

For example, consider the ring of integers $\mathbb{Z}$. As remarked before, this is a PID. Each of its ideals is of the form $n\mathbb{Z}$ for some integer $n \geq 0$. Furthermore, $\mathbb{Z}/n\mathbb{Z}$ is the "ring of integers modulo n", which has cardinality n if $n \neq 0$. The maximal ideals of $\mathbb{Z}$ are of the form $p\mathbb{Z}$, with $p > 0$ a prime number. Hence, $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$ is the field of cardinality p.[18] The polynomial ring $\mathbb{F}_p[X]$ is a PID. If $f(X) \in \mathbb{F}_p[X]$ is an irreducible polynomial of degree $d \geq 1$, then $\mathbb{F}_p[X]/(f(X))$ is the finite field with $q := p^d$ elements. Recall that every finite field may be constructed in this way. Note that, in particular, $\mathbb{F}_q$ is an extension field of $\mathbb{F}_p$ of degree d.

[16] It suffices to recall that if G is a nonabelian group, then the construction of the quotient group G/H works in essentially the same way as in the abelian case if the additional condition holds that the subgroup H in question is *normal*, i.e., $gHg^{-1} \subset H$ for all $g \in G$. In the abelian case, normal subgroups and subgroups coincide.

[17] In fact, kerϕ is a normal subgroup of G whether G is abelian or not.

[18] In the cryptographic literature the notation $\mathbb{Z}_p$ is sometimes used instead of $\mathbb{Z}/p\mathbb{Z}$. We adhere to the latter, as the notation $\mathbb{Z}_p$ is more commonly used in the mathematical literature to designate the ring of p-adic integers.

If $\phi : A \longrightarrow A'$ is a morphism of commutative rings, then $\ker\phi$ is an ideal of A and $\mathrm{im}\phi$ is a subring of A'. Moreover, the assignment $a + \ker\phi \mapsto \phi(a)$ defines a ring isomorphism $A/\ker\phi \longrightarrow \mathrm{im}\phi$. In particular, if ϕ is surjective, then $A/\ker\phi \simeq A'$ as rings.

Let R be a ring, let M be an R-module, and let $N \subset M$ be a submodule. Considering M as an abelian group and N as a subgroup, the quotient M/N is formed, which is an abelian group. By definition, $RN \subset N$. Hence, the ring morphism $\psi : R \longrightarrow \mathrm{End}(M/N)$, where $\psi_r(\overline{m}) = \overline{rm}$ for all $r \in R, m \in M$, is well-defined and does not depend on the choice of class representative. In conclusion, M/N is an R-module as well. Vector spaces do not require a separate treatment as these form a subclass. If $\phi : M \longrightarrow M'$ is a module morphism, then $\ker\phi$ is a submodule of M and $\mathrm{im}\phi$ is a submodule of M'. Moreover, the map $a + \ker\phi \mapsto \phi(a)$ defines an R-module isomorphism $M/\ker\phi \longrightarrow \mathrm{im}\phi$. In particular, if ϕ is surjective, then $M/\ker\phi \simeq M'$ as R-modules.

10.5 Direct Products and Direct Sums

10.5.1 Direct Products

We now very briefly discuss direct products. Let $\mathcal{I}$ be a nonempty index set and, for each $i \in \mathcal{I}$, let X_i be a group. Then there is a natural group structure on the Cartesian product $\prod_{i \in \mathcal{I}} X_i$, given by component-wise operations. For example, in such a product $X \times Y$, the X-composition is taken in the X-part and, independently, the Y-composition is taken in the Y-part, i.e.,

$$(x, y) \cdot (x', y') := (xx', yy').$$

for all $x, x' \in X, y, y' \in Y$.

This basic idea extends naturally to the case of rings (modules / vector spaces). For example, suppose A, A' are rings. Then $A \times A'$ is a ring with A-addition (resp., A-multiplication) in the A-part and, independently, A'-addition (resp., A'-multiplication) in the A'-part. Note that, if X, Y are fields, then their direct product is *not* a field: each non-zero element with 0 in some part lacks a multiplicative inverse. That said, their product *is* a commutative ring. These remarks are also valid if the index set is infinite.

10.5.2 Direct Sums

A *direct sum* is a constrained direct product, as follows. A direct sum of abelian groups G_i is their direct product under the condition that, for each element of the product, almost all entries are zero, i.e., at most a finite number of entries are nonzero. A direct sum is denoted $\bigoplus G_i$ instead of $\prod G_i$. In the case of a finite number of factors, a direct sum is the same as a direct product. If $n \geq 1$ is an integer and G is a group, then G^n denotes the group $\bigoplus_{i=1}^n G$ (or equivalently, $\prod_{i=1}^n G$). This definition extends to rings and modules as well.

For example, if S is a set, then the *free abelian group* generated by S is the direct sum $\bigoplus_{i\in S} \mathbb{Z}$. If $S = \emptyset$, this is the null-module.

As another example of the use of direct sums, let R be a ring and let M be a (nontrivial) free R-module. For notational simplicity assume that M has a finite basis. Let $S \subset M$ be a nonempty set, consisting of elements $e_1, \ldots, e_n$. Consider the map

$$R^n \longrightarrow M$$

$$(\lambda_1, \ldots, \lambda_n) \mapsto \lambda_1 e_1 + \cdots + \lambda_n e_n.$$

Then S is a basis of M if and only if this map is an isomorphism of R-modules.

An important example is the main result on finite abelian groups, which gives a direct sum decomposition.

THEOREM 10.2 *Let G be a finite abelian group. Then there are (unique) integers $a_1, \ldots, a_\ell \geq 2$ such that $a_1 \mid \cdots \mid a_\ell$ and*

$$G \;\simeq\; \bigoplus_{i=1}^{\ell} \mathbb{Z}/a_i\mathbb{Z}.$$

In particular, a finite abelian group is a direct sum of cyclic groups.

This can be verified as follows. Suppose $g_1, \ldots, g_n$ generate G. Consider the surjective morphism

$$\phi : \mathbb{Z}^n \longrightarrow G$$

$$(\lambda_1, \ldots, \lambda_n) \mapsto \lambda_1 g_1 + \cdots + \lambda_n g_n.$$

of $\mathbb{Z}$-modules. Then its kernel is a free submodule of $\mathbb{Z}^n$. By application of Theorem 10.1 to $\ker\phi$ and by taking into account that $G \simeq \mathbb{Z}^n/\ker\phi$, the theorem follows.

10.5.3 The Chinese Remainder Theorem for Commutative Rings

Let A be a commutative ring. Let I, J be ideals of A. Then I, J are *co-prime* if

$$I + J = A.$$

If I, J are co-prime ideals of A, then I^r, J^s are co-prime ideals of A as well, for each pair of integers $r, s \geq 1$. [19]

THEOREM 10.3 *Suppose $I_1, \ldots, I_m \subset A$ are pairwise co-prime ideals ($m > 1$), i.e., $I_i + I_j = A$ if $i \neq j$. Then the map*

$$\psi : A/(I_1 \cdots I_m) \longrightarrow A/I_1 \times \ldots \times A/I_m,$$

[19] Namely, suppose $a + b = 1$, with $a \in I$ and $b \in J$. Then work out $(a + b)^k = 1$ for some large enough positive integer k.

$$a \bmod I_1 \cdots I_m \mapsto (a \bmod I_1, \ldots, a \bmod I_m)$$

is an isomorphism of rings.

As an application, if $n > 1$ is an integer and $n = p_1^{e_1} \cdots p_k^{e_k}$ is its factorization into positive powers of distinct positive prime numbers, we get the result that

$$\mathbb{Z}/n\mathbb{Z} \;\simeq\; \bigoplus_{\ell=1}^{k} \mathbb{Z}/p_\ell^{e_\ell}\mathbb{Z}$$

as rings.

10.6 Basic Field Theory

Let K be a field and let L be an extension field.

If $S \subset L$ is a set, then $K(S)$ denotes the field generated by K and S, i.e., the intersection of all subfields K' of L such that K is a subfield of K' and $S \subset K'$. If $L = K(S)$ for some finite set $S \subset L$, then L *is finitely generated over* K. If $S = \{x_1, \ldots, x_m\}$, then we write $K(x_1, \ldots, x_m)$ for $K(S)$.

10.6.1 Algebraic Extensions

An element $x \in L$ is *algebraic* over K if there is a polynomial $f(X) \in K[X]$, $\deg(f) \geq 1$ such that $f(x) = 0$. A polynomial $f(X) \in K[X]$ is *monic* if its leading coefficient is equal to 1. Suppose $x \in L$ is algebraic over K. Consider the surjective ring morphism $\phi : K[X] \longrightarrow K(x)$ defined by the assignment

$$g(X) \mapsto g(x).$$

Since $K[X]/\ker\phi \simeq K(x)$ and $K(x)$ is a field, it holds that $\ker\phi$ is a maximal ideal of $K[X]$. The unique monic irreducible polynomial $f(X) \in K[X]$ that generates this ideal is the *minimal polynomial* of x over K. In other words, it is the unique monic polynomial $f(X) \in K[X]$ of least degree ≥ 1 such that $f(x) = 0$.

The extension field L is *algebraic over* K if each $x \in L$ is algebraic over K. If $[L : K] < \infty$, then L is algebraic over K.

If L' is an intermediate extension $K \subset L' \subset L$, then L is algebraic over K if and only if L is algebraic over L' and L' is algebraic over K. Also, if $[L : K] < \infty$ the tower relation

$$[L : K] = [L : L'] \cdot [L' : K]$$

holds. [20]

If $x \in L$ is not algebraic over K, then it is *transcendental* over K. Elements $x_1, \ldots, x_m \in L$ are *algebraically independent* over K if they do not jointly satisfy

[20] This also holds if $[L : K] = \infty$. But it is obvious in this case and not very informative.

some nontrivial polynomial relation over K. Precisely, consider the *evaluation map*

$$K[X_1, \ldots, X_m] \longrightarrow L$$

$$f \mapsto f(x_1, \ldots, x_m).$$

Then it is required that this map is injective. A subset of L that is maximal with respect to algebraic independence over K is a *transcendence basis*. This notion of basis bears similarities to the notion of basis from linear algebra: any two transcendence bases have the same cardinality and any algebraically independent set can be completed to a basis. The cardinality of a basis is the *transcendence degree* of L over K. If L is algebraic over K any transcendence basis is empty and hence the transcendence degree is 0. The field of rational functions $K(X)$, i.e., the field of fractions of the polynomial ring $K[X]$, has transcendence degree 1 over K. If S is a transcendence basis of L over K then L is algebraic over $K(S)$ by maximality, where $K(S)$ denotes the subfield of L generated by K and S.

10.6.2 Algebraic Closure

A field K is *algebraically closed* if each polynomial in $K[X]$ of degree ≥ 1 has at least one root in K. Equivalently, each such polynomial factorizes as a product of linear terms in $K[X]$. [21] An *algebraic closure* of K is an algebraic extension field of K that is algebraically closed.

An algebraic closure of K exists and it is unique, *up to K-isomorphism*, i.e., a field isomorphism that fixes K point-wise. In particular, the set of distinct roots in an algebraic closure $\overline{K}$ of a polynomial $f(X) \in K[X]$ with $n := \deg(f) \geq 1$ is also unique up to such a K-isomorphism, while their multiplicities are unique. In particular, there are exactly n roots with multiplicities in an algebraic closure.

For example, the field of complex numbers is the algebraic closure of the field of real numbers. The algebraic closure of the field of rational numbers is a proper subfield of the field of complex numbers. An algebraic closure $\overline{\mathbb{F}}_q$ of $\mathbb{F}_q$ is an infinite extension field of $\mathbb{F}_q$ that essentially consists of the "union" of all of its finite extensions, properly "glued together" using a so-called *direct limit* in order to make this precise.

10.6.3 Separable Extensions

Suppose L is algebraic over K. Let $\overline{L}$ be an algebraic closure of L. [22] A *K-embedding* of L into $\overline{L}$ is a field morphism $L \longrightarrow \overline{L}$ that leaves K pointwise fixed. Note that a morphism between fields is always injective. Indeed, its kernel is an ideal and out of the only two possibilities for this ideal, i.e., the zero ideal and the whole field, the latter can be discarded as 1 must be mapped to 1.

[21] Use successive application of the *Euclidean algorithm* to split off one linear term after another.

[22] This is an algebraic closure of K as well, since L is algebraic over K.

If $n := [L : K] < \infty$, then there are at most n distinct K-embeddings σ of L into $\overline{L}$. It should be noted that $\sigma(L) \subset L$ implies that σ is a K-automorphism of L: σ is in particular an endormorphism of L as a finite-dimensional K-vector space, so σ is surjective. If $x \in L$ and $L = K(x)$, then the elements of the form $\sigma(x) \in \overline{L}$, where σ runs through the distinct K-embeddings of L, correspond one-to-one with the distinct roots [23] in $\overline{L}$ of the minimal polynomial $f(X)$ of x. [24] These are called the *conjugates* of x.

An element $x \in L$ is *separable* over K if it is algebraic over K and its minimal polynomial $f(X) \in K[X]$ has exactly $\deg(f)$ distinct roots. Moreover, L is *separable over* K if each element $x \in L$ is separable over K. For separability, the tower property holds that if L' is an intermediate extension, then L is separable over K if and only if L is separable over L' and L' is separable over K. If $[L : K] < \infty$, separability of L over K is equivalent to the existence of exactly $[L : K]$ distinct K-embeddings of L into $\overline{L}$.

If $[L : K] < \infty$ and if L is separable over K, then L is a *simple* extension of K, i.e., $L = K(x)$ for some $x \in L$. [25]

The field K is *perfect* if L' is separable over K for all finite-degree extension fields L' of K. For example, finite fields are perfect, as well as any field of characteristic 0.

10.6.4 Galois Correspondence

We state the main result of Galois theory for Galois field extensions of finite degree. We still suppose L is algebraic over K. The field L is *normal* over K if for each $x \in L$ it holds that each of the roots of its minimal polynomial is contained in L. If $n := [L : K] < \infty$, this is equivalent to requiring that each of the $\leq n$ distinct K-embeddings σ of L into $\overline{L}$ maps is a K-automorphism of L. This is the case if and only if $\sigma(L) \subset L$ for each of these embeddings σ.

The field L is a finite-degree *Galois extension* of K if $n < \infty$, L is separable over K and L is normal over K. Equivalently, each of the n distinct K-embeddings of L into $\overline{L}$ is a K-automorphism of L, i.e., $|\mathrm{Aut}(L/K)| = [L : K] < \infty$. This group is then the *Galois group* of the extension.

The *Galois correspondence* provides a complete description of the intermediate fields $K \subset K' \subset L$ in terms of the Galois group.

THEOREM 10.4 *Let K be a field and let L be a finite-degree extension field.*

[23] i.e., the roots of the distinct monic linear factors of the factorization of $f(X)$ in $\overline{L}[X]$.

[24] This follows from the following two facts. First, $f(x) = 0$ implies $f(\sigma(x)) = 0$ since σ fixes K. Second, if $y \in \overline{L}$ is also a root of $f(X)$, then there is a unique field morphism $K(x) \longrightarrow K(y)$ such that $x \mapsto y$.

[25] More generally, the Primitive Element Theorem states that an algebraic extension of finite degree is simple if and only if the number of intermediate extensions is finite.

Suppose L is Galois over K, with Galois group G. Then there is a one-to-one, inclusion-reversing correspondence between the set of intermediate fields $K \subset K' \subset L$ and the set of intermediate groups H with $\{1\} \subset H \subset G$, characterized as follows.

- *In the forward direction, an intermediate field K' is mapped to the subgroup H of G that fixes K' pointwise.*
- *In the other direction, a subgroup H is mapped to the intermediate field K' that is fixed pointwise by H.*

Additionally, K' is Galois over K if and only if the subgroup H of G corresponding to K' is a normal subgroup of G. In this case, G/H is isomorphic to the Galois group of the extension $K \subset K'$ (and the map $\overline{\sigma} \mapsto \sigma_{|K'}$ gives an isomorphism between these groups).

For example, the finite field $\mathbb{F}_{q^n}$ ($n \geq 1$ an integer) is a Galois extension of the finite field $\mathbb{F}_q$ of degree n. Its Galois group is cyclic and is generated by the *Frobenius automorphism* of $\mathbb{F}_{q^n}$ given by $x \mapsto x^q$.

Suppose L is separable over K and $n := [L : K] < \infty$.

Fix some algebraic closure $\overline{L}$ of L. Let $\sigma_1, \ldots, \sigma_n$ be the distinct K-embeddings of L into $\overline{L}$. Let $x \in L$. Then the *norm*, resp. *trace* of x in L/K are defined as follows:

$$\mathrm{N}_{L/K}(x) = \prod_{i=1}^{n} \sigma_i(x) \in \overline{L},$$

$$\mathrm{Tr}_{L/K}(x) = \sum_{i=1}^{n} \sigma_i(x) \in \overline{L}.$$

By Galois theory, it holds that

$$\mathrm{N}_{L/K}(x) \in K \ , \ \mathrm{Tr}_{L/K}(x) \in K.$$

In particular, the norm and trace do not depend on the choice of an algebraic closure of L. Consider the *normal closure* L' of L in $\overline{L}$, i.e., the compositum of the subfields $\sigma_1(L), \ldots, \sigma_n(L)$ of $\overline{L}$. This contains L as a subfield (since the identity is among those K-embeddings). Observe that L' is a finite-degree Galois extension of K. [26] Write G' for its Galois-group. Select an arbitrary $\tau \in G'$. Then $\tau \circ \sigma_1, \ldots, \tau \circ \sigma_n$ is a permutation of $\sigma_1, \ldots, \sigma_n$, since each of those gives a different K-embedding of L into $\overline{L}$. Therefore, for each $x \in L$, the norm and the trace of x are fixed under each $\tau \in G'$. The desired claim follows, since an element of L' is fixed under all of G' if and only if it is an element of K.

[26] Finiteness of the extension degree and separability are straightforward. The fact that any K-embedding τ of L' into some algebraic closure is a K-automorphism of L' follows directly from the definition of L' in combination with the fact that $\tau \circ \sigma_1, \ldots, \tau \circ \sigma_n$ is a permutation of $\sigma_1, \ldots, \sigma_n$; see also below.

The norm is multiplicative, i.e., $\mathrm{N}_{L/K}(xy) = \mathrm{N}_{L/K}(x) \cdot \mathrm{N}_{L/K}(y)$ for all $x, y \in L$ and that the trace is a K-linear form.

We give some examples. Suppose $x \in L$ satisfies $L = K(x)$. This makes sense since L/K is a separable, finite-degree extension by assumption. Let $f(X) \in K[X]$ be the minimal polynomial of x. Then the constant term of $f(X)$ equals $(-1)^n \cdot \mathrm{N}_{L/K}(x)$ and the coefficient of X^{n-1} in $f(X)$ equals $-\mathrm{Tr}_{L/K}(x)$. As another example, suppose $K = \mathbb{F}_q$ and $L = \mathbb{F}_{q^k}$ for some positive integer k. Then

$$\mathrm{Tr}_{L/K}(y) = \sum_{i=0}^{k-1} y^{q^i}$$

for all $y \in \mathbb{F}_{q^k}$, since the Galois group of $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$ is generated by the Frobenius automorphism.

10.7 Algebraic Number Fields

10.7.1 Basic Definitions

An *algebraic number field* is an extension field K of the rational numbers $\mathbb{Q}$ such that $[K : \mathbb{Q}] < \infty$. Note that K is a simple extension of $\mathbb{Q}$, i.e., $K = \mathbb{Q}(x)$ for some $x \in K$. [27] An element $y \in K$ is *integral* over the rational integers $\mathbb{Z}$ if its minimal polynomial $f(X) \in \mathbb{Q}[X]$ of y satisfies $f(X) \in \mathbb{Z}[X]$: each coefficient of this monic polynomial is an integer.

The subset $\mathcal{O}_K \subset K$ consisting of all $y \in K$ that are integral over $\mathbb{Z}$ is a subring of K, the *ring of integers* of K. The field of fractions of $\mathcal{O}_K$ equals K. [28] If $\overline{\mathbb{Q}}$ is an algebraic closure of the rational numbers, then the set of elements $y \in \overline{\mathbb{Q}}$ such that y is integral over $\mathbb{Z}$, the *integral closure* of $\mathbb{Z}$ in $\overline{\mathbb{Q}}$, is a subring of $\overline{\mathbb{Q}}$. Trivially, if y is integral over $\mathbb{Q}$, then so are all of its conjugates over $\mathbb{Q}$.

For example, the ring of integers of $\mathbb{Q}$ is $\mathbb{Z}$ as no element in $\mathbb{Q} \setminus \mathbb{Z}$ is a zero of a monic polynomial with integer coefficients. As another example, let $K = \mathbb{Q}(i)$ with $i \in \mathbb{C}$ such that $i^2 = -1$, then $\mathcal{O}_K = \mathbb{Z}[i]$, the *Gaussian integers* $a + bi$ with $a, b \in \mathbb{Z}$. Even though a number field K is always a simple extension of $\mathbb{Q}$, it is typically not true that $\mathcal{O}_K$ is a simple extension of $\mathbb{Z}$.

10.7.2 Dedekind Domains and Class Groups

Let K be a number field. The first question about $\mathcal{O}_K$ is to what extent it resembles the rational integers $\mathbb{Z}$. An answer is given by the result that $\mathcal{O}_K$ is a *Dedekind domain.*

Let A be a domain that is not a field and let $Q(A)$ be its field of fractions. Then A is a Dedekind domain if the following conditions are satisfied.

[27] The extension is separable since it is in characteristic 0. So, together with the fact that the degree is finite, the Primitive Element Theorem gives the desired result.

[28] It is easy to see that, for each $y' \in K$, there is some nonzero $\ell \in \mathbb{Z}$ such that $\ell \cdot y' \in \mathcal{O}_K$.

1. A is Noetherian, i.e., each of its ideals is finitely generated as an A-module.
2. A is integrally closed within $Q(A)$, i.e., if $f(X) \in A[X]$ is a monic polynomial and if $z \in Q(A)$ satisfies $f(z) = 0$, then $z \in A$.
3. Each nonzero prime ideal of A is maximal.

Basic examples of Dedekind domains are the ring of rational integers and the polynomial ring over a field. In each case, it is straightforward to show that the ring in question is integrally closed within its field of fractions, whereas the other properties are clear already. An immediate consequence of $\mathbb{Z}$ being integrally closed is that, for all $x \in \mathcal{O}_K$, it holds that $\mathrm{N}_{K/\mathbb{Q}}(x) \in \mathbb{Z}$ and $\mathrm{Tr}_{K/\mathbb{Q}}(x) \in \mathbb{Z}$: from the definition and from basic field theory, these values are in $\mathcal{O}_K \cap \mathbb{Q} = \mathbb{Z}$. It holds that $x \in \mathcal{O}_K$ is a unit if and only if $|\mathrm{N}_{K/\mathbb{Q}}(x)| = 1$. In the forward direction, suppose $x, x' \in \mathcal{O}_K$ satisfy $xx' = 1$. Then

$$1 = \mathrm{N}_{K/\mathbb{Q}}(1) = \mathrm{N}_{K/\mathbb{Q}}(xx') = \mathrm{N}_{K/\mathbb{Q}}(x) \cdot \mathrm{N}_{K/\mathbb{Q}}(x').$$

Since $\mathrm{N}_{K/\mathbb{Q}}(x), \mathrm{N}_{K/\mathbb{Q}}(x') \in \mathbb{Z}$, the claim follows. The other direction follows at once from the observation that, for each $x \in \mathcal{O}_K$, there is $y \in \mathcal{O}_K$ such that $\mathrm{N}_{K/\mathbb{Q}}(x) = x \cdot y$ (use the fact that one of the embeddings in the definition of the norm is the identity). Also, it can be shown that each ideal of $\mathcal{O}_K$ is generated as an $\mathcal{O}_K$-module by at most two elements of $\mathcal{O}_K$.

If A is a Dedekind domain, it enjoys unique factorization of ideals into products of prime *ideals*. Concretely, suppose I is an ideal of A with $(0) \subsetneq I \subset A$. Then

$$I = \prod_P P^{e_P},$$

where the product ranges over all nonzero primes ideals P in A and where e_P is a nonnegative integer which is nonzero for at most finitely many of them. This expression is unique. Note that this "prime-ideal-factorization" is in fact a relaxation of the "prime-*element*-factorization" property enjoyed by unique factorization domains. [29]

A nonzero A-submodule I of $Q(A)$ is a *fractional ideal* of A if there exists $c \in A$ with $c \neq 0$ such that $cI \subset A$. The product of two fractional ideals, defined similarly as the product of two ideals, is a fractional ideal as well. It is clear that A acts as a neutral element: $AI = I$ since $1 \in A$. Hence, the fractional ideals form a (commutative) multiplicative monoid. It can be shown that the fractional ideals form a group if A is a Dedekind domain. Note that in order for the group property to hold, the only condition left to be verified is that for each fractional ideal I there is a fractional ideal J such that $IJ = A$. [30]

Suppose A is a Dedekind domain. If $x \in Q(A)$ with $x \neq 0$, then xA is a *principal fractional ideal*. These form a subgroup of the group of fractional ideals.

[29] It should be noted that in the case of number fields, $\mathcal{O}_K$ is typically *not* a unique factorization domain.

[30] Sure, there is $c \in A$ such that $(c) \cdot I \subset A$ by definition. But the point is existence of a fractional ideal J such that IJ achieves *equality* with A.

The *ideal class group* of A is then the group of fractional ideals taken modulo the principal fractional ideals. For example, the ideal class group of the number field K (i.e., that of $\mathcal{O}_K$) is a finite (commutative) group and its cardinality h is the *class number* of K. To say that $\mathcal{O}_K$ is a PID is equivalent to saying that $h = 1$.

Finally, if $I \neq (0)$ is a fractional ideal of A, then

$$I = \prod_P P^{e'_P},$$

where the product ranges over all nonzero primes ideals P in A and where e'_P is an integer which is nonzero for at most finitely many of them. This expression is unique. In other words, the group of fractional ideals of a Dedekind domain is isomorphic to the free abelian group generated by its prime ideals.

As an aside, it can be shown that a Dedekind domain A can be equivalently defined as a domain A that is not a field such that its fractional ideals form a group.

10.7.3 The Fundamental Identity

A further important property is that $\mathcal{O}_K$ *is a free* $\mathbb{Z}$*-module of rank* $n := [K : \mathbb{Q}]$. Before we establish the consequences, we define the notion of an *order*, which is motivated by this fact. An order in a number field K of degree n is a subring of $\mathcal{O}_K$ that is of rank n as a $\mathbb{Z}$-module. Sometimes $\mathcal{O}_K$ is referred to as the *maximal order.*

Suppose P is a nonzero prime ideal of $\mathcal{O}_K$. It follows easily that

$$P \cap \mathbb{Z} = p\mathbb{Z}$$

for some prime number $p \in \mathbb{Z}$ with $p > 0$. Consequently, $\mathcal{O}_K/P$ is the finite field $\mathbb{F}_{p^f}$ for some integer f with $1 \leq f \leq n$: $\mathcal{O}_K/P$ is a field since P is maximal and $\mathbb{Z}/(P \cap \mathbb{Z})$ $(=\mathbb{F}_p)$ is a subring of $\mathcal{O}_K/P$, which is henceforth an $\mathbb{F}_p$-vector space of dimension at most n as it is generated by $\overline{b_1}, \ldots, \overline{b_n}$ where $b_1, \ldots, b_n$ is a $\mathbb{Z}$-basis of $\mathcal{O}_K$.

Now let $p \geq 2$ be a prime number. Consider the Dedekind-factorization

$$p\mathcal{O}_K = P_1^{e_1} \cdots P_k^{e_k}$$

into positive powers of pairwise distinct prime ideals $P_1, \ldots, P_k$. We say that the prime number p *ramifies* in the number field if some $e_i > 1$ and we say that it is *unramified* otherwise.

By the CRT,

$$\mathcal{O}_K/p\mathcal{O}_K \simeq \mathcal{O}_K/P_1^{e_1} \times \cdots \times \mathcal{O}_K/P_k^{e_k}.$$

Note that $\mathcal{O}_K/p\mathcal{O}_K$ is an $\mathbb{F}_p$-vector space of dimension n. [31] Let f_i be the positive

[31] In fact, it is an $\mathbb{F}_p$-algebra of dimension n. See Section 10.8.

integer such that

$$\mathcal{O}_K/P_i = \mathbb{F}_{p^{f_i}},$$

for $i = 1, \ldots, k$. [32] Then the *Fundamental Identity* states that

$$e_1 f_1 + \cdots + e_k f_k = n.$$

10.7.4 Cyclotomic Number Fields

Here is an example that is relevant for our purposes. Let $p \in \mathbb{Z}$ be a prime number with $p > 2$. [33] Consider the polynomial $X^p - 1 \in \mathbb{Z}[X]$. Its roots in the complex numbers $\mathbb{C}$ are exactly the elements of the group of *p-th roots of unity*. This is a cyclic group of order p, generated by a primitive p-th root of unity, say

$$\omega = e^{\frac{2\pi i}{p}}$$

(where $e = 2.71..$ is the base of the natural logarithm and where $i \in \mathbb{C}$ satisfies $i^2 = -1$). Next, consider the polynomial

$$f(X) = X^{p-1} + \cdots + X + 1 \in \mathbb{Z}[X],$$

which is irreducible in $\mathbb{Q}[X]$ (as follows after substituting $X = Y + 1$ and applying *Eisenstein's irreducibility criterion*). Since $(X - 1)f(X) = X^p - 1$, the roots of $f(X)$ are exactly the p-th roots of unity with 1 excluded, i.e., the roots of $f(X)$ are $\omega, \ldots, \omega^{p-1}$.

The *p-th cyclotomic number field* is the number field

$$K := \mathbb{Q}(\omega) \simeq \mathbb{Q}[X]/(f(X)),$$

which is of degree $p - 1$ over $\mathbb{Q}$. It is immediate that K is a Galois extension of $\mathbb{Q}$ of degree $p-1$ with a cyclic Galois group. The *p-th ring of cyclotomic integers* is the ring $\mathcal{O}_K$. The ring $\mathcal{O}_K$ is a simple extension of $\mathbb{Z}$. In fact,

$$\mathcal{O}_K = \mathbb{Z}[\omega],$$

i.e., $\mathcal{O}_K$ is the ring generated by $\mathbb{Z}$ and ω. This ring has the special property that for each prime number $p' \in \mathbb{Z}$, the ideal (p') of $\mathcal{O}_K$ factorizes in such a way that the powers occurring in the factorization are all equal to 1, *except when* $p' = p$ where it holds that

$$p\mathcal{O}_K = P^{p-1}$$

where P is the unique nonzero prime ideal of $\mathcal{O}_K$ with

$$P \cap \mathbb{Z} = p\mathbb{Z},$$

i.e., the ideal

$$P = (1 - \omega).$$

[32] Note that $\mathcal{O}_K/P_i \,\cap\, \mathbb{Z} = p\mathbb{Z}$, for $i = 1, \ldots, k$.
[33] The discussion to follow is rather vacuous for $p = 2$ (but true).

From the Fundamental Identity,

$$p - 1 = e_1 f_1 = (p-1) \cdot 1$$

and thus

$$\mathcal{O}_K / P \simeq \mathbb{F}_p.$$

Finally, we state some further useful property concerning the primitive roots of unity. Note that

$$\mathrm{N}_{K/\mathbb{Q}}(1 - \omega^i) = p,$$

for $i = 1, \ldots, p-1$. This is easy to see, as the polynomial $g(X) = f(-X + 1) \in \mathbb{Q}[X]$ is the minimal polynomial over $\mathbb{Q}$ for the complete set of conjugates $1 - \omega, \ldots, 1 - \omega^{p-1}$. Now just read off the constant term in $g(X)$. Next, observe that, for $i = 1, \ldots, p-1$, we have

$$\frac{1 - \omega^i}{1 - \omega} = 1 + \cdots + \omega^{i-1}$$

and, by multiplicativity of the norm,

$$\mathrm{N}_{K/\mathbb{Q}}\left(\frac{1 - \omega^i}{1 - \omega}\right) = 1.$$

In conclusion, we have the following lemma.

LEMMA 10.5 *Let $p > 2$ be a prime number. Let $\mathcal{O}_K$ denote the ring of integers of the p-th cyclotomic number field. Let $\omega \in \mathcal{O}_K$ be a primitive p-th root of unity. Then, for $i = 0, \ldots, p-2$, it holds that $\sum_{j=0}^{i} \omega^j$ is a unit of $\mathcal{O}_K$.*

10.8 Algebras

10.8.1 Basic Definitions

Let K be a field. *For our purposes, a K-algebra A will mean a commutative ring A such that K is a subring.* [34] In particular, this means that A is a K-vector space of dimension ≥ 1. Note that $1_K = 1_A$.

Products of K-algebras, e.g., the n-fold direct product K^n, where $n \geq 1$ is an integer, will be viewed as K-algebras with component-wise multiplication as ring-multiplication and with K "diagonally embedded", i.e., $\lambda \in K$ is given by $(\lambda, \ldots, \lambda, \ldots)$ in the product. In particular, the multiplicative unity is the element $(1_K, \ldots, 1_K, \ldots)$.

A *morphism* $\phi : A \longrightarrow B$ of K-algebras is a map that is both a ring morphism and a K-vector space morphism. Note that this facilitates both ring-theoretic as well as linear algebraic arguments. Two K-algebras A, B are isomorphic if there

[34] Note that this definition is more restrictive than common definitions in the literature, where typically there is no a priori requirement concerning commutativity. Besides, algebras may be defined over rings instead of fields. There are further relaxations. See [126], Ch. III, par. 1, p. 121.

are K-algebra morphisms $\phi : A \longrightarrow B$ and $\phi' : B \longrightarrow A$ such that $\phi \circ \phi'$ is the identity on B and $\phi' \circ \phi$ is the identity on A. A morphism is an isomorphism if and only if it is bijective.

10.8.2 Trace, Norm and Discriminant

Let V be an n-dimensional K-vector space ($1 \leq n < \infty$) and let ϕ be an endomorphism of V. Let $\Phi := (\phi_{ij})$ be the matrix representing ϕ with respect to some given basis. The *characteristic polynomial* of ϕ is defined as

$$f(X) := \det(\Phi - X \cdot I_n) \in K[X],$$

where I_n denotes the $n \times n$ identity matrix over K. The *determinant* of ϕ is $\det(\Phi)$. The *trace* of ϕ is defined as

$$\mathrm{Tr}(\phi) = \sum_{i=1}^{n} \phi_{ii} \ \in K.$$

These definitions do not depend on the choice of basis. It holds that

$$f(X) = X^n - \mathrm{Tr}(\phi) \cdot X^{n-1} + \cdots + (-1)^n \cdot \det(\phi).$$

Suppose A is a K-algebra with $n := \dim_K A < \infty$. For each $a \in A$, define

$$\mu_a : A \longrightarrow A$$

$$x \mapsto ax,$$

the *multiplication-by-a-map* on A. Note that this is a K-vector space endomorphism of A. The *trace function* of A/K is defined as

$$\mathrm{Tr}_{A/K} : A \longrightarrow K$$

$$a \mapsto \mathrm{Tr}(\mu_a).$$

This is a K-linear form. Note that $\mathrm{Tr}_{A/K}(xy)$ is a K-bilinear form on $A \times A$ (i.e., it is K-linear in each of its arguments x, y). It is *nondegenerate* if for each $x \in A$ there is $y \in A$ such that $\mathrm{Tr}_{A/K}(xy) \neq 0$.

PROPOSITION 10.6 *If A is an* extension field *of K of* degree $n < \infty$*, then the following statements are equivalent.*

1. *A is separable over K.*
2. *The trace function $\mathrm{Tr}_{A/K}$ is not identically zero.*
3. *The K-bilinear form $\mathrm{Tr}_{A/K}(xy)$ on $A \times A$ is nondegenerate.*
4. *The K-linear forms $\phi : A \longrightarrow K$ coincide with the K-linear forms $\mathrm{Tr}_{A/K}(ax) : A \longrightarrow K$, with $a \in A$ fixed.*

PROPOSITION 10.7 *If A is a separable field extension of K of degree $n < \infty$, then the following holds.*

1. *The definition of trace as given above coincides with the definition in terms of the K-embeddings of A into some algebraic closure as given before.*
2. *Let $a \in A$. The characteristic polynomial of μ_a equals the minimal polynomial of a over K taken to the $[A : K(a)]$-th power. In particular, if $A = K(a)$, then the minimal polynomial of a over K and the characteristic polynomial of μ_a are identical.*
3. *There exists a* dual basis *for each given basis. Concretely, if $e_1, \ldots, e_n \in A$ constitute a K-basis of A, then there is a K-basis $e'_1, \ldots, e'_n$ of A such that* $\mathrm{Tr}(e_i e'_j) = \delta_{ij}$ *(Kronecker delta).*

The second claim implies that

$$\det(\mu_a) = \left(\mathrm{N}_{K(a)/K}(a)\right)^{[A:K(a)]}$$

and that

$$\mathrm{Tr}_{A/K}(a) = [A : K(a)] \cdot \mathrm{Tr}_{K(a)/K}(a).$$

Note that the third claim is just linear algebra; it is an immediate consequence of the fact that $\mathrm{Tr}(xy)$ is a non-degenerate K-bilinear form.

In general, when A is a K-algebra with $n := \dim_K A < \infty$, the *discriminant* of a set of elements $(a_1, \ldots, a_n) \in A$ is defined as

$$\mathrm{D}_{A/K}(a_1, \ldots, a_n) = \det(\mathrm{Tr}(a_i a_j)),$$

which is an element of K.

The discriminant of A is the discriminant of a K-basis of A. This is only well-defined up to multiplication by the square of a unit in K. [35] This follows immediately by linear algebra. The discriminant of A over K is said to *differ from 0* if the discriminant of a K-basis differs from 0. Otherwise, the discriminant of A over K is said to equal 0. The discriminant is *multiplicative* in the sense that the discriminant of a finite direct product of K-algebras is nonzero if and only if the discriminant of each of the factors is nonzero.

With some straightforward formula manipulations, we get the following useful expressions for the discriminant.

Proposition 10.8 *If A is a separable field extension of K of degree $n < \infty$ and $a \in A$ is such that $A = K(a)$, then the following holds.*

1. *The discriminant of the K-basis $1, a, \ldots, a^{n-1}$ of A satisfies*

$$\mathrm{D}_{A/K}(1, a, \ldots, a^{n-1}) = (-1)^{\frac{1}{2}n(n-1)} \cdot \mathrm{N}_{A/K}(f'(a))$$

where $f(X) \in K[X]$ is the minimal polynomial of a over K and $f'(X) \in K[X]$ is its formal derivative. [36]

[35] Namely, the square of the determinant of a base-change.
[36] Also in this case a dual basis can be expressed in terms of $f'(X)$. For more information, see [126], Ch. VI, par. 5.

2. *If $a_1, \dots, a_n$ is the complete set of conjugates of a over K, then*

$$\mathrm{D}_{A/K}(1, a, \dots, a^{n-1}) = \prod_{i<j}(a_i - a_j)^2.$$

3. *Let $a'_1, \dots, a'_n \in A$. Then $\mathrm{D}_{A/K}(a'_1, \dots, a'_n) \neq 0$ if and only if $a'_1, \dots, a'_n$ constitute a K-basis of A.*

Definition 10.9 *The* discriminant *of a monic polynomial $f(X) \in K[X]$ of degree $n \geq 1$ is defined as*

$$D(f) = \prod_{1 \leq i < j \leq n} (a_i - a_j)^2,$$

where $a_1, \dots, a_n$ is a complete set of roots (with multiplicity) in some splitting field of $f(X)$.

Note that $D(f) = 0$ if and only if $f(X)$ has roots with multiplicity greater than 1. As $D(f)$ is a symmetric polynomial expression in the roots of $f(X)$, it follows by Galois theory [37] that the discriminant can (in principle) be given as a polynomial expression in the coefficients of the polynomial. [38] For instance, suppose $f(X) \in K[X]$ is an irreducible *trinomial*, i.e., $f(X) = X^n + uX + v$ for some $u, v \in K$. Then a straightforward calculation [39] exploiting the above norm-derivative expression of the discriminant yields

$$D(f) = (-1)^{\frac{1}{2}n(n-1)}(n^n v^{n-1} + (-1)^{n-1}(n-1)^{n-1}u^n).$$

For $n = 2$, this gives the familiar expression $u^2 - 4v$ from high-school algebra. For $n = 3$, this gives the expression $-27v^2 - 4u^3$, which is relevant when dealing with Weierstrass equations $Y^2 = X^3 + uX + v$ from the theory of elliptic curves (in characteristic $\neq 2, 3$).

Proposition 10.10 *Let A be a K-algebra with $n := \dim_K A < \infty$. Then the following holds.*

1. *Suppose A is an extension field of K. Then A is separable over K if and only if the discriminant of A/K is nonzero.*
2. *If A contains a nonzero nilpotent, then the discriminant of A/K equals 0.*

Proof We start by verifying the first claim. The forward direction holds on account of linear algebra. If ψ is a nondegenerate K-bilinear form on an n-dimensional K-vector space and $e_1, \dots, e_n \in V$, then $\det(\psi(e_i, e_j)) \neq 0$ if and only if $e_1, \dots, e_n$ constitutes a K-basis of V. Since the trace function $\mathrm{Tr}(xy)$ is a nondegenerate K-bilinear form on A by separability, the result follows. In the

[37] This follows from the basic result that symmetric polynomials can be expressed as polynomials in the elementary symmetric ones. See e.g. [126], p. 272, Example 4.

[38] In the case of norm and trace, which are similarly symmetric functions, this expression is particularly straightforward.

[39] For instance, see [161], Section 2.7.

other direction, if A is not separable over K, then the trace function vanishes, and so does the discriminant.

As to the second claim, suppose $a \in A$ is a nonzero nilpotent. Then μ_a is a nilpotent K-vector space endomorphism of A. In particular, it cannot have nonzero eigenvalues. Therefore, all of the n roots of its characteristic polynomial are 0. Consequently, the trace equals 0. Now choose a K-basis of A that includes a. Then the product between a and any basis element y is nilpotent, hence $\mathrm{Tr}_{A/K}(ay) = 0$. It follows directly that the discriminant vanishes, as the matrix from its definition has a row consisting of zeros only. □

10.8.3 Basic Characterization Theorems

Finite products of finite degree extension fields of K are among the most elementary examples of K-algebras. There is the following characterization.

THEOREM 10.11 *Let A be a K-algebra with $n := \dim_K A < \infty$. Then:*

1. *A has 0 as its only nilpotent if and only if A is a finite direct product of finite-degree extension fields of K.*
2. *A has non-zero discriminant over K if and only if A is a finite direct product of finite-degree separable extension fields of K.*

In particular, if K is a perfect field, then A has 0 as its only nilpotent if and only if A has nonzero discriminant.

PROOF The first claim is shown as follows (forward direction only as the other is immediate). Let P be any nonzero prime ideal of A. We claim that P is maximal. Since P is prime, A/P is a domain. Also, A/P is a K-vector space of finite dimension. For a nonzero element $\mu \in A/P$, consider the multiplication-by-μ map on A/P, which is a vector space isomorphism: it is injective since A/P is a domain and an injective endomorphism of a finite dimensional vector space is an isomorphism. Consequently, there is some $\mu' \in A/P$ such that $\mu\mu' = \mu'\mu = 1$. Thus, A/P is a field and P is maximal.

Next, the nilradical (i.e., the ideal of A consisting of all its nilpotents) is (0) by hypothesis. Since the nilradical of a commutative ring is the intersection of its prime ideals, it follows here that the intersection taken over all maximal ideals in A is (0).

Finally, observe that there are at most finitely many maximal ideals of A. Consider the canonical map of rings

$$A \longrightarrow A/P_1 \times \cdots \times A/P_\ell,$$

where $P_1, \ldots, P_\ell$ are distinct maximal ideals of A. This is a morphism of K-algebras and the terms on the right are finite-degree extension fields of K. By the Chinese Remainder Theorem, this map is surjective. By linear algebra, the K-dimension on the left (which equals n) is an upper bound for the total K-dimension on the right (which is at least ℓ). It follows that $\ell \leq n$. Therefore, A

has at most finitely many maximal ideals. Now assume that $P_1, \dots, P_\ell$ enumerate all of the maximal ideals of A. Then this canonical map is also injective, since the intersection of the maximal ideals of A is (0), as established above. Hence, it is a K-algebra isomorphism.

The second claim follows by combining the first with Proposition 10.10 (also use multiplicativity of the discriminant). This concludes the proof.[40] $\square$

We give an important application to algebraic number theory. The discriminant of a number field is the discriminant of a $\mathbb{Z}$-basis of its ring of integers $\mathcal{O}$. This definition does not depend on the choice of the basis, since a base-change incurs a multiple equal to the square of the determinant of the base-change and this determinant equals ± 1 here. Reasoning as in the case of trace and norm, the discriminant of a number field is an integer. If the number field is the field of rational numbers itself, then the discriminant equals 1. If the degree of the number field is greater than 1, it holds that the discriminant is not equal to $-1, 0, 1$. By Stickelberger's Criterion, [41] it is 0 mod 4 or 1 mod 4. Using Theorem 10.1, it is not hard to see that the discriminant of an order in $\mathcal{O}$ equals the square of a nonzero integer times the discriminant of the number field. As an application, let

$p \in \mathbb{Z}$ be a positive prime number. By inspection we see that p is unramified if and only if $\mathcal{O}/p\mathcal{O}$ is a product of finite extension fields of $\mathbb{F}_p$. This holds if and only if the discriminant of $\mathcal{O}/p\mathcal{O}$ is nonzero. As the latter number is obtained by reducing the discriminant of the number field modulo p, it follows, in particular, that a positive, finite number of primes $p \in \mathbb{Z}$ ramify.

There is the following corollary. Suppose $a \in \mathbb{C}$ is integral over $\mathbb{Z}$ and $f(X) \in \mathbb{Z}[X]$ is its minimal polynomial. Then $D(f) \in \mathbb{Z}$ and, by Proposition 10.8, the prime number p divides $D(f)$ if and only if the factorization of the polynomial $\overline{f}(X) \in \mathbb{F}_p[X]$, i.e., $f(X)$ reduced modulo p, is not square-free. Moreover, if $\deg f(X) > 1$ then there is at least one prime number p such that the factorization in $\overline{f}(X) \in \mathbb{F}_p[X]$ is not square-free.

10.9 Tensor Products

Let A be a commutative ring.

10.9.1 Tensor Products of Modules

The *tensor product* of A-modules is defined by means of a *universal property.*

Let V, W, Z be A-modules. An A-*bilinear map* $\phi : V \times W \longrightarrow Z$ is a map such that it is A-linear in each of its two arguments. [42]

[40] For a general characterization of finite-dimensional K-algebras, see e.g. [130], Sec. 2.6. The proof of Theorem 10.11 given here is a special case.

[41] See e.g. [125].

[42] This extends naturally to the notion of n-multilinear maps.

Definition 10.12 (Universal Property) *Let V, W be A-modules. A tensor product of V and W is a pair $(\mathcal{U}, \iota)$ such that the following holds.*

1. *$\mathcal{U}$ is an A-module and $\iota : V \times W \longrightarrow \mathcal{U}$ is an A-bilinear map.*
2. *Let Z be an A-module and let $\psi : V \times W \longrightarrow Z$ be an A-bilinear map. Then there is a* unique *A-module morphism $f : \mathcal{U} \longrightarrow Z$ such that $\psi = f \circ \iota$.*

If a tensor product exists, it is *uniquely unique* in the following sense: if $(\mathcal{U}', \iota')$ is a tensor product of V and W as well, then there is a unique A-module isomorphism $g : \mathcal{U} \longrightarrow \mathcal{U}'$ such that $\iota' = g \circ \iota$. [43]

The existence of a tensor product is shown by a "canonical construction." This particular construction is then declared to be *the* tensor product of V and W. This makes sense due to strong equivalence as mentioned above.

Theorem 10.13 *The tensor product exists.*

We verify this theorem below. Consider the free A-module $\mathcal{M}$ generated by all pairs $(v, w) \in V \times W$, i.e., $\mathcal{M}$ consists of all *formal* sums $\sum \lambda(v, w) \cdot (v, w)$ with $(v, w) \in V \times W$ and $\lambda(v, w) \in A$ such that $\lambda(v, w) \neq 0$ for at most a finite number of terms $(v, w) \in V \times W$. By convention, $1 \cdot (v, w)$ is denoted (v, w).

Also consider the submodule $\mathcal{N}$ of $\mathcal{M}$ generated by all elementary expressions of the form

1. $(\lambda \cdot v, w) - \lambda \cdot (v, w)$,
2. $(v, \lambda \cdot w) - \lambda \cdot (v, w)$,
3. $(v + v', w) - (v, w) - (v', w)$,
4. $(v, w + w') - (v, w) - (v, w')$,

where $\lambda \in A$, $v, v' \in V$, $w, w' \in W$.

The quotient $\mathcal{M}/\mathcal{N}$ is an A-module as well. Define

$$v \otimes w := \overline{(v, w)} = (v, w) + \mathcal{N} \ \in \ \mathcal{M}/\mathcal{N},$$

a *tensor*. Each element of $\mathcal{M}/\mathcal{N}$ can be written as a finite A-sum of tensors (though such a decomposition is not unique in general!).

Set

$$V \otimes_A W = \mathcal{M}/\mathcal{N}$$

and define

$$\iota : V \times W \longrightarrow V \otimes_A W$$

$$(v, w) \mapsto v \otimes w.$$

[43] The property "uniquely unique" is stronger than the property "unique up to isomorphism". For instance, finite fields and algebraic closures satisfy the latter, weaker notion but not the former, stronger one.

DEFINITION 10.14 *$(V \otimes_A W, \iota)$ is the tensor product of the A-modules V and W.*

REMARK 10.1 If there is no ambivalence, we often drop the subscript A from the notation $V \otimes_A W$.

One sees at once that the tensor product admits the following "calculus":

THEOREM 10.15 *For all $\lambda \in A$, $v, v' \in V$, $w, w' \in W$, it holds that*

1. $(\lambda \cdot v) \otimes w = \lambda \cdot (v \otimes w)$,
2. $v \otimes (\lambda \cdot w) = \lambda \cdot (v \otimes w)$,
3. $(v + v') \otimes w = v \otimes w + v' \otimes w$,
4. $v \otimes (w + w') = v \otimes w + v \otimes w'$.

The calculus implies that ι is an A-bilinear map. It is left to argue that $(V \otimes W, \iota)$ is universal in the sense of Definition 10.12. Suppose Z is an A-module and $\psi : V \times W \longrightarrow Z$ is an A-bilinear map. If there exists an A-module morphism $f : V \otimes W \longrightarrow Z$ such that $\psi = f \circ \iota$, then it must be *unique*. Indeed, since $f(v \otimes w) = \psi(v, w)$ for all tensors $v \otimes w$ and since, as remarked above, each element of $V \otimes W$ is a finite K-sum of tensors, the morphism f is uniquely defined by its action on the tensors.

Therefore, it remains to show that a factorization $f \circ \iota$ of ψ *exists*. The map ψ defines an A-module morphism $\mathcal{M} \to Z$ by sending a term $(v, w) \in \mathcal{M}$ to $\psi(v, w) \in Z$ and by extending this A-linearly. By inspection, the kernel of this morphism contains $\mathcal{N}$. Hence, this induces an A-module morphism f on $\mathcal{M}/\mathcal{N}$ such that $f(v \otimes w) = \psi(v, w)$ for each tensor $v \otimes w$ with $v \in V$, $w \in W$. This concludes the argument.

Tensor products often create a "larger space" (as we shall see shortly). But tensor products may also cause a "collapse". Here is an example. Take the additive groups $\mathbb{Z}/2\mathbb{Z}$ and $\mathbb{Z}/3\mathbb{Z}$ as $\mathbb{Z}$-modules and let $x \otimes y \in \mathbb{Z}/2\mathbb{Z} \otimes \mathbb{Z}/3\mathbb{Z}$. Then $(x \otimes y) = (3x \otimes y) = (x \otimes 3y) = (x \otimes 0) = 0$. So the tensor product is trivial in this case.

REMARK 10.2 The definition of tensor product extends in a straightforward way to tensor products of $n \geq 2$ terms, replacing bilinear maps by n-multilinear maps and adapting the other definitions accordingly.

REMARK 10.3 We make some general remarks about the use of the tensor product.

1. The point about the tensor product $V \otimes W$ is its (unique) existence according to the universal property. The construction of the module $\mathcal{M}/\mathcal{N}$ merely serves to show that the definition is not vacuous.
2. The fact that the tensor product establishes a one-to-one correspondence between A-bilinear maps $V \times W \longrightarrow Z$ and the A-module morphisms $F : V \otimes W \longrightarrow Z$, is often used to "linearize" a "bilinear problem" by passing to a tensor product.

10.9.2 Some Basic Theory of the Tensor Product

Commuting with Direct Sums

As a first result, we note that

$$A \otimes_A A \simeq A.$$

In particular, this means that the tensor product is not trivial if A is nontrivial. We now verify the result. Consider the A-bilinear map

$$\mu : A \times A \longrightarrow A$$

$$(a, b) \mapsto ab.$$

By the universal property, there is a unique A-module morphism

$$f : A \otimes_A A \longrightarrow A$$

such that

$$\mu = f \circ \iota.$$

Each element of $A \otimes_A A$ can be represented by some tensor $a \otimes 1$ with $a \in A$. So surjectivity follows since $f(a \otimes 1) = a$ for each $a \in A$. Likewise, injectivity follows since $f(a \otimes 1) = 0$ if and only if $a = 0$ (note that $0 \otimes 1 = 0 \otimes 0$). In conclusion, f is an A-module isomorphism. By a similar argument, if M is an A-module, then

$$A \otimes M \simeq M.$$

If $\mathcal{I}$ is an index set, M_i is an A-module for each $i \in \mathcal{I}$, and N is an A-module, then it holds that

$$\left(\bigoplus_{i \in \mathcal{I}} M_i \right) \otimes N \simeq \bigoplus_{i \in \mathcal{I}} (M_i \otimes N).$$

Likewise,

$$N \otimes \left(\bigoplus_{i \in \mathcal{I}} M_i \right) \simeq \bigoplus_{i \in \mathcal{I}} (N \otimes M_i).$$

Hence:

Lemma 10.16 *If M is an A-module and N is a free A-module with basis $\{f_j\}_{j \in \mathcal{J}}$ then there is a one-to-one correspondence between the elements of $M \otimes N$ and terms of the form*

$$\sum_{j \in \mathcal{J}} m_j \otimes f_j,$$

where $m_j \in M$ for each $j \in \mathcal{J}$ and all except at most finitely many m_j's are equal to 0.

Of course, the lemma is easily adapted to the case when M is free.

REMARK 10.4 If M is free as well, with basis $\{e_i\}$, then this implies directly that $M \otimes N$ is free, with basis $\{e_i \otimes f_j\}$.

For example, let K be a field and let M, N be finite-dimensional K-vector spaces, with respective dimensions n and m. Then $M \otimes N \simeq K^{nm}$ as vector spaces. In particular, $K^n \otimes K^m \simeq K^{nm}$.

Base Extension

Let M be an A-module. Let B be a commutative extension ring of A. The A-module $B \otimes_A M$ may be viewed as a *B-module* in a natural way. Likewise, this holds for the A-module $M \otimes_A B$.

The idea is to define B-scalar multiplication on $B \otimes_A M$ simply by the assignment

$$b \cdot (b' \otimes m) = (bb' \otimes m)$$

and to extend it B-linearly to all of $B \otimes_A M$. We now sketch the proof that this works.

With a short argument that invokes the universal property twice, one shows that there is a unique A-bilinear map from $B \times (B \otimes_A M)$ to $B \otimes_A M$ that sends $(b, (b' \otimes m))$ to $(bb' \otimes m)$. This immediately renders the (unique) B-module structure on $B \otimes_A M$ that is consistent with the assignment above.

Briefly, one considers the A-trilinear map from $B \times B \times M$ to $B \otimes_A M$ that sends (b, b', m) to $(bb' \otimes m)$. By the universal property, [44] this yields a unique A-linear map from $B \otimes_A (B \otimes_A M)$ to $B \otimes_A M$. [45] Once again by the universal property, this yields the claimed (unique) A-bilinear map from $B \times (B \otimes_A M)$ to $B \otimes_A M$. [46]

Tensor Products of Algebras

Let K be a field. Let A, B be K-algebras. Then the K-vector space $A \otimes_K B$ may be viewed *as a K-algebra* in a natural way.

The idea is to induce a ring structure by defining

$$(a \otimes b) \cdot (a' \otimes b') = (aa') \otimes (bb')$$

and to extend this to all of $A \otimes_K B$. The field K is naturally embedded via

$$\lambda \mapsto \lambda \cdot (1_A \otimes 1_B)$$

and the ring multiplication alluded to above is compatible with the K-scalar

[44] applied in the case of a tensor product with three terms.

[45] The brackets may be placed on account of *associativity* of the tensor product. For instance, if M, M', M'' are A-modules, then $M \otimes M' \otimes M'' \simeq (M \otimes M') \otimes M'' \simeq M \otimes (M' \otimes M'')$.

[46] Please refer to [126], Chapter XVI, par. 4, pp. 623. Base extension is actually more general. Namely, the argument sketched here also works for each commutative ring B which may be viewed as an A-module. This includes, of course, the case we have presented, i.e., where A is a subring of B.

multiplication already enjoyed by $A \otimes_K B$. We also have a natural embedding of the K-algebra A into $A \otimes_K B$ via

$$a \mapsto (a \otimes 1_B).$$

Likewise, there is an embedding of the K-algebra B. Using a similar style of reasoning as in the case of base extension, it is not so hard to verify that this works correctly. [47]

Tensor Products of Module Morphisms

Let A be a commutative ring and let M, M', N, N' be A-modules. Suppose

$$\phi : M \longrightarrow M' \quad \text{and} \quad \psi : N \longrightarrow N'$$

are A-module morphisms. Consider the tensor product $M \otimes N$ with the natural inclusion $\iota : M \times N \longrightarrow M \otimes N$ and the tensor product $M' \otimes N'$ with the natural inclusion $\iota' : M' \times N' \longrightarrow M' \otimes N'$.

Define the assignment

$$u \otimes v \mapsto \phi(u) \otimes \psi(v),$$

for all $u \in M, v \in N$. The A-linear extension of this assignment to all of $M \otimes N$ defines an A-module morphism from $M \otimes N$ to $M' \otimes N'$, denoted $\phi \otimes \psi$. But it requires a proof that this map is indeed well-defined.

The map

$$\phi \times \psi : M \times N \longrightarrow M' \times N'$$

$$(u, v) \mapsto (\phi(u), \psi(v))$$

is an A-bilinear map. Composition with the natural inclusion ι' yields the A-bilinear map

$$\iota' \circ (\phi \times \psi) : M \times N \longrightarrow M' \otimes N'.$$

By the universal property, it follows at once that, as desired, there is a unique A-linear map

$$F : M \otimes N \longrightarrow M' \otimes N'$$

such that

$$\iota' \circ (\phi \times \psi) = F \circ \iota$$

and

$$F(u \otimes v) = \phi(u) \otimes \psi(v)$$

for all $u \in M, v \in N$.

In the special case where M, N are free A-modules of finite rank, then

$$\mathrm{End}_A(M) \otimes_A \mathrm{End}_A(N) \simeq \mathrm{End}_A(M \otimes_A N),$$

[47] Please refer to [126], Chapter XVI, par. 6, pp. 629–631.

where $\mathrm{End}_A(M)$ (resp., $\mathrm{End}_A(N)$) denotes the endomorphism ring of M (resp., N).

The following lemma is straightforward by the discussion above. Let M'', N'' be A-modules. Suppose

$$\phi' : M' \longrightarrow M'' \text{ and } \psi' : N' \longrightarrow N''$$

are A-module morphisms.

LEMMA 10.17 (COMPOSITION) *It holds that*

$$(\phi' \otimes \psi') \circ (\phi \otimes \psi) = (\phi' \circ \phi) \otimes (\psi' \circ \psi).$$

In particular, if ϕ, ψ are isomorphisms, then so is $\phi \otimes \psi$. Moreover,

$$(\phi \otimes \psi)^{-1} = \phi^{-1} \otimes \psi^{-1}.$$

PROOF The first claim follows at once from the definition. As to the remaining claims, just apply $\phi^{-1} \otimes \psi^{-1}$ after $\phi \otimes \psi$ (resp., vice-versa). This gives the identity map on $M \otimes N$ (resp., $M' \otimes N'$). □

See [126] for a generalization. [48]

10.9.3 Examples

Tensoring-Up of a Finite Abelian Group

Let G be a finite abelian group, viewed as a $\mathbb{Z}$-module. Let A be a commutative extension ring of $\mathbb{Z}$ that is free of rank n as a $\mathbb{Z}$-module. Select a $\mathbb{Z}$-basis $\{e_i\}_{i=1}^n$ of A. The elements of the $\mathbb{Z}$-module $G \otimes A$ are in one-to-one correspondence with the expressions of the form $\sum_{i=1}^n g_i \otimes e_i$ with $g_1, \ldots, g_n \in G$. In particular, $G \otimes A$ is finite and it has cardinality $|G|^n$, irrespective of the cardinality of A.

By base extension, this is an A-module. If $a \in A$ then

$$a \cdot (\sum_{i=1}^n g_i \otimes e_i) = \sum_{i=1}^n (g_i \otimes ae_i) = \sum_{i=1}^n g_i' \otimes e_i,$$

for some $g_1', \ldots, g_n' \in G$. The g_i''s in the latter equality are $\mathbb{Z}$-linear combinations of the g_i's. The $\mathbb{Z}$-coefficients only depend on a and the $e_i \cdot e_j$'s and are easily obtained once a and the $e_i \cdot e_j$'s are expressed as $\mathbb{Z}$-coordinate vectors with respect to the basis. This turns A-scalar multiplication into "linear algebra" (over $\mathbb{Z}$, that is).

Tensoring-Up of a Vector Space

In the second example, let K be a field and let V be a K-vector space of finite dimension n. If L is an extension field of K, then L is a K-vector space. By base extension, $L \otimes_K V$ can be viewed as an L-vector space. We claim that

[48] Ch. XVI, par. 1, pp. 605–606.

$\dim_L L \otimes V = n$. So, in a sense, the vector space is just "scaled up" to include, more generally, L-multiples of vectors, while the dimension does not change as a result. Moreover, V as a K-vector space is embedded in a natural way.

This can be verified as follows. Let $\{\mathbf{e}_i\}_{i=1}^n$ be a K-basis for V. We have already seen that there is a one-to-one correspondence between the elements of $L \otimes V$ and the elements of the form $\sum_i \lambda_i \otimes \mathbf{e}_i$. By base extension, this is an L-vector space and we may equivalently state that the one-to-one correspondence is with elements of the form $\sum_i \lambda_i \cdot (1 \otimes \mathbf{e}_i)$ instead. The claim follows at once. The K-vector space V is naturally embedded via $v \mapsto 1 \otimes v$.

Tensoring-Up of an Algebra

Let K be a field.

1. Let L be an extension field of K. Then
$$L \otimes_K K[X] \simeq L[X]$$
as L-algebras.
2. If $f(X) \in K[X]$ is a polynomial of degree n and if K' is an extension field of K, then
$$K' \otimes_K (K[X]/(f(X) \cdot K[X])) \simeq K'[X]/(f(X) \cdot K'[X]),$$
as K'-algebras.
3. Let $\overline{K}$ be an algebraic closure of K. Let L be a finite-degree separable field extension of K. Then
$$\overline{K} \otimes_K L \simeq \overline{K}^{[L:K]}$$
as $\overline{K}$-algebras.

We briefly comment on the first claim. There is one-to-one-correspondence between polynomials $f(X) = \sum_{i=0}^{\ell} \lambda_i X^i \in L[X]$ and expressions of the form $\sum_{i=0}^{\ell} \lambda_i \otimes X^i$. This gives a K-vector space isomorphism. Ring multiplication in $L \otimes_K K[X]$ mimics exactly the multiplication in $L[X]$ as well. Applying base extension and viewing L as embedded via $\lambda \mapsto (\lambda \otimes 1)$, it follows there is an L-algebra isomorphism.

For the second we do not give a proof; it follows from the first, in combination with some further results on tensor products. [49]

As to the third claim, this is implied by the second example in combination with the following observations. Write $L = K(x)$ for some $x \in L$ (which is possible by the assumptions on L) and take $f(X) \in K[X]$ as the minimal polynomial of x. When viewed as an element of $\overline{K}[X]$, it factors as a product of $[L : K]$ distinct linear polynomials. Then apply the Chinese Remainder Theorem (CRT) and use the fact that a polynomial ring over a field modulo an ideal generated by a linear polynomial is the field itself. This way it follows that $\overline{K}[X]/(f(X)) \simeq \overline{K}^{[L:K]}$.

[49] Specifically, exactness properties. See e.g. [2].

11

Secret Sharing

Contents

11.1 Introduction

A (t, n)-threshold secret sharing scheme, where t, n are integers with $0 \leq t < n$, provides a means to "disperse" secret data s into a sequence of n pieces of data, called *shares*, such that any $\leq t$ shares jointly give *no information* on the secret s (*t-privacy*), whereas any $\geq t + 1$ of these shares jointly determine the secret s *uniquely* (*$(t + 1)$-reconstruction*).

Thus, an adversary's uncertainty about the secret is not reduced by gaining access to t shares: from the point of view of the adversary, the a priori uncertainty about the secret is equal to the a posteriori uncertainty about the secret. In particular, if the adversary had no information before accessing t shares, then the adversary still has no information afterwards as a result. Moreover, if the adversary somehow manages to cause the loss (erasure) of t' shares, where $n-t' \geq t + 1$, then the secret can still be reconstructed uniquely from $t + 1$ remaining shares.

In the case $t = 0$, there is a trivial scheme by declaring each share to be equal to the secret. [1] The case $1 \leq t = n - 1$ is more interesting. It has an elegant solution based on an idea similar to that underlying the *one-time pad* encryption scheme. Say that the secret s is encoded as an element of a given finite group G. Note that the group as such is *not* secret. Then the n shares $(s_1, \dots, s_n) \in G^n$ are selected uniformly random subject to the condition that their product satisfies

$$s_1 \cdots s_n = s.$$

This works, for instance, by selecting

$$(\rho_1, \dots, \rho_{n-1}) \in G^{n-1}$$

uniformly at random and independently of the secret s and by defining the shares as

$$s_1 = \rho_1, \dots, s_{n-1} = \rho_{n-1}\ ,\ s_n = \rho_{n-1}^{-1} \cdots \rho_1^{-1} \cdot s.$$

Note that

$$s_1 \cdots s_n = \rho_1 \cdots \rho_{n-1} \cdot \left(\rho_{n-1}^{-1} \cdots \rho_1^{-1} \cdot s\right) = s$$

and that any vector of $n - 1$ shares has the uniform distribution on G^{n-1} and is

[1] We find it convenient to allow this "pathetic" case $t = 0$. However, we do not find it convenient to allow the other pathetic case $t = n$.

independently distributed of the secret s. If G is Abelian and if we use additive notation, then we may, of course, define the n-th share instead as

$$s_n = s - \left(\sum_{i=1}^{n-1} s_i \right),$$

which gives

$$s = \sum_{i=1}^{n} s_i.$$

This scheme is sometimes referred to as the *n-out-of-n secret sharing scheme.* The cases $1 \leq t < n-1$ require additional ideas, which we explain later. Note that the process of creating shares is *probabilistic*, i.e., besides the secret itself it requires (secret) "coin flips".

Secret sharing was invented in 1979 by Adi Shamir [163], and independently at the same time by Bob Blakley [24], to alleviate the single-point-of-failure problem for storage of secret data. Their respective solutions work for any threshold. Meanwhile, secret sharing has become a fundamental cryptographic primitive with a host of applications, most notably in threshold cryptography and secure multi-party computation, as we have seen in Part I.

Let K be a field. Let $p > 0$ be a prime number and let v be a positive integer. Define $q = p^v$. Let $\mathbb{F}_q$ be a finite field with q elements.

11.2 Notation

Let $\mathcal{F}$ be a nonempty set and let $\mathcal{I}$ be a nonempty finite set. Then $\mathcal{F}[\mathcal{I}]$ denotes the set of functions

$$x : \mathcal{I} \longrightarrow \mathcal{F}.$$

If $B \subset \mathcal{I}$ is nonempty, then the *projection function*

$$\pi_B : \mathcal{F}[\mathcal{I}] \longrightarrow \mathcal{F}[B]$$

maps each function $x \in \mathcal{F}[\mathcal{I}]$ simply to its restriction to the subdomain B. In other words, the function $\pi_B(x) \in \mathcal{F}[B]$ is defined such that it agrees with the function x on all of B. The dependence of projection maps on $\mathcal{F}$ and $\mathcal{I}$ is suppressed in the notation, as these will always be clear from the context. For each set $B \subset \mathcal{I}$, its *complement* is defined as $\overline{B} = \mathcal{I} \setminus B$.

We may identify $\mathcal{F}[\mathcal{I}]$ with the Cartesian-product of sets $\Omega := \prod_{i \in \mathcal{I}} \mathcal{F}$ in the obvious way. We say that $\mathcal{F}$ is the *alphabet* and $\mathcal{I}$ is the *index set.* For each $\mathbf{x} \in \Omega$, we write $\mathbf{x} = (x_j)_{j \in \mathcal{I}}$, the *coordinate vector.* Thus, $\mathbf{x} \in \Omega$ corresponds to $x \in \mathcal{F}[\mathcal{I}]$ such that $x(i) = x_i$ for all $i \in \mathcal{I}$. Suppose $B \subset \mathcal{I}$ is non-empty. For each $\mathbf{x} \in \Omega$, we may say that $\pi_B(\mathbf{x})$ corresponds to $(x_i)_{i \in B} \in \prod_{i \in B} \mathcal{F}$. As a shorthand for $\pi_B(\mathbf{x})$, the notation $\mathbf{x}_B$ is sometimes used.

For an integer $\ell \geq 1$, the notation $\mathcal{F}^\ell$ refers to the ℓ-fold Cartesian product of sets $\mathcal{F}\times\cdots\times\mathcal{F}$, leaving any index set implicit. When we say that $\mathcal{F}^\ell$ is *indexed* by a set $\mathcal{I}$ we mean that $|\mathcal{I}| = \ell$ and that the coordinates are labeled by $\mathcal{I}$ in some way. Often, an index set $\mathcal{I}$ is a subset of the integers. This way, a coordinate vector can be made explicit by an ordered tuple of the form $(\cdot, \ldots, \cdot)$, under the convention that "the indices increase monotonically from left to right." This is convenient in some cases. When speaking of a projection function $\pi_B : \mathcal{F}^\ell \longrightarrow \mathcal{F}^{|B|}$, we tacitly assume that the image $\mathcal{F}^{|B|}$ is indexed by B.

In the context of linear algebra, specifically, in expressions involving multiplication of a matrix by a vector, it is convenient to treat a vector $\mathbf{x}$ as a *column vector*. The associated row vector, i.e., its *transpose*, is denoted $\mathbf{x}^T$. Likewise, the transpose of a matrix M is denoted M^T. For example, if M is square matrix (over say, some given ring) and $\mathbf{x}$ is a vector (defined over the same ring and of the appropriate length), then we have the products $\mathbf{x}^T M$ and $M\mathbf{x}$.

11.3 Interpolation Codes

Shamir's scheme was introduced in Chapter 3. We will explain the scheme in a different way, from a combinatorial perspective based on *interpolation codes*. This will help us generalize the approach later.

11.3.1 Definitions

Definition 11.1 (Code) *A* code *of length $\ell \geq 1$ over an alphabet $\mathcal{F}$ is a nonempty set $C \subset \mathcal{F}^\ell$. The elements of C are (*code*)*words*.

Indexing $\mathcal{F}^\ell$ with some set $\mathcal{I}$, we may equivalently say that the code C is a nonempty set of functions $x : \mathcal{I} \longrightarrow \mathcal{F}$.

Definition 11.2 (Interpolation Code) *Let C be a code of length $\ell \geq 1$ over a* finite *alphabet $\mathcal{F}$ of cardinality $\bar{q} \geq 2$. Write $\mathcal{I}$ for an index set of $\mathcal{F}^\ell$. Let r be an integer with $1 \leq r \leq \ell$. Then C is an $(r, \ell, \bar{q})$-*interpolation code *if, for each $B \subset \mathcal{I}$ with $|B| = r$, the projection map*

$$\pi_B : C \longrightarrow \mathcal{F}^r,$$

$$\mathbf{x} \mapsto \pi_B(\mathbf{x})$$

is bijective*. The class of all $(r, \ell, \bar{q})$-interpolation codes is denoted* $\mathrm{IC}(r, \ell, \bar{q})$.

Equivalently, it is required that there is a *unique* $\mathbf{x} \in C$ with $\mathbf{x}_B = \mathbf{w}$ for each pair $(B, \mathbf{w})$ with $B \subset \mathcal{I}$, $|B| = r$, and $\mathbf{w} \in \mathcal{F}^r$. Note that if C is an $(r, \ell, \bar{q})$-interpolation code, then $|C| = \bar{q}^r$.

Remark 11.1 Interpolation codes coincide with the *orthogonal arrays* $\mathrm{OA}(r, \ell, \bar{q})$, an extensively studied classical notion in combinatorics. See for instance [178, 105]. For reasons to become clear shortly, the nomenclature used here suits our purposes better.

11.3.2 Some Basic Properties

In the "extreme cases" $r = 1, \ell$ the existence of interpolation codes is trivial to analyze. Also, the case $r = \ell - 1$ is easy.

Lemma 11.3 *Let $\ell, \bar{q}$ be integers with $\ell \geq 1$ and $\bar{q} \geq 2$. Then the following holds.*

1. $\mathrm{IC}(1, \ell, \bar{q}) \neq \emptyset$.
2. $\mathrm{IC}(\ell - 1, \ell, \bar{q}) \neq \emptyset$ *if* $\ell \geq 2$.
3. $\mathrm{IC}(\ell, \ell, \bar{q}) \neq \emptyset$.

Proof As to the first claim, $C \in \mathrm{IC}(1, \ell, \bar{q})$ if and only if any matrix whose rows are in one-to-one correspondence with the words of C has the property that, for each column, each of the $\bar{q}$ elements of the alphabet over which C is defined occurs exactly once in that column. As to the third claim, $C \in \mathrm{IC}(\ell, \ell, \bar{q})$ if and only if C is equal to the ℓ-fold Cartesian product over its alphabet. As to the second claim, take the additive group of the ring $\mathbb{Z}/\bar{q}\mathbb{Z}$ of integers modulo $\bar{q}$ as the alphabet. Define the code C of length ℓ over this alphabet as the code consisting of all words $(g_1, \ldots, g_\ell) \in (\mathbb{Z}/\bar{q}\mathbb{Z})^\ell$ such that $g_1 = g_2 + \cdots + g_\ell$. Then $C \in \mathrm{IC}(\ell - 1, \ell, \bar{q})$. □

It is a priori not clear for which other parameters $r, \ell, \bar{q}$ it holds that $\mathrm{IC}(r, \ell, \bar{q}) \neq \emptyset$. The following straightforward reduction sometimes aids inductive proofs.

Lemma 11.4 *Suppose* $\mathrm{IC}(r, \ell, \bar{q}) \neq \emptyset$. *Let k be an integer.*

1. (Puncturing) If $0 \leq k \leq \ell - r$, *then* $\mathrm{IC}(r, \ell - k, \bar{q}) \neq \emptyset$.
2. (Shortening) If $0 \leq k \leq r - 1$, *then* $\mathrm{IC}(r - k, \ell - k, \bar{q}) \neq \emptyset$.

Proof The claims are trivial if $k = 0$. So assume $k > 0$. Let $C \in \mathrm{IC}(r, \ell, \bar{q})$ and write $\mathcal{I}$ for an index set of $\mathcal{F}^\ell$. The first claim is immediate by *puncturing*: select a set $B \subset \mathcal{I}$ with $|B| = \ell - k$ and take the code $\pi_B(C)$. Equivalently, remove all of the $\overline{B}$-indexed coordinates. Since $1 \leq k \leq \ell - r$, the resulting code of length $\ell - k \geq r$ is $(r, \ell - k, \bar{q})$-interpolating. The second claim follows by *shortening*. Select a pair $(B, \mathbf{u})$ such that $B \subset \mathcal{I}$ with $|B| = k$ and $\mathbf{u} \in \mathcal{F}^k$. Take the subcode consisting of all $\mathbf{x} \in C$ with $\mathbf{x}_B = \mathbf{u}$ and remove all B-indexed coordinates. Since $1 \leq k \leq r - 1$, this code is $(r - k, \ell - k, \bar{q})$-interpolating. □

11.4 Secret Sharing from Interpolation Codes

Interpolation codes can be used to construct (t, n)-secret sharing schemes, as we now explain.

11.4.1 The Scheme and Its Analysis

Suppose C is a $(t + 1, n + 1, \bar{q})$-interpolation code such that $0 \leq t < n$. Write $\mathcal{F}$ for the alphabet and take $\mathcal{I} = \{0, 1 \ldots, n\}$ as an index set for $\mathcal{F}^{n+1}$.

Then C gives rise to a (t, n)-threshold secret sharing scheme where the secret can be chosen arbitrarily in $\mathcal{F}$ and where each share consists of a single element of $\mathcal{F}$, as follows. The code C as such is *public knowledge.*

1. Let $s \in \mathcal{F}$ be the secret.
2. Select $\mathbf{s} = (s_0, s_1, \ldots, s_n) \in C$ uniformly at random conditioned on $s_0 = s$.
3. The n shares are set to $(s_1, \ldots, s_n) \in \mathcal{F}^n$.

Here is an analysis of this scheme. Write $\mathcal{I}^* = \mathcal{I} \setminus \{0\}$. Consider, for each $B \subset \mathcal{I}$ with $B \neq \emptyset$, the map

$$\pi'_B : C \longrightarrow \mathcal{F}^{|B|},$$

$$\mathbf{s} \mapsto \mathbf{s}_B.$$

If $|B| = t + 1$, then this map is bijective by definition of the interpolation property. If $|B| = t + 1$ and $0 \in B$, then, for each $s \in \mathcal{F}$, there are exactly $|\mathcal{F}|^t \geq 1$ words $\mathbf{s} \in C$ such that $s_0 = s$. Thus, the scheme is *well-defined*, i.e., given any $s \in \mathcal{F}$ there is indeed $\mathbf{s} \in C$ such that $s_0 = s$.

As for $(t+1)$-*reconstruction*, since $\mathbf{s}_B$ determines $\mathbf{s}$ uniquely for all sets $B \subset \mathcal{I}$ with $|B| = t+1$, this holds in particular for all sets $B \subset \mathcal{I}^*$ with $|B| = t+1$. Note that, since $t < n$, there is at least one such set B, and hence the shares jointly determine the secret uniquely.

Finally, t-*privacy* can be argued as follows. The case $t = 0$ is not interesting as any single share in this scheme determines the secret uniquely. So assume $t \geq 1$. Fix a pair (B, s) arbitrarily such that $B \subset \mathcal{I}^*$, $|B| = t$, and $s \in \mathcal{F}$. Consider the map

$$\pi'_{\{0\} \cup B} : C \longrightarrow \mathcal{F} \times \mathcal{F}^t,$$

$$\mathbf{s} \mapsto (s_0, \mathbf{s}_B).$$

This is well-defined since $t \geq 1$, and it is bijective as argued above. Therefore, when fixing $s_0 \in \mathcal{F}$ arbitrarily, it induces a bijection between $\mathcal{F}^t$ and the subset of words $\mathbf{s} \in C$ with $s_0 = s$. It follows that $\mathbf{s}_B$ is distributed according to the uniform distribution on $\mathcal{F}^t$ if $s \in \mathcal{F}$ is fixed and if $\mathbf{s} \in C$ is chosen uniformly at random such that $s_0 = s$. Therefore, in particular, the distribution of $\mathbf{s}_B$ does not depend on the secret s. This independence means that any t shares jointly give no information on the secret, or rather, they do not decrease the uncertainty about the secret.

To be more precise, consider the following *Gedanken Experiment*, still assuming that $t \geq 1$. It is a "game" played between a *challenger* and a computationally unbounded party, the *distinguisher.*

1. The distinguisher selects a pair (B, s) such that $B \subset \mathcal{I}^*$, $|B| = t$, and $s \in \mathcal{F}$.
2. The distinguisher reveals his choice (B, s).
3. Secretly, the challenger proceeds as follows.

- Select a *challenge bit* $b \in \{0,1\}$ uniformly at random.
- If $b = 0$, perform secret-sharing. Select $\mathbf{s} \in C$ uniformly at random conditioned on $s_0 = s$ and set $\mathbf{u} = \mathbf{s}_B \in \mathcal{F}^t$.
- If $b = 1$, refrain from secret-sharing. Instead select $\mathbf{u} \in \mathcal{F}^t$ uniformly at random.
- Reveal $\mathbf{u}$ to the distinguisher.

4. The distinguisher selects a *guessing bit* $\widehat{b} \in \{0,1\}$ and reveals it to the challenger.
5. Declare that the distinguisher *wins* if $\widehat{b} = b$. Otherwise the distinguisher *loses*.

The distinguisher is not "psychic" so his best strategy is to select a guessing function

$$g \ : \ \mathcal{F}^t \longrightarrow \{0,1\},$$

$$\mathbf{u} \mapsto \widehat{b}$$

that optimizes its chances, [2] given its initial choice (B, s). By the observation above, the distribution of $\mathbf{u}$ in the case $b = 0$ is the same distribution as in the case $b = 1$ (namely, the uniform distribution on $\mathcal{F}^t$). This implies that the guessing bit $\widehat{b}$ is distributed independently from the challenge bit b. Since b has the uniform distribution on $\{0,1\}$, the probability that $b = \widehat{b}$ is $1/2$, *no matter how the distinguisher plays.* So, it is not possible to gain any advantage over "blind random guessing."

This means that the distinguisher cannot distinguish a "world" where he was handed t shares "arising from a correct secret-sharing" from a "world" where he was just handed "random garbage." Hence, we may say that the *a posteriori uncertainty* that a "t-bounded adversary" has about the secret is equal to the *a priori uncertainty* the adversary has about the secret *before* any secret sharing at all took place.

At this point we will not formally summarize our findings about secret sharing in a theorem. This will be done later on, after we have given a general and formal definition of secret sharing.

Finally, note that Lemma 11.3 in combination with the construction above gives an $(n-1, n)$-threshold secret sharing scheme over an alphabet of cardinality $\bar{q}$, whenever $n, \bar{q} \geq 2$. Note that it also gives an (uninteresting) $(0, n)$-threshold secret sharing scheme. This formalizes the schemes sketched in Section 11.1. More interesting schemes will follow after a closer look into the existence of interpolation codes.

[2] Since there is no bound on the computational resources of the adversary, the adversary's best strategy here is deterministic. Indeed, for any probabilistic strategy such an adversary can simply determine in advance which particular sequence of coin flips in this strategy gives the highest chance of success.

11.4.2 A Variation: Multi-Secret Sharing

When given a $(t+k, n+k, \bar{q})$-interpolation code where n, t, k are integers such that $t \geq 0$, $k \geq 1$ and $t+k \leq n$, there is an alternative scheme that offers an interesting trade-off. [3] Write $\mathcal{F}$ for the alphabet.

Namely, instead of having secrets in $\mathcal{F}$, the secrets can be taken in $\mathcal{F}^k$ instead. This is done by designating k special indices, say $0, \ldots, k-1$, and selecting a random word such that the vector defined by those k coordinates equals the secret. This is well-defined for reasons similar as before. This increases the "capacity" of the scheme in that the amount of data that can be secret-shared in a single execution of the scheme has increased sbstantially, while at the same time, each of the shares still sits in $\mathcal{F}$.

However, there are also two drawbacks to this variation. First, the maximum number of players that can be handled in a scheme based on a given code decreases by $k-1$. Second, although there is t-privacy, there will be $(t+k)$-reconstruction, rather than $(t+1)$-reconstruction. Hence, there is a gap of k between privacy and reconstruction. In particular, if $k > 1$, it is no longer a threshold scheme.

11.5 Existence of Interpolation Codes

We shall investigate the existence of interpolation codes more thoroughly.

So far we have only discussed the existence of interpolation codes in three extreme (and trivial) cases $r \in \{1, \ell-1, \ell\}$ in Lemma 11.3 and two corresponding (t, n)-threshold secret sharing schemes with $t = 0$ (from the case $r = 1$) or $t = n-1$ (from the case $r = \ell - 1$). From the case $r = \ell$ no threshold secret sharing scheme was derived.

We first construct interpolation codes for the whole range of r (and not just the extremes), under some conditions on the alphabet. These codes are obtained as *evaluation codes* and their analysis is based on the classical Lagrange Interpolation Theorem, given below in a somewhat more modern form.

By combining these interpolation codes with the threshold secret sharing scheme from Section 11.4, we will obtain Shamir's (t, n)-threshold secret sharing scheme, which handles the full range $0 \leq t < n$.

11.5.1 Lagrange's Interpolation Theorem

Definition 11.5 *Let $m \geq 0$ be an integer. Then $K[X]_{\leq m}$ denotes the K-vector space consisting of the polynomials $f(X) \in K[X]$ with* $\deg(f) \leq m$.[4]

[3] The construction we present is actually a special case of a more general construction for so-called *ramp schemes* developed in [137, 25, 121] during the early 1980s. In [96] this idea was exploited as part of a powerful application to secure multi-party computation with low amortized complexity.

[4] Recall that if $f(X) \equiv c$, where $c \in K$, then $\deg(f) = 0$ if $c \neq 0$ and $\deg(f) = -\infty$ if $c = 0$.

THEOREM 11.6 *(Lagrange) Let $m \geq 1$ be an integer. Suppose $\alpha_1, \ldots, \alpha_m \in K$ are pairwise distinct. Then the* evaluation map

$$\mathcal{E} : K[X]_{\leq m-1} \longrightarrow K^m,$$

$$f(X) \mapsto (f(\alpha_1), \ldots, f(\alpha_m)),$$

is an isomorphism of K-vector spaces.

A more classical formulation would state that if $(\alpha_1, y_1), \ldots, (\alpha_m, y_m)$ are arbitrary points in the plane K^2 such that $\alpha_1, \ldots, \alpha_m$ are pairwise distinct, then there exists a unique polynomial $f(X) \in K[X]$ such that $\deg(f) \leq m - 1$ and such that "the graph of $f(X) = Y$ passes through these m points," i.e., $f(\alpha_i) = y_i$ for $i = 1, \ldots, m$.

PROOF It is clear that $\mathcal{E}$ is a morphism of K-vector spaces. To show that the morphism $\mathcal{E}$ is an isomorphism, we show it is surjective and injective. If $m = 1$, then $\mathcal{E}$ is the identity map: for each $y_1 \in K$, the unique polynomial f of degree at most 0 such that $\mathcal{E}(f) = y_1$ is the constant polynomial $f(X) \equiv y_1$.

Suppose $m > 1$. As the dimension of the K-vector space on the left equals that of the one on the right (namely, each equals m), by linear algebra it suffices to prove that the morphism $\mathcal{E}$ is injective. If $\mathcal{E}(f) = \mathbf{0}$, then $f(X)$ has m distinct zeros, while its degree is at most $m - 1$. As the number of zeros of a polynomial (over a field) of non-negative degree is at most its degree, it follows that the polynomial identity $f(X) = 0$ holds. Hence, $\mathcal{E}$ is injective. This concludes the proof.

For later use it is convenient to construct the inverse of $\mathcal{E}$ explicitly. If $m = 1$, then $\mathcal{E}^{-1}$ is the identity map. Suppose $m > 1$. For $i = 1, \ldots, m$, define the polynomial

$$\delta_i(X) = \prod_{1 \leq k \leq m, k \neq i} \frac{X - \alpha_k}{\alpha_i - \alpha_k} \in K[X].$$

Note that this is well-defined since $\alpha_i - \alpha_k \neq 0$ if $i \neq k$. It is verified immediately that the polynomial $\delta_i(X)$ satisfies the following properties ($i = 1, \ldots, m$).

1. $\deg(\delta_i) = m - 1$.
2. $\delta_i(\alpha_i) = 1$.
3. $\delta_i(\alpha_j) = 0$ if $1 \leq j \leq m$ and $j \neq i$.

We claim that the inverse $\mathcal{E}^{-1}$ is given by

$$\mathcal{E}^{-1} : K^m \longrightarrow K[X]_{\leq m-1},$$

$$(y_1, \ldots, y_m) \mapsto \sum_{i=1}^{m} y_i \cdot \delta_i(X).$$

This is well-defined since the polynomial on the right has degree at most $m - 1$.

To verify the claim, let $f(X) \in K[X]_{\leq m-1}$. Then

$$f(X) = \sum_{i=1}^{m} y_i \cdot \delta_i(X) \in K[X],$$

where

$$y_1 := f(\alpha_1), \dots, y_m := f(\alpha_m).$$

Indeed, for $j = 1, \dots, m$, we have

$$f(\alpha_j) = \sum_{i=1}^{m} y_i \cdot \delta_i(\alpha_j) = y_j \cdot 1 + \sum_{i \neq j} y_i \cdot 0 = y_j.$$

□

We will give alternative proofs in Section 11.6. In Section 11.13 we will generalize this result to commutative rings.

Notation being as in Theorem 11.6 and its proof, the following corollary is helpful later on.

Corollary 11.7 *Let $a \in K$. Then there exists a unique K-linear form $\phi : K^m \to K$ such that, for all $f(X) \in K[X]_{m-1}$, it holds that*

$$\phi(f(\alpha_1), \dots, f(\alpha_m)) = f(a).$$

Proof The claim is obvious if $m = 1$. Suppose $m > 1$. If each of ϕ, ϕ' satisfies the requirement, then $\phi - \phi'$ is identically 0 on K^m, as the image of $\mathcal{E}$ is equal to K^m. Therefore, ϕ, ϕ' are identical as K-linear forms. This settles uniqueness. As to existence, define ϕ as follows. For each $\mathbf{y} \in K^m$, $a \in K$, define

$$\phi(\mathbf{y}) := f(a),$$

where

$$f = \mathcal{E}^{-1}(\mathbf{y}).$$

This is well-defined, as $\mathcal{E}$ is invertible. Moreover, this is a K-linear form that satisfies the requirement.

The coefficients of this form can be given explicitly, as follows.

$$\phi : K^m \longrightarrow K$$

$$(Y_1, \dots, Y_m) \mapsto \sum_{i=1}^{m} \delta_i(a) \cdot Y_i$$

with $\delta_i(X) \in K[X]$ $(i = 1, \dots, m)$ as defined as in the proof of Theorem 11.6. □

Definition 11.8 (Linear Code) *An* $\mathbb{F}_q$-linear code *of length $\ell \geq 1$ is a subspace C of the $\mathbb{F}_q$-vector space $\mathbb{F}_q^{\ell}$. Its* dimension $\dim(C)$ *is its dimension as a subspace.*

Note that $0 \in C$, so $C \neq \emptyset$.

An immediate consequence of Lagrange's Interpolation Theorem for the existence of (linear) interpolation codes is as follows.

THEOREM 11.9 *Suppose r, ℓ are integers such that $1 \leq r \leq \ell \leq q$. Then there exists an $\mathbb{F}_q$-linear (r, ℓ, q)-interpolation code.*

PROOF Select pairwise distinct elements $\alpha_1, \ldots, \alpha_\ell \in \mathbb{F}_q$. This is possible since $\ell \leq q$. Consider the *polynomial evaluation code*

$$C = \{(f(\alpha_1), \ldots, f(\alpha_\ell)) \mid f(X) \in \mathbb{F}_q[X], \deg(f) \leq r-1\} \subset \mathbb{F}_q^\ell.$$

This code is $\mathbb{F}_q$-linear since the evaluation map is a morphism of vector spaces and C is its image. Fix $\mathcal{I} = \{1, \ldots, \ell\}$ as the index set. Select an arbitrary pair $(B, \mathbf{v})$ with

$$B \subset \{1, \ldots, \ell\} \ , \ |B| = r \text{ and } \mathbf{v} = (v_i)_{i \in B} \in \mathbb{F}_q^r.$$

This is possible since $r \leq \ell$. By Theorem 11.6, there exists a unique polynomial $g(X) \in \mathbb{F}_q[X]$ such that

$$\deg(g) \leq r-1 \text{ and } g(\alpha_i) = v_i,$$

for all $i \in B$. Hence, there is a unique word $\mathbf{x} \in C$ such that $\mathbf{x}_B = \mathbf{v}$, namely,

$$\mathbf{x} = (g(\alpha_1), \ldots, g(\alpha_\ell)).$$

□

In Section 11.7 we will relax the condition to $1 \leq r \leq \ell \leq q+1$ by exploiting an extra evaluation point that was not visible so far, namely "the point at infinity".

11.5.2 Shamir's Scheme

Shamir's Threshold Secret Sharing Scheme is obtained by instantiating the threshold secret sharing scheme based on interpolation codes with the codes from Theorem 11.9.

Concretely, Shamir's (t, n)-threshold scheme is as follows. Suppose $q > n$. Let t, n be integers with $0 \leq t < n$. Let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ be pairwise distinct and non-zero. This makes sense since $q > n$. *Note:* $t, n, q, \alpha_1, \ldots, \alpha_n$ are *public data.*

- Write $s \in \mathbb{F}_q$ for the secret.
- Select a polynomial $f(X) \in \mathbb{F}_q[X]$ uniformly at random, conditioned on
  1. $\deg(f) \leq t$.
  2. $f(0) = s$.

  Note that this is the same as selecting a coefficient vector

  $$(a_0, a_1, \ldots, a_t) \in \mathbb{F}_q^{t+1},$$

  uniformly at random, conditioned on $a_0 = s$, and setting

  $$f(X) = s + a_1 X + \ldots + a_t X^t \in \mathbb{F}_q[X].$$

- The n shares in the secret s are then as follows:

$$s_1 = f(\alpha_1) \in \mathbb{F}_q, \ldots, s_n = f(\alpha_n) \in \mathbb{F}_q.$$

This scheme satisfies t-privacy and $(t+1)$-reconstruction with $t+1 \leq n$.

Remark 11.2 Using Corollary 11.7 and its proof, it follows how the secret s can be reconstructed from any $t+1$ shares s_i. Indeed, for each set $B \subset \{1, \ldots, n\}$ with $|B| = t+1$ there exists a linear form ϕ^B such that $\phi^B((s_i)_{i \in B})$ is equal to the secret with probability 1. From the Proof of Corollary 11.7, it follows that

$$s = \phi^B((s_i)_{i \in B}) = \sum_{i \in B} \delta_{i,B} \cdot s_i,$$

where, for each $i \in B$,

$$\delta_{i,B} = \prod_{j \in B, j \neq i} \frac{-\alpha_j}{\alpha_i - \alpha_j} \in \mathbb{F}_q.$$

11.6 Alternative Proofs of Lagrange's Theorem

Several key ideas in basic secret sharing are derived from Lagrange's Interpolation Theorem, variations thereof (for instance, over number fields), or more advanced generalizations of it (such as implied by the Riemann-Roch Theorem for algebraic function fields). Therefore it makes sense to present alternative proofs of Lagrange's Interpolation Theorem in its most basic form.

11.6.1 Proof by Chinese Remainder Theorem

Lagrange's Interpolation Theorem can also be viewed as a special case of the Chinese Remainder Theorem (CRT). We will verify that this is indeed the case.

For the statement of the CRT, please refer to Theorem 10.3. We start by including a proof sketch of the CRT for comparison with our first proof of Lagrange Interpolation. Let A be a commutative ring. By definition, ideals I, J in A co-prime if $I + J = A$. Restating the definition, ideals I, J of A are co-prime if and only if there is $(x, y) \in I \times J$ such that $x + y = 1$, or equivalently, there is $y \in A$ such that $y \equiv 1 \bmod I$ and $y \equiv 0 \bmod J$. This extends trivially to more than two pair-wise co-prime ideals. With this reformulation in mind, under the hypothesis of the CRT, it is straightforward to craft $x_1, \ldots, x_m \in A$ such that, for each x_j $(j = 1, \ldots, m)$, it holds that $x_j \equiv 1 \bmod I_j$ and $x \equiv 0 \bmod I_k$ if $k \neq j$.

If we consider the map defined by the assignment $a \mapsto (a \bmod I_1, \ldots, a \bmod I_m)$, then we see at once by the discussion above that it is surjective. We also see at once that its kernel equals $I_1 \cap \cdots \cap I_m$. We claim that this is in fact equal to $I_1 \cdots I_m$. We verify this in case $m = 2$; the general case follows similarly. First, the inclusion $IJ \subset I \cap J$ always holds. In the other direction, let $(x, y) \in I \times J$ be such that $x + y = 1$. This makes sense since I, J are pairwise co-prime. Let

$z \in I \cap J$. Then $xz + yz = z$ and we see that $z \in IJ$. This completes the proof sketch. In hindsight, the inverse of the isomorphism constructed is the map

$$\prod_{j=1}^{m} A/I_j \longrightarrow A/\prod_{j=1}^{m} I_j,$$

$$(\overline{y}_1, \dots, \overline{y}_m) \mapsto \overline{x}_1\overline{y}_1 + \cdots + \overline{x}_m\overline{y}_m,$$

where the x_i's are the elements crafted above.

Now we apply CRT to Lagrange Interpolation. Suppose

$$\alpha_1, \dots, \alpha_m \in K$$

are pairwise distinct ($m > 1$). Set

$$A = K[X],$$

and

$$I_1 = (X - \alpha_1) \cdot K[X], \dots, I_m = (X - \alpha_m) \cdot K[X].$$

Note that these ideals are pairwise co-prime. Indeed, $I_i + I_j$ contains the unit $\alpha_j - \alpha_i \in K \setminus \{0\}$ if $i \neq j$, since $(X - \alpha_i) - (X - \alpha_j) = \alpha_j - \alpha_i \neq 0$.

By the CRT,

$$K[X]/\prod_{i=1}^{m}(X - \alpha_i) \simeq K[X]/(X - \alpha_1) \times \cdots \times K[X]/(X - \alpha_m).$$

The residue-classes on the left-hand side are represented exactly by the polynomials $f(X) \in K[X]$ of degree at most $m-1$, as all is taken modulo the polynomial $\prod_{i=1}^{m}(X - \alpha_i)$ (which is of degree m).

The isomorphism ψ given by the CRT sends a polynomial f of degree at most $m - 1$ to

$$(f(X) \bmod (X - \alpha_1), \dots, f(X) \bmod (X - \alpha_m)).$$

But since α_j is a zero of the polynomial $f(X) - f(\alpha_j)$, it holds that $X - \alpha_j$ divides $f(X) - f(\alpha_j)$ and hence that

$$f(\alpha_j) \equiv f(X) \pmod{X - \alpha_j},$$

for $j = 1, \dots, m$. Therefore, ψ sends a polynomial f of degree at most $m - 1$ to

$$(f(\alpha_1), \dots, f(\alpha_m)).$$

This concludes the proof.

Note that the discussion on the inverse of the CRT isomorphism above also puts the polynomials $\delta_i(X)$ from the proof of Theorem 10.3, or more precisely, the inverse evaluation map $\mathcal{E}^{-1}$, into the perspective of the CRT; namely, when crafting the x_i's from the discussion above in the specific setting of Lagrange interpolation, one obtains the $\delta_i(X)$'s.

11.6.2 Proof by Vandermonde Determinant

Assume $m > 1$. Theorem 11.6 can also be proved directly with linear algebra. Consider the square matrix M defined over K, with the vector $(1, \alpha_i, \alpha_i^2, \ldots, \alpha_i^{m-1})$ as its i-th row, $1 \leq i \leq m$, a *Vandermonde matrix*. It is well-known from algebra that

$$\det(M) = \prod_{1 \leq i < j \leq m} (\alpha_j - \alpha_i).$$

This identity is easily verified by "expanding the determinant of the matrix along the left-most column" and by using induction.

Since the α_i's are distinct by assumption, we have

$$\det(M) \neq 0.$$

A moment's reflection reveals that

$$\mathcal{E}(f) = \mathbf{y} \in K^m$$

if and only if

$$M\mathbf{a} = \mathbf{y},$$

where

$$\mathbf{a} = (a_0, \ldots, a_{m-1})^T \in K^m$$

such that

$$f(X) = a_0 + a_1 X + \ldots + a_{m-1} X^{m-1} \in K[X].$$

Indeed, for $i = 1, \ldots, m$, the i-th equation in the linear system is of the form

$$a_0 + a_1 \alpha_i + \ldots + a_{m-1} \alpha_i^{m-1} = y_i.$$

Since $\det(M) \neq 0$, the result follows.

11.7 Using the Point at Infinity

There is one "extra point of evaluation" that can be added to Theorem 11.6. This is explained below.

DEFINITION 11.10 *Let $m \geq 1$ be an integer. For $f(X) \in K[X]$, define*

$$f(\infty_m) = a_{m-1},$$

where $a_{m-1} \in K$ is the coefficient of X^{m-1} in $f(X)$. Here, ∞_m is to be treated as a formal symbol.

THEOREM 11.11 *Let $m \geq 1$ be an integer. Suppose $\alpha_1, \ldots, \alpha_m \in K \cup \{\infty_m\}$ are pairwise distinct. Then the map*

$$\mathcal{E}' : K[X]_{\leq m-1} \longrightarrow K^m,$$

$$f \mapsto (f(\alpha_1), \ldots, f(\alpha_m))$$

is an isomorphism of K-vector spaces.

Proof If $\alpha_1, \ldots, \alpha_m \in K$, then the claim follows by Theorem 11.6. So, assume, without loss of generality, $\alpha_m = \infty_m$.

It is clear that $\mathcal{E}'$ is a morphism of K-vector spaces. To show that it is an isomorphism, we show it is surjective and injective. If $m = 1$, then $\mathcal{E}'$ is the identity map.

Suppose $m > 1$. As the dimension of the K-vector space on the left equals that of the one on the right (namely, each equals m), by linear algebra it suffices to prove that the morphism $\mathcal{E}'$ is injective. Suppose $\mathcal{E}'(f) = \mathbf{0}$. Since

$$f(\infty_m) = 0,$$

it holds that the degree of $f(X)$ is at most $m-2$. However, $f(X)$ has $m-1$ zeros, since

$$f(\alpha_1) = \cdots = f(\alpha_{m-1}) = 0.$$

Therefore, $f(X)$ is identical to the 0-polynomial.

An explicit description of the inverse of $\mathcal{E}'$ can be easily extracted from the proof of Theorem 11.6. If $m = 1$, this is trivial. Suppose $m > 1$. Consider the morphism

$$\mathcal{E} : K[X]_{\leq m-2} \longrightarrow K^{m-1},$$

$$g(X) \mapsto (g(\alpha_1), \ldots, g(\alpha_{m-1})),$$

and its inverse $\mathcal{E}^{-1}$. For each $(y_1, \ldots, y_m) \in K^m$, simply define

$$(g_0, \ldots, g_{m-2}) := \mathcal{E}^{-1}(y_1 - y_m\alpha_1^{m-1}, \ldots, y_{m-1} - y_m\alpha_{m-1}^{m-1}).$$

Then

$$f(X) = g_0 + \cdots + g_{m-2}X^{m-2} + y_mX^{m-1}$$

is the $\mathcal{E}'$-inverse of $(y_1, \ldots, y_m)$. $\square$

Using algebraic geometry, it can be justified in what way exactly $f(\infty_m)$ is in fact “an evaluation at the point at infinity.”

An immediate consequence of this theorem is that we get the following relaxation for the conditions of Theorem 11.9.

Theorem 11.12 *Suppose r, ℓ are integers such that $1 \leq r \leq \ell \leq q+1$. Then there exists an $\mathbb{F}_q$-linear (r, ℓ, q)-interpolation code.*

With Theorem 11.11 in hand, the proof of the theorem above is essentially the same as that of Theorem 11.9.

11.8 Secret-Recovery in Presence of Corruptions

Shamir's (t, n)-threshold secret sharing scheme is secure against a *passive* t-bounded adversary: t shares give no information about the secret, while $t + 1$ shares determine it uniquely. We now consider an *active* t-bounded adversary, i.e., one that gets to select a set B with $|B| = t > 0$, to inspect the shares $(s_i)_{i\in B}$ and to replace the original t shares $(s_i)_{i\in B}$ by corrupted shares $(\tilde{s}_i)_{i\in B} \neq (s_i)_{i\in B}$. Note that there is still t-privacy with respect to such an adversary.

In Chapter 5, we explained that for some values of n and t it is in principle possible to recover the original polynomial (and hence the secret) from the modified shares. Here, we shall see that one can even do so efficiently.

11.8.1 Definitions

Let $\mathcal{F}$ be a set with $|\mathcal{F}| \geq 2$ and let $\ell \geq 1$ be an integer. Fix some index set $\mathcal{I}$ for $\mathcal{F}^\ell$.

Definition 11.13 (Hamming Distance) *Let* $\mathbf{x}, \mathbf{y} \in \mathcal{F}^\ell$. *The* Hamming-distance *between* $\mathbf{x}$ *and* $\mathbf{y}$ *is defined as*

$$d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) = |\{j \in \mathcal{I} \ : \ x_j \neq y_j\}|.$$

If $\mathcal{F}$ *is a field, the* Hamming-weight *of* $\mathbf{x} \in \mathcal{F}^\ell$ *is defined as*

$$w_{\mathrm{H}}(\mathbf{x}) = |\{j \in \mathcal{I} \ : \ x_j \neq 0\}|.$$

Definition 11.14 (Minimum Distance) *Let* $C \subset \mathcal{F}^\ell$ *be a code with* $|C| > 1$. *Its* minimum distance *is defined as*

$$d_{\min}(C) = \min_{\mathbf{x},\mathbf{y}\in C:\mathbf{x}\neq\mathbf{y}} d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}).$$

Lemma 11.15 (Minimum Distance of a Linear Code) *Suppose that* $C \subset \mathbb{F}_q^\ell$ *is an* $\mathbb{F}_q$*-linear code of positive dimension. Then*

$$d_{\min}(C) = \min_{\mathbf{x}\in C:\mathbf{x}\neq\mathbf{0}} w_{\mathrm{H}}(\mathbf{x}).$$

Proof It is easy to see that

$$d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) = w_{\mathrm{H}}(\mathbf{x} - \mathbf{y}),$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^\ell$. Since $\mathbf{x} - \mathbf{y} \in C$ whenever $\mathbf{x}, \mathbf{y} \in C$, the claim follows. □

Lemma 11.16 (metric) *Hamming-distance gives a* metric *on* $\mathcal{F}^\ell$. *In particular, it satisfies the* triangle inequality, *i.e.,*

$$d_{\mathrm{H}}(\mathbf{x}, \mathbf{z}) \leq d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) + d_{\mathrm{H}}(\mathbf{y}, \mathbf{z}),$$

for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathcal{F}^\ell$.

PROOF It is immediate that, for all $\mathbf{x}, \mathbf{y} \in \mathcal{F}^\ell$, it holds that $d_\mathrm{H}(\mathbf{x}, \mathbf{y}) \in \mathbb{R}$, $d_\mathrm{H}(\mathbf{x}, \mathbf{y}) \geq 0$, $d_\mathrm{H}(\mathbf{x}, \mathbf{y}) = d_\mathrm{H}(\mathbf{y}, \mathbf{x})$ and that $d_\mathrm{H}(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$. Thus, it remains to verify that the triangle inequality holds. The least sufficient number of coordinate modifications to pass from $\mathbf{x}$ to $\mathbf{z}$ directly equals $d_\mathrm{H}(\mathbf{x}, \mathbf{z})$. First passing from $\mathbf{x}$ to $\mathbf{y}$ and then on to $\mathbf{z}$ requires in total $d_\mathrm{H}(\mathbf{x}, \mathbf{y}) + d_\mathrm{H}(\mathbf{y}, \mathbf{z})$ modifications. The claim follows. $\square$

DEFINITION 11.17 (HAMMING-SPHERE) *Let* $\mathbf{x} \in \mathcal{F}^\ell$ *and let* ρ *be a nonnegative integer. The* Hamming-sphere $B(\mathbf{x}; \rho)$ *of* radius ρ centered at $\mathbf{x}$ *is the set*

$$B(\mathbf{x}; \rho) = \{\mathbf{x}' \in \mathcal{F}^\ell \ : \ d_\mathrm{H}(\mathbf{x}, \mathbf{x}') \leq \rho\} \subset \mathcal{F}^\ell.$$

LEMMA 11.18 *Let* $C \subset \mathcal{F}^\ell$ *be a code with* $|C| > 1$. *Set*

$$\rho = \left\lfloor \frac{d_{\min}(C) - 1}{2} \right\rfloor.$$

Then, for any $\mathbf{x}, \mathbf{x}' \in C$ *with* $\mathbf{x} \neq \mathbf{x}'$, *it holds that*

$$B(\mathbf{x}; \rho) \cap B(\mathbf{x}'; \rho) = \emptyset.$$

PROOF Suppose there is $\mathbf{x}'' \in \mathcal{F}^\ell$ such that $\mathbf{x}'' \in B(\mathbf{x}; \rho) \cap B(\mathbf{x}'; \rho)$. Then

$$d_\mathrm{H}(\mathbf{x}, \mathbf{x}'') \leq \rho \ \text{ and } \ d_\mathrm{H}(\mathbf{x}'', \mathbf{x}') \leq \rho.$$

Hence, by the triangle-inequality,

$$d_\mathrm{H}(\mathbf{x}, \mathbf{x}') \leq d_\mathrm{H}(\mathbf{x}, \mathbf{x}'') + d_\mathrm{H}(\mathbf{x}'', \mathbf{x}') \leq 2 \cdot \rho \leq d_{\min}(C) - 1 < d_{\min}(C),$$

a contradiction. $\square$

An immediate consequence of this lemma is the following. Let an arbitrary vector $\mathbf{x} \in C$ be given and suppose that $\mathbf{x}$ is replaced by some $\tilde{\mathbf{x}} \in \mathcal{F}^\ell$ with $d_\mathrm{H}(\mathbf{x}, \tilde{\mathbf{x}}) \leq \rho$: at most ρ *corruptions* have been introduced. Then, despite these corruptions, the vector $\tilde{\mathbf{x}}$ determines $\mathbf{x}$ uniquely: the vector $\mathbf{x}$ is the unique codeword that is closest to $\tilde{\mathbf{x}}$. [5] Note that this argument may fail if $\rho' > \rho$ corruptions are introduced.[6]

LEMMA 11.19 *Suppose* $C \subset \mathcal{F}^\ell$ *is a* $(t{+}1, \ell, \bar{q})$*-interpolation code. Then* $d_{\min}(C) = \ell - t$.

PROOF If $\mathbf{x}, \mathbf{y} \in C$ and $\mathbf{x} \neq \mathbf{y}$, then $\mathbf{x}, \mathbf{y} \in C$ can agree in at most t coordinates; otherwise $\mathbf{x} = \mathbf{y}$ by the $(t + 1)$-interpolation property. Hence, $d_{\min}(C) \geq \ell - t$. Fix an arbitrary $\mathbf{v} \in \mathcal{F}^t$ and an arbitrary set $B \subset \mathcal{I}$ such that $|B| = t$. Then, by the $(t + 1)$-interpolation property, there are exactly $|\mathcal{F}| \geq 2$ vectors $\mathbf{x} \in C$ such

[5] Of course, this observation is at the heart of the theory or error correcting codes, see e.g. [138].

[6] Yet, if the number of corruptions is at most $d_{\min}(C) - 1$, then at least it can be *detected* that there are corruptions, since such number of corruptions will not transform any codeword to some other codeword.

that $\mathbf{x}_B = \mathbf{v}$. Any two distinct such words $\mathbf{x}, \mathbf{y} \in C$ satisfy $d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) \leq \ell - t$. The claim follows. □

Now consider Shamir's (t, n)-threshold secret sharing scheme. Write C for the underlying $(t+1, n+1, q)$-interpolation code, and C' for "the part of C that corresponds to the shares," i.e., C but without its 0-th coordinate. Note that C' is a $(t+1, n, q)$-interpolation code and that there is a one-to-one correspondence between C and C'. We have $d_{\min}(C') = n - t$. By the considerations above, $\lfloor \frac{n-t-1}{2} \rfloor$ corruptions can be tolerated. Therefore, in order to tolerate a t-bounded adversary, it is a *sufficient* condition that

$$t \leq \left\lfloor \frac{n-t-1}{2} \right\rfloor,$$

which is equivalent to

$$t < \frac{n}{3}.$$

It is easy to show that $t < \frac{n}{3}$ is also a *necessary* condition. Towards a contradiction, assume t corruptions can be tolerated if $t \geq \frac{n}{3}$. Select two words $\mathbf{x}, \mathbf{y}$ in C such that $x_0 \neq y_0$ and such that they are at distance exactly $n - t + 1$ from one another. Let $\mathbf{x}', \mathbf{y}'$ be the corresponding words in C'. These are at the smallest possible distance $n - t$ from one another. Select an integer ρ' such that

$$\left\lfloor \frac{n-t-1}{2} \right\rfloor < \rho' \leq t.$$

Such ρ' exists since $t \geq \frac{n}{3}$ by assumption. Since

$$2\rho' \geq n - t,$$

it follows that there exists

$$\mathbf{z}' \in B(\mathbf{x}', \rho') \cap B(\mathbf{y}', \rho').$$

It is clear that there is ambiguity in such $\mathbf{z}'$: it could have arisen from "$\mathbf{x}'$ plus $\rho' \leq t$ corruptions" (in which case the secret would have been x_0) or it could have arisen from "$\mathbf{y}'$ plus $\rho' \leq t$ corruptions" (in which case the secret would have been $y_0 \neq x_0$).

11.8.2 The Welch/Berlekamp Algorithm

In case of Shamir's (t, n)-secret sharing scheme, the secret can be reconstructed *efficiently* in the presence of an active t-bounded adversary if $t < \frac{n}{3}$. Although the problem as such is non-linear, it can be linearized by means of a clever trick. The method we present is due to Welch and Berlekamp from the 1970s. In the late 1990s, Sudan [171] and Guruswami and Sudan [106] showed that a generalization of the mathematical idea behind this algorithm leads to a significantly better algorithm for the so-called *list decoding problem* introduced by Elias in the 1950s. [7] Our presentation follows Sudan.

Consider Shamir's secret sharing scheme as presented in Section 11.5.2. Write $K = \mathbb{F}_q$. Assume $t < \frac{n}{3}$. Consider a vector of shares for a secret $s \in K$

$$\mathbf{s}_f^* = (s_1, \ldots, s_n) = (f(\alpha_1), \ldots, f(\alpha_n)) \in K^n,$$

where $f(X) \in K[X]_{\leq t}$.

Let $\mathbf{e} \in K^n$ be an "error vector" subject to

$$w_H(\mathbf{e}) \leq t.$$

Define

$$\tilde{\mathbf{s}}^* = \mathbf{s}_f^* + \mathbf{e} = (\tilde{s}_1, \ldots, \tilde{s}_n),$$

the *corrupted* share vector.

The method "interpolates" the n points $(\alpha_i, \tilde{s}_i)$ by a bi-variate polynomial $Q(X,Y) \in K[X,Y]$ of a special form. From a computational view this comes down to solving a system of linear equations. Finally, the polynomial $f(X)$ is extracted from $Q(X,Y)$ in a very simple way.

Polynomials $h(X,Y) \in K[X,Y]$ are of the form

$$h(X,Y) = \sum_{i,j \geq 0} a_{ij} X^i Y^j,$$

where the $a_{ij} \in K$ are such that all but finitely many of them are zero. Now let $Q(X,Y) \in K[X,Y]$ be any polynomial such that

1. $Q(\alpha_i, \tilde{s}_i) = 0$, for $i = 1, \ldots, n$.
2. $Q(X,Y) = f_0(X) - f_1(X) \cdot Y$, where
   - $f_0(X), f_1(X) \in K[X]$,
   - $f_1(0) = 1$,
   - $\deg f_0(X) \leq 2t$, and
   - $\deg f_1(X) \leq t$.

We will show the following.

- There exists at least one solution $Q(X,Y)$ satisfying the constraints above.
- *Some* solution can be found by solving a system of linear equations.
- For *all* possible solutions $Q(X,Y)$ defined by some $f_0(X), f_1(X)$, it holds that

$$f(X) = \frac{f_0(X)}{f_1(X)}.$$

[7] For subsequent results in this area, please refer e.g. to [107] and to the references therein.

First note that the conditions on $Q(X,Y)$ can be stated equivalently in terms of a system of linear equations, by taking its coefficients as the variables. To show that there exists at least one solution $Q(X,Y)$, let $B \subset \{1,\ldots,n\}$ be the set of indices i with $e_i \neq 0$.

If $B = \emptyset$, then

$$Q(X,Y) = f(X) - 1 \cdot Y$$

is a solution. Now suppose $B \neq \emptyset$. Define

$$k(X) = \prod_{i \in B} \frac{(X - \alpha_i)}{-\alpha_i}.$$

Note that

1. $k(X) \in K[X]$.
2. $k(0) = 1$.
3. $\deg k(X) \leq t$.
4. $k(\alpha_i) = 0$ if $i \in B$.

Now

$$Q(X,Y) = k(X) \cdot f(X) - k(X) \cdot Y$$

is a solution. The only condition left to be verified is that it interpolates as required. If $i \in B$, then

$$Q(\alpha_i, \tilde{s}_i) = k(\alpha_i) \cdot f(\alpha_i) - k(\alpha_i) \cdot \tilde{s}_i = 0 \cdot s_i - 0 \cdot \tilde{s}_i = 0.$$

On the other hand, if $i \notin B$, then

$$Q(\alpha_i, \tilde{s}_i) = k(\alpha_i) \cdot f(\alpha_i) - k(\alpha_i) \cdot \tilde{s}_i = k(\alpha_i) \cdot s_i - k(\alpha_i) \cdot s_i = 0,$$

where we have used that $(\alpha_i, s_i) = (\alpha_i, \tilde{s}_i)$ if $i \notin B$.

It is left to show that for each solution $Q(X,Y)$ given by some $f_0(X), f_1(X)$ it holds that $f(X) = f_0(X)/f_1(X)$. By the fact that $f_1(0) = 1$, it follows that $f_1(X) \not\equiv 0$. Therefore, the fraction is well-defined. It is left to show it equals $f(X)$, as desired.

To this end, define

$$Q'(X) = Q(X, f(X)) \in K[X],$$

and note that

$$\deg Q'(X) \leq 2t.$$

For $i \notin B$, it holds that

$$Q'(\alpha_i) = Q(\alpha_i, f(\alpha_i)) = Q(\alpha_i, s_i) = Q(\alpha_i, \tilde{s}_i) = 0.$$

Since $t < n/3$ and $|B| \leq t$, it holds that

$$n - |B| \geq n - t > 2t.$$

We conclude that the number of zeroes of $Q'(X)$ exceeds its degree. Therefore, $Q'(X)$ must be the zero-polynomial, and hence the identity

$$f_0(X) - f_1(X) \cdot f(X) = 0$$

holds in $K[X]$. This establishes the claim.

11.9 Formal Definition of Secret Sharing

We give a formal definition of secret sharing. Before doing so, we state some terminology concerning random variables and information theory. For an introduction to information theory please refer to [62]. [8]

11.9.1 Some Notions from Information Theory

Let $[0, 1]$ denote the interval of real numbers λ with $0 \leq \lambda \leq 1$.

Definition 11.20 (Probability Space) *A* finite probability space *consists of a finite nonempty set* Ω*, the* sample space*, and a* probability function

$$P : \Omega \longrightarrow [0, 1]$$

with

$$\sum_{\omega \in \Omega} P(\omega) = 1.$$

Definition 11.21 (Events) *A subset* $\mathcal{E} \subset \Omega$ *is an* event*. Its* probability *is*

$$P(\mathcal{E}) := \sum_{\omega \in \mathcal{E}} P(\omega).$$

By default, $P(\emptyset) = 0$*.*

Definition 11.22 (Random Variable) *A* random variable X *defined on a finite probability space* (Ω, P) *consists of a finite nonempty set* $\mathcal{X}$*, the* alphabet*, and a map* $X : \Omega \longrightarrow \mathcal{X}$*. By definition,* $X = x$ *is the event*

$$\{\omega \in \Omega : X(\omega) = x\}.$$

The probability distribution *of* X *is defined as the function*

$$P_X : \mathcal{X} \longrightarrow [0, 1]$$

$$x \mapsto P(X = x).$$

[8] For a quick introduction to information theory from the point of view of cryptography, please refer to [71]. For elementary calculus of (conditional) probabilities involving random variables, please refer to [164].

Remark 11.3 (Alphabet Notation) In general, the alphabet of a random variable will be denoted by a calligraphic version of the name of the random variable. For instance, the alphabet of X is $\mathcal{X}$.

Let X, Y be random variables defined on a finite probability space (Ω, P).

Definition 11.23 (Uniform Distribution) *The random variable X has the* uniform distribution *if*

$$P_X(x) = \frac{1}{|\mathcal{X}|}$$

for all $x \in \mathcal{X}$.

Definition 11.24 (Products of Random Variables) *The* product

$$XY : \Omega \longrightarrow \mathcal{X} \times \mathcal{Y}$$

is the random variable defined by

$$XY(\omega) = (X(\omega), Y(\omega)).$$

Definition 11.25 (Independence of Random Variables) *The random variables X, Y are* independent *if*

$$P_{XY}(x, y) = P_X(x) \cdot P_Y(y)$$

for all $x \in \mathcal{X}, y \in \mathcal{Y}$.

Lemma 11.26 *The random variable XY has the uniform distribution if and only if X and Y are independent and both X and Y have the uniform distribution.*

Proof We show the forward direction. The other direction is immediate. For all $(x, y) \in \mathcal{X} \times \mathcal{Y}$, the following holds. First,

$$P_{XY}(x, y) = \frac{1}{|\mathcal{X}| \cdot |\mathcal{Y}|}$$

since there is the uniform distribution on $\mathcal{X} \times \mathcal{Y}$. Second,

$$P_X(x) = \sum_{y \in \mathcal{Y}} P_{XY}(x, y) = |\mathcal{Y}| \cdot \frac{1}{|\mathcal{X}| \cdot |\mathcal{Y}|} = \frac{1}{|\mathcal{X}|},$$

and

$$P_Y(y) = \sum_{x \in \mathcal{X}} P_{XY}(x, y) = |\mathcal{X}| \cdot \frac{1}{|\mathcal{X}| \cdot |\mathcal{Y}|} = \frac{1}{|\mathcal{Y}|}.$$

Therefore,

$$P_{XY}(x, y) = P_X(x) \cdot P_Y(y).$$

□

Definition 11.27 (Conditioned Random Variable) *If $\mathcal{E}$ is an event with $P(\mathcal{E}) > 0$, then $P_{X|\mathcal{E}}$ denotes the probability distribution of X conditioned on $\mathcal{E}$. Precisely,*

$$P_{X|\mathcal{E}}(x) = \frac{P(X = x \ \cap \ \mathcal{E})}{P(\mathcal{E})}$$

for all $x \in \mathcal{X}$.

It is sometimes convenient to speak of the random variable X conditioned on an event $\mathcal{E}$ with $P(\mathcal{E}) > 0$ as a random variable in its own right. This requires some care, as the probability space must be adapted in order to do so. Precisely, the probability space is adapted to the pair $(\mathcal{E}, P')$ where P' is the probability function

$$P' : \mathcal{E} \longrightarrow [0,1]$$

$$\omega \mapsto \frac{P(\omega)}{P(\mathcal{E})}.$$

We now define $X^{\mathcal{E}}$ as the random variable on the finite probability space $(\mathcal{E}, P')$ such that, as a function, $X^{\mathcal{E}}$ is simply the restriction of X to the subdomain $\mathcal{E}$. Note that $P'_{X^{\mathcal{E}}}$ is nothing else than the conditional probability distribution $P_{X|\mathcal{E}}$.

Definition 11.28 (Support) *The* support *of the random variable X, denoted* supp(X), *is the set consisting of all $x \in \mathcal{X}$ such that $P_X(x) > 0$.*

Note that it is not assumed that $\mathcal{X} = \text{supp}(X)$.

Definition 11.29 (Vector of Random Variables) *A* vector of random variables *is a vector $\mathbf{X} = (X_j)_{j \in \mathcal{I}}$ such that*

- *The* index set *$\mathcal{I} \subset \mathbb{Z}$ is finite and non-empty.*
- *Each of the X_j's is a random variable and they are all defined on the same finite probability space.*

Note that a vector of random variables is in fact a nonempty finite product of random variables, indexed by a set of integers.

Definition 11.30 *Let $\mathbf{X} = (X_j)_{j \in \mathcal{I}}$ be a vector of random variables. If $B \subset \mathcal{I}$ with $B \neq \emptyset$, then $\mathbf{X}_B$ denotes the vector of random variables $(X_j)_{j \in B}$. Moreover, $\mathcal{X}_B$ denotes the Cartesian product $\prod_{j \in B} \mathcal{X}_j$. If $B = \mathcal{I}$, then $\mathcal{X}$ is a shorthand for $\mathcal{X}_B$.*

Definition 11.31 (Shannon-Entropy) *Let $P : U \to [0,1]$ be a probability function defined on a nonempty finite set U. Its* Shannon entropy *is defined as* [9]

$$H(P) = -\sum_{\mathbf{u} \in U} P(\mathbf{u}) \cdot \log_2 P(\mathbf{u}).$$

[9] We adhere to the usual convention that $0 \cdot \log_2 0 = 0$, which is justified by a simple continuity argument.

For the random variable X we define

$$H(X) = H(P_X).$$

Recall that Shannon-entropy can be interpreted essentially as the optimal encoding length $\lambda(P)$ of outcomes of a random process distributed according to P, when prefix-free encoding is used and when optimality means minimizing the expected bit-length of the encoding, where the expectation is taken over P.[10]

Definition 11.32 (Encoding Length) *The* average encoding length *of a vector $\mathbf{X} = (X_j)_{j\in\mathcal{I}}$ of random variables is defined as $\lambda(\mathbf{X}) = \frac{1}{|\mathcal{I}|} \cdot \sum_{j\in\mathcal{I}} H(X_j)$.*

Definition 11.33 (Conditional Entropy) *The* entropy of X conditioned on Y *is defined as*

$$H(X|Y) = \sum_{y\in\mathcal{Y}: P_Y(y)>0} P_Y(y) \cdot H(X|Y=y),$$

where $H(X|Y=y) = H(P_{X|Y=y})$.

In other words, $H(X|Y)$ is the expected entropy of $H(X|Y=y)$ with $P_Y(y) > 0$, where the expectation is taken over P_Y.

We will just need the basic facts from information theory given below.

Lemma 11.34 *Let (X, Y, Z) be a vector of random variables. Then the following holds.*

1. $0 \leq H(X) \leq \log_2 |\mathcal{X}|$. *Equality on the left holds if and only if there exists $x \in \mathcal{X}$ with $P_X(x) = 1$. Equality on the right holds if and only if X has the uniform distribution.*
2. *(Monotonicity 1) $H(X|Z) \leq H(XY|Z)$*
3. *(Monotonicity 2) $0 \leq H(X|Y) \leq H(X)$. Equality on the left holds if and only if Y determines X with probability 1, i.e., for each $y \in \mathcal{Y}$ with $P_Y(y) > 0$, there is a unique $x \in \mathcal{X}$ such that $P_{X|Y=y}(x) = 1$. Moreover, $H(X|Y) = 0$ implies $H(Y) \geq H(X)$. Equality on the right holds if and only if X and Y are independent.*
4. *(Chain Rule) $H(XY|Z) = H(X|Z) + H(Y|XZ)$.*

The proofs of these claims can be found in [62].

[10] It holds that $H(P) \leq \lambda(P) \leq H(P) + 1$. In an appropriate asymptotic version of this problem, $\lambda(P) = H(P)$.

11.9.2 Defining Secret Sharing

Let $\mathbf{X}$ be a vector of random variables, indexed by a set $\mathcal{I}$.

Definition 11.35 (Reconstructing Set) *Let $B \subset \mathcal{I}$ with $B \neq \emptyset$ and let $j \in \mathcal{I}$. Then B is a* reconstructing set for $\{j\}$ *if $H(X_j|\mathbf{X}_B) = 0$, i.e., $\mathbf{X}_B$ determines X_j.*

Note that if $j \in B$, then B is a reconstructing set for $\{j\}$.

Definition 11.36 (Privacy Set) *Let $B \subset \mathcal{I}$ with B and let $j \in \mathcal{I}$. If $B \neq \emptyset$, then B is a* privacy set for $\{j\}$ *if $B = \emptyset$ if $H(X_j|\mathbf{X}_B) = H(X_j)$, i.e., $\mathbf{X}_B$ is independent from X_j. If $B = \emptyset$, then B is a privacy set by default.*

Remark 11.4 It follows immediately from the definitions that a nonempty set $B \subset \mathcal{I}$ is both a reconstructing- *and* a privacy set for $\{j\}$ if and only if $H(X_j) = 0$. In particular, if $H(X_j) > 0$ and B is not a reconstructing set for $\{j\}$, then there is a positive amount of uncertainty about X_j when given $\mathbf{X}_B$.

We now define secret sharing.

Definition 11.37 (Secret Sharing) *A* secret sharing scheme *is a vector $\mathbf{S}$ of random variables, indexed by a set $\mathcal{I}$ with $0 \in \mathcal{I}$ and $|\mathcal{I}| > 1$, such that the following holds.*

- Uniformity of S_0:
$$H(S_0) = \log_2 |\mathcal{S}_0| \geq 1.$$
In other words, the random variable S_0 has the uniform distribution on $\mathcal{S}_0$, where $|\mathcal{S}_0| > 1$.
- Reconstruction: *The set $\mathcal{I}^* := \mathcal{I} \setminus \{0\}$ is a reconstructing set for $\{0\}$.*
- Privacy: *By default, the empty set is a privacy set for $\{0\}$.* [11]

Definition 11.38 *The set $\mathcal{I}^*$ is the* player set. *Define $n(\mathbf{S}) = |\mathcal{I}^*|$, the number of players. The variable S_0 is the* secret. *For each $j \in \mathcal{I}^*$, the variable S_j is the* share *for player j. The set $\mathcal{S}_j$ is the corresponding* share-space.

The definition can be stated in terms of (generalized) codes, as follows. Let $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_n$ be nonempty finite sets. Define $\mathcal{S} := \mathcal{S}_0 \times \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$. Let $C \subset \mathcal{S}$ be non-empty and let P be a probability distribution on C. Now sample $\mathbf{x} = (x_0, x_1, \ldots, x_n) \in C$ according to P. The uniformity condition from the definition says that x_0 has the uniform distribution on $\mathcal{S}_0$ and that $|\mathcal{S}_0| > 1$. Let $B \subset \mathcal{I}^*$ with $B \neq \emptyset$. Then B is a reconstructing set if x_0 can be guessed with probability 1 from $\mathbf{x}_B$ alone. The definition requires that $\mathcal{I}^*$ is a reconstructing set. The set B is a privacy set if $\mathbf{x}_B$ is independently distributed from x_0. By default, the

[11] There are no *a priori* requirements concerning privacy sets. But, of course, secret sharing starts making sense when there are *nonempty* privacy sets.

empty set is a privacy set. There are no further a priori requirements. Note that if we cast secret sharing from interpolation codes in this framework, in particular Shamir's scheme, then the distribution imposed on the code is the uniform one.

An equivalent definition of privacy sets is as follows. Let $B \subset \mathcal{I}^*$ with $B \neq \emptyset$. Then B is a privacy set if $P_{\mathbf{S}_B | S_0 = s} = P_{\mathbf{S}_B}$ for each $s \in \mathcal{S}_0$. This means that if $\mathbf{s}$ is sampled from $\mathcal{S}$ conditioned on $S_0 = s$ for some secret $s \in \mathcal{S}_0$,[12] the shares $\mathbf{s}_B$ for a non-empty privacy set B have a distribution that does not depend on the particular secret s. For this reason, it should be clear that the explanation of privacy and reconstruction in secret sharing based on interpolation codes from Section 11.4 generalizes immediately to the general definition of secret sharing given here.

Note that the definition allows the secret and individual shares to be in different sets, not even necessarily of the same cardinality (as opposed to secret sharing based on interpolation codes from Section 11.4, where all are in the same set $\mathcal{F}$).

Definition 11.39 *The* length of the secret *is* $\lambda_0(\mathbf{S}) := \log_2 |\mathcal{S}_0|$. *The* average length of the shares *is* $\lambda^*(\mathbf{S}) := \lambda(\mathbf{S}^*)$, *where* $\mathbf{S}^* = (S_j)_{j \in \mathcal{I}^*}$.

Remark 11.5 Some authors do not impose the uniform distribution on the secret but require that $H(S_0) > 0$ and that the support of S_0 is all of $\mathcal{S}_0$, all other requirements being the same. The resulting definition is essentially the same as the one given here. Briefly, one passes back to it by conditioning on each secret separately, and "gluing" these conditional distributions together by imposing the uniform probability distribution on the secret. See [28] for details.

Definition 11.40 (Access Structure) *The* access structure $\Gamma(\mathbf{S})$ *consists of all reconstructing sets* $B \subset \mathcal{I}^*$.

By definition of a secret sharing scheme, it holds that $\Gamma(\mathbf{S}) \neq \emptyset$ since $\mathcal{I}^* \in \Gamma(\mathbf{S})$.

Definition 11.41 (r-Reconstruction) *Let r be a positive integer. The secret sharing scheme $\mathbf{S}$ has* r-reconstruction *if for each subset $B \subset \mathcal{I}^*$ with $|B| \geq r$ it holds that $B \in \Gamma(\mathbf{S})$. The smallest positive integer r such that $\mathbf{S}$ has r-reconstruction is denoted $r(\mathbf{S})$.*

Note that $1 \leq r(\mathbf{S}) \leq n(\mathbf{S})$. Furthermore, for all integers r with $r(\mathbf{S}) \leq r \leq n(\mathbf{S})$ there is r-reconstruction.

Definition 11.42 (Adversary Structure) *The* adversary structure $\mathcal{A}(\mathbf{S})$ *consists of all privacy sets* $B \subset \mathcal{I}^*$.

By definition of a secret sharing scheme, it holds that $\mathcal{A}(\mathbf{S}) \neq \emptyset$ since $\emptyset \in \mathcal{A}(\mathbf{S})$.

[12] How this is done *algorithmically* depends of course on the details of the secret sharing scheme.

Definition 11.43 (t-privacy) *Let t be a nonnegative integer. The secret sharing scheme* $\mathbf{S}$ *has t-privacy if for each subset $B \subset \mathcal{I}^*$ with $|B| \leq t$ it holds that $B \in \mathcal{A}(\mathbf{S})$. The largest nonnegative integer t such that* $\mathbf{S}$ *has t-privacy is denoted $t(\mathbf{S})$.*

As before, 0-privacy means by convention that "the empty set of shares does not decrease uncertainty about the secret," which makes sense. Note that $t(\mathbf{S}) = 0$ does not mean there is no nonempty privacy set. It just means that $\{j\} \in \Gamma(\mathbf{S})$ for some $j \in \mathcal{I}^*$. The definition of secret sharing allows the "cryptographically non-interesting" case that there is no non-empty privacy set at all. However, it is useful for technical reasons in proofs to allow this case. Similarly, it is useful to allow $n(\mathbf{S}) = 1$. Note that for all integers t with $0 \leq t \leq t(\mathbf{S})$ there is t-privacy.

By the definitions and by Remark 11.4, we have the following lemma.

Lemma 11.44 $\mathcal{A}(\mathbf{S}) \cap \Gamma(\mathbf{S}) = \emptyset$. *In particular,* $0 \leq t(\mathbf{S}) < r(\mathbf{S}) \leq n(\mathbf{S})$.

Definition 11.45 $\mathcal{A}'(\mathbf{S}) = 2^{\mathcal{I}^*} \setminus \Gamma(\mathbf{S})$. [13]

Note that $\mathcal{A}(\mathbf{S}) \subset \mathcal{A}'(\mathbf{S})$ by Lemma 11.44. Furthermore, note that if $\mathcal{A}(\mathbf{S}) \subsetneq \mathcal{A}'(\mathbf{S})$, then there is a set which is neither a privacy set nor a reconstructing set.

The following straightforward lemma gives a sufficient condition for proving privacy. It is quite useful when analyzing concrete schemes.

Lemma 11.46 (Sufficient Condition for Privacy) *Let $B \subset \mathcal{I}^*$ with $B \neq \emptyset$. If the distribution of $S_0 \times \mathbf{S}_B$ is the uniform distribution on $\mathcal{S}_0 \times \mathrm{supp}(\mathbf{S}_B)$, then B is a privacy set.*

The proof follows by direct application of Lemma 11.26

11.9.3 Combinatorial Structures

There are several interesting combinatorial structures associated with a secret sharing scheme. These are also relevant to applications of secret sharing.

Definition 11.47 (Monotone Structure) *A* monotone structure *is a pair (Γ, Ω) where Ω is a non-empty finite set, the* domain, *and where Γ is a collection of subsets of Ω such that:*

- $\emptyset \notin \Gamma$.
- $\Omega \in \Gamma$.
- Γ *is closed under taking supersets. This means that if $B \in \Gamma$ and $B \subset B' \subset \Omega$, then $B' \in \Gamma$.*

[13] The notation $2^{\mathcal{I}^*}$ refers to the collection of all subsets of the set $\mathcal{I}^*$.

DEFINITION 11.48 (MINIMAL SET) *Let* (Γ, Ω) *be a monotone structure and let* $B \in \Gamma$*. Then* B *is* minimal *if it is a minimal element in the partial order on* Γ *induced by set-inclusion. In other words,* B *is minimal if, for each proper subset* $B'' \subsetneq B$*, it holds that* $B'' \notin \Gamma$*.*

DEFINITION 11.49 (CONNECTED MONOTONE STRUCTURE) *A monotone structure* (Γ, Ω) *is* connected *if, for each* $i \in \Omega$*, there is a minimal set* $B \in \Gamma$ *with* $i \in B$*.*

DEFINITION 11.50 (ANTI-MONOTONE STRUCTURE) *An* anti-monotone structure *is a pair* $(\mathcal{A}, \Omega)$ *where* Ω *is a non-empty finite set, the* domain*, and where* $\mathcal{A}$ *is a collection of subsets of* Ω *such that*

- $\emptyset \in \mathcal{A}$.
- $\Omega \notin \mathcal{A}$.
- $\mathcal{A}$ *is closed under taking subsets. This means that if* $B \in \mathcal{A}$ *and* $B'' \subset B \subset \Omega$*, then* $B'' \in \mathcal{A}$*.*

Note that if **S** is a secret sharing scheme, then $\Gamma(\mathbf{S})$ is a monotone structure. Also note that both $\mathcal{A}(\mathbf{S})$ and $\mathcal{A}'(\mathbf{S})$ are anti-monotone structures.

DEFINITION 11.51 (DUAL) *If* (Γ, Ω) *is a monotone structure, then its* dual *is the monotone structure* (Γ^*, Ω)*, where*

$$\Gamma^* = \{B \subset \Omega \ : \ \Omega \setminus B \ \notin \Gamma\}.$$

The following lemma is straightforward to prove.

LEMMA 11.52 $(\Gamma^*)^* = \Gamma$.

11.9.4 Perfect and Ideal Schemes

Throughout this section, let **S** be a secret sharing scheme on the player set $\mathcal{I}^*$.

DEFINITION 11.53 (PERFECT SCHEMES) **S** *is* perfect *if* $\mathcal{A}(\mathbf{S}) = \mathcal{A}'(\mathbf{S})$*, i.e.,* $\Gamma(\mathbf{S}) \cup \mathcal{A}(\mathbf{S})$ *is a partition of* $2^{\mathcal{I}^*}$*.*

In other words, for each $B \subset \mathcal{I}^*$ with $B \neq \emptyset$, the corresponding shares either jointly give full information about the secret, or do not reduce uncertainty at all. Note that disjointness follows by Lemma 11.44.

DEFINITION 11.54 *Suppose* **S** *is perfect. Then* **S** *is* connected *if* $\Gamma(\mathbf{S})$ *is connected.*

In other words, this definition excludes "dummy players" in a perfect secret sharing scheme.

DEFINITION 11.55 (IDEAL SCHEMES) *Suppose* $\mathbf{S}$ *is perfect and connected. Then* $\mathbf{S}$ *is* ideal *if* $|\mathcal{S}_j| = |\mathcal{S}_0|$ *for all* $j \in \mathcal{I}^*$. *Without loss of generality we may assume* $\mathcal{S}_i = \mathcal{S}_j$ *for all* $i, j \in \mathcal{I}$ *and define* $\mathcal{F} := \mathcal{S}_0$. *We say that* $\mathbf{S}$ *is defined over the alphabet* $\mathcal{F}$.

Note that this definition is motivated by the result in Theorem 11.85, which we show later.

We are now in the position to adequately summarize the existence results on threshold secret sharing that we showed.

DEFINITION 11.56 (THRESHOLD SCHEMES) *The scheme* $\mathbf{S}$ *is a* (t, n)-threshold *secret sharing scheme if* $n = n(\mathbf{S})$, $t = t(\mathbf{S})$, *and* $r(\mathbf{S}) = t(\mathbf{S}) + 1$.

Note that if $\mathbf{S}$ is a threshold secret sharing scheme, then it is perfect and connected. It is perfect, since each set $B \subset \mathcal{I}^*$ satisfies either $|B| \leq t(\mathbf{S})$ (in which case it is a privacy set) or $|B| \geq r(\mathbf{S}) = t(\mathbf{S}) + 1$ (in which case it is a reconstructing set). Moreover, $\Gamma(\mathbf{S})$ is connected, since for each $j \in \mathcal{I}^*$ there is a set $B \subset \mathcal{I}^*$ with $j \in B$ and $|B| = r(\mathbf{S})$, and such a set B is minimal.

DEFINITION 11.57 (IDEAL THRESHOLD SECRET SHARING SCHEME) *Let* $t, n, \bar{q}$ *be integers with* $\bar{q} \geq 2$ *and* $0 \leq t < n$. *The class of* ideal (t, n)*-threshold secret sharing schemes over an alphabet of size* $\bar{q}$ *is denoted* $\mathrm{ITSS}(t, n, \bar{q})$.

THEOREM 11.58 *The secret sharing scheme from Section 11.4 constructed from a* $(t + 1, n + 1, \bar{q})$*-interpolation code* $(0 \leq t < n)$ *over an alphabet of cardinality* $\bar{q} \geq 2$ *is an ideal* (t, n)*-threshold scheme on* n *players with secret-space as well as each share-space having cardinality* $\bar{q}$.

11.10 Linear Secret Sharing Schemes

Linear secret sharing schemes, first studied in generality by Brickell [32], can be defined in several ways. We find it convenient here to start by giving a definition and analysis based on linear codes inspired by Massey [134, 135], who first considered linear secret sharing from arbitrary linear codes. Afterwards, we cast Brickell's *vector space construction* in this light.

We start with a straightforward but useful lemma.

LEMMA 11.59 (REGULARITY OF FINITE MORPHISMS) *Let* $\phi : G \longrightarrow H$ *be a morphism of groups. Suppose* G *is finite. Suppose* $x \in G$ *is selected uniformly random. Then* $\phi(x)$ *is uniformly random on* $\mathrm{im}\ \phi \subset H$, *the image of* ϕ. *In particular, if* ϕ *is surjective then* $\phi(x)$ *is uniformly random on* H.

PROOF We show that ϕ is *regular*, i.e., each element of im ϕ has the same number of distinct preimages under ϕ. Let $h' \in \mathrm{im}\ \phi$. Define

$$\phi^{-1}(h') := \{g \in G : \phi(g) = h'\}.$$

Let

$$g' \in \phi^{-1}(h').$$

Observe that

$$\phi^{-1}(h') = g' \cdot \ker\, \phi.$$

Moreover,

$$|g' \cdot \ker\, \phi| = |\ker\, \phi|,$$

since multiplication by g' induces a permutation of G. It follows that the sets of the form $\phi^{-1}(h')$ with $h' \in \mathrm{im}\, \phi$ partition G into disjoint sets of equal cardinality. The claim follows. □

Throughout this section, if $\ell > 1$ is an integer, then, unless stated otherwise, it is assumed that $\mathbb{F}_q^\ell$ is indexed by the set $\mathcal{I} = \{0, 1, \ldots, \ell - 1\}$. Moreover $\mathcal{I}^* := \{1, \ldots, \ell - 1\}$. For a nonempty set $B \subset \mathcal{I}^*$, define $\overline{B} := \mathcal{I}^* \setminus A$. If $\mathbf{x} \in \mathbb{F}_q^\ell$ and $B = \emptyset$, then we write $\mathbf{x}_B = 0$ by default.

11.10.1 Massey's Scheme

Definition 11.60 (Linear Secret Sharing for $\mathbb{F}_q$ over $\mathbb{F}_q$) *Let $C \subset \mathbb{F}_q^{n+1}$ be an $\mathbb{F}_q$-linear code with $n \geq 1$. Then C is a linear secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$ if the following conditions are satisfied.*

1. $\pi_0(C) \neq \{\mathbf{0}\} \subset \mathbb{F}_q$.
2. *For all $\mathbf{x} \in C$ it holds that if $\mathbf{x}_{\mathcal{I}^*} = \mathbf{0} \in \mathbb{F}_q^n$ then $x_0 = 0 \in \mathbb{F}_q$.*

This defines a secret sharing scheme on n players in the sense of Definition 11.37 as follows. The projection $\pi_0 : C \longrightarrow \mathbb{F}_q$ is a morphism of $\mathbb{F}_q$-vector spaces. By the first condition, it is not identically zero. Since the image is 1-dimensional, this projection is surjective. By Lemma 11.59 it holds that if $\mathbf{x} \in C$ is selected uniformly at random, then x_0 has the uniform distribution on $\mathbb{F}_q$, the secret-space.

In addition, the second condition means that from the n shares $x_1, \ldots, x_n$ jointly, each of which sits in a share-space $\mathbb{F}_q$, there is no ambivalence about the secret x_0: they jointly determine it with probability 1. Indeed, there is such ambivalence if and only if there are $\mathbf{x}', \mathbf{x}'' \in C$ are such that $\mathbf{x}'_{\mathcal{I}^*} = \mathbf{x}''_{\mathcal{I}^*}$ but $x'_0 \neq x''_0$. Since the code is $\mathbb{F}_q$-linear, the latter is equivalent to the existence of $\mathbf{x} \in C$ such that $\mathbf{x}_{\mathcal{I}^*} = \mathbf{0}$ but $x_0 \neq 0$.

Definition 11.61 (Dual of a Linear Code) *Let $C \subset \mathbb{F}_q^\ell$ be an $\mathbb{F}_q$-linear code with $\ell \geq 1$. Then its* dual *$C^\perp \subset \mathbb{F}_q^\ell$ is the $\mathbb{F}_q$-linear code consisting of all $\mathbf{y} \in \mathbb{F}_q^\ell$ such that $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ for all $\mathbf{x} \in C$.*

Lemma 11.62 *Let $C \subset \mathbb{F}_q^\ell$ be an $\mathbb{F}_q$-linear code with $\ell \geq 1$. Then the following holds.*

1. $\dim_{\mathbb{F}_q} C^\perp = \ell - \dim_{\mathbb{F}_q} C$.

2. $C = (C^\perp)^\perp$.

PROOF This is implied by standard facts from algebra. Let V be a finite-dimensional K-vector space. Let V^* denote the space of K-vector space morphisms $V \longrightarrow K$. Then V is isomorphic to V^*. Namely, if ψ is a non-degenerate bilinear form on $V \times V$, then ψ induces an isomorphism $\mathbf{a} \mapsto \psi(\mathbf{a}, \cdot)$ from V to V^*. If $W \subset V$ is a subspace, then the subspace of V^* consisting of maps that vanish on W has dimension $\dim_K V - \dim_K W$. This can be seen, for instance, by taking a basis of V that includes a basis of W. This shows the first claim.

Now set $V = \mathbb{F}_q^\ell$, $W = C$, and take as ψ the standard inner product on $\mathbb{F}_q^\ell$. Observe that $C^\perp$ corresponds to the K-linear forms vanishing on C.

By definition of the dual code,

$$C \subset (C^\perp)^\perp.$$

Since

$$\dim_{\mathbb{F}_q}(C^\perp)^\perp = \ell - \dim_{\mathbb{F}_q} C^\perp = \ell - (\ell - \dim_{\mathbb{F}_q} C) = \dim_{\mathbb{F}_q} C,$$

it follows by linear algebra that

$$(C^\perp)^\perp = C$$

This shows the second claim. □

LEMMA 11.63 *Let $C \subset \mathbb{F}_q^{n+1}$ be a linear secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$ on $n \geq 1$ players. Then $C^\perp \subset \mathbb{F}_q^{n+1}$ is also a linear secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$ on n players.*

PROOF The dual code $C^\perp$ is an $\mathbb{F}_q$-linear code as well. We verify that it satisfies the conditions of Definition 11.60. Suppose $\pi_0(C^\perp) = \{0\}$. Then $(1, 0 \ldots, 0) \in (C^\perp)^\perp$ by definition of the dual. By Lemma 11.62, it follows that $(1, 0 \ldots, 0) \in C$. This gives a contradiction with the second property of Definition 11.60 applied to C. Finally, suppose there is $\mathbf{x}^* \in C^\perp$ such that $x_0^* = 1$ but $\mathbf{x}^*_{\mathcal{I}^*} = \mathbf{0}$. Then, by definition of the dual code, this implies $\pi_0(C) = \{0\}$. This gives a contradiction with the first property of Definition 11.60 applied to C. □

11.10.2 Basic Results by Dualization

LEMMA 11.64 *Let $C \subset \mathbb{F}_q^{n+1}$ be a linear secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$ on $n \geq 1$ players. Let $B \subset \mathcal{I}^*$ be a nonempty set. Then the following statements are equivalent.*

1. *$B \in \Gamma(C)$.*
2. *For all $\mathbf{x} \in C$ it holds that if $\mathbf{x}_B = \mathbf{0}$ then $x_0 = 0$.*
3. *There is $\mathbf{x}^* \in C^\perp$ with $x_0^* = 1$ and $\mathbf{x}^*_{\overline{B}} = \mathbf{0}$.*

PROOF The equivalence of the first and second claims is argued in a similar way as it has already been argued for $B = \mathcal{I}^*$ right after Definition 11.60.

We now argue that the first claim implies the third. Fix some $B \in \Gamma(C)$. Define the map

$$\rho : \pi_B(C) \longrightarrow \mathbb{F}_q,$$

$$\mathbf{x}_B \mapsto x_0.$$

This is well-defined since $B \in \Gamma(C)$: if $\mathbf{x}_B = \mathbf{x}'_B$ for some $\mathbf{x}, \mathbf{x}' \in C$, then $x_0 = x'_0$. This map is clearly $\mathbb{F}_q$-linear. The third claim follows by basic linear algebra: e.g., extend ρ arbitrarily to an $\mathbb{F}_q$-linear form on $\mathbb{F}_q^{|B|}$. Suppose this form is given as $\sum_{i \in B} u_i X_i$, where $u_i \in \mathbb{F}_q$ for each $i \in B$. Then the vector $(u_i)_{i \in \mathcal{I}} \in \mathbb{F}_q^{n+1}$ with $u_0 = -1$ and $u_i = 0$ if $i \notin B$ satisfies the requirements.

To argue that the third claim implies the first claim, note that

$$x_0 = -\sum_{i \in B} x_i x_i^*$$

for all $\mathbf{x} \in C$. Hence, $B \in \Gamma(C)$. □

LEMMA 11.65 *Let $C \subset \mathbb{F}_q^{n+1}$ be a linear secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$ on n players. Let $B \subset \{1, \dots, n\}$ be a nonempty set. Then the following statements are equivalent.*

1. *$B \in \mathcal{A}(C)$.*
2. *There is $\mathbf{x} \in C$ such that $x_0 = 1$ and $\mathbf{x}_B = \mathbf{0}$.*

PROOF If $B \in \mathcal{A}(C)$, there is ambivalence, i.e, there are $\mathbf{x}', \mathbf{x}'' \in C$ with $\mathbf{x}'_B = \mathbf{x}''_B$ but $x'_0 \neq x''_0$. By linearity, this means there is $\mathbf{x} \in C$ such that $x_0 = 1$ and $\mathbf{x}_B = \mathbf{0}$. On the other hand, the existence of $\mathbf{x} \in C$ such that $x_0 = 1$ and $\mathbf{x}_B = \mathbf{0}$ implies that the map $C \longrightarrow \mathbb{F}_q \times \pi_B(C)$ defined by the assignment $\mathbf{x}' \mapsto (x'_0, \mathbf{x}'_B)$ is a surjective morphism. Indeed, $\mathbf{x}' + \lambda\mathbf{x}$ maps to $(x'_0 + \lambda, \mathbf{x}'_B)$ for all $\mathbf{x}' \in C$ and for all $\lambda \in \mathbb{F}_q$. By Lemma 11.46 it follows that $B \in \mathcal{A}(C)$. □

COROLLARY 11.66 *Let $C \subset \mathbb{F}_q^{n+1}$ be a linear secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$ on n players. Let $B \subset \mathcal{I}^*$. Then:*

- *$B \in \Gamma(C)$ if and only if $\overline{B} \in \mathcal{A}(C^\perp)$.*
- *$B \in \mathcal{A}(C)$ if and only if $\overline{B} \in \Gamma(C^\perp)$.*

THEOREM 11.67 (DUALIZATION) *Let $C \subset \mathbb{F}_q^{n+1}$ be a linear secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$ on n players. Then*

1. *The linear secret sharing scheme C is perfect.*
2. *$\Gamma(C^\perp) = (\Gamma(C))^*$.*

Proof Suppose $B' \subset \mathcal{I}^*$ is nonempty. The statement that there is $\mathbf{x} \in C$ such that $x_0 = 1$ and $\mathbf{x}_{B'} = \mathbf{0}$ is the negation of the statement that for all $\mathbf{x} \in C$ it holds that if $\mathbf{x}_{B'} = \mathbf{0}$, then $x_0 = 0$. Therefore, for all $B \subset \mathcal{I}^*$, it holds that

$$B \in \mathcal{A}(C) \text{ if and only if } B \notin \Gamma(C).$$

This proves the first claim. Applying Corollary 11.66, it follows that, for all $B \subset \mathcal{I}^*$,

$$B \in \Gamma(C^\perp) \text{ if and only if } \overline{B} \notin \Gamma(C).$$

This proves the second claim. □

Remark 11.6 If the access structure of a linear secret sharing scheme is connected, then the scheme is ideal.

Dualization in the context of secret sharing has been introduced in [167, 120].

Definition 11.68 *Let $C \subset \mathbb{F}_q^\ell$ be an $\mathbb{F}_q$-linear code with $\ell \geq 1$. Then $w_0(C)$ is the minimum of $w_{\mathrm{H}}(\mathbf{x})$ taken over all $\mathbf{x} \in C$ with $x_0 = 1$, if such $\mathbf{x}$ exists. If not, $w_0(C) = 0$ by default.*

Remark 11.7 As opposed to the minimum distance $d_{\min}(C)$, the parameter $w_0(C)$ depends on how the code is indexed. More precisely, it depends on which coordinate is labeled by 0.

Theorem 11.69 (Parameter Control from Linear Codes) *Let $C \subset \mathbb{F}_q^{n+1}$ be an $\mathbb{F}_q$-linear code with $n \geq 1$. If*

$$w_0(C) \geq 2$$

then C is a linear secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$ on n players with t-privacy and r-reconstruction such that

$$t = w_0(C^\perp) - 2 \quad \text{and} \quad r = n - w_0(C) + 2,$$

which is sharp in both cases. In particular, if

$$d_{\min}(C) \geq 2 \quad \text{and} \quad d_{\min}(C^\perp) \geq 2,$$

then there is t-privacy and r-reconstruction with

$$t = d_{\min}(C^\perp) - 2 \quad \text{and} \quad r = n - d_{\min}(C) + 2.$$

Proof From $w_0(C) \geq 2$ it follows that each $\mathbf{x} \in C$ with $x_0 \neq 0$ satisfies $\mathbf{x}_{\mathcal{I}^*} \neq \mathbf{0}$ and that such $\mathbf{x} \in C$ exists. From this observation it follows that C is a linear secret sharing scheme. Note that $w_0(C) \geq 2$ implies $w_0(C^\perp) \geq 2$, as $w_0(C^\perp) = 1$ implies $w_0(C) = 0$ and $w_0(C^\perp) = 0$ implies $w_0(C) = 1$.

As to reconstruction, let $\mathbf{x} \in C$. Suppose $\mathbf{x}_B = \mathbf{0}$ for some $B \subset \mathcal{I}^*$ with $|B| \geq n - w_0(C) + 2$. Then

$$w_{\mathrm{H}}(\mathbf{x}) \leq w_{\mathrm{H}}(x_0) + w_0(C) - 2 \leq w_0(C) - 1.$$

Therefore, it holds that $x_0 = 0$. On the other hand, suppose $\mathbf{x} \in C$ satisfies $x_0 = 1$ and $w_{\mathrm{H}}(\mathbf{x}) = w_0(C)$. Then there is a set $B \subset \mathcal{I}^*$ with $|B| = n - w_0(C) + 1$ such that $\mathbf{x}_B = \mathbf{0}$. Therefore, $B \in \mathcal{A}(C)$. Thus, there is no r'-reconstruction with $r' < n - w_0(C) + 2$. Hence, there is reconstruction as claimed, and the bound is sharp.

As to privacy, this follows immediately by dualizing the result on reconstruction. By Corollary 11.66, there is t-privacy in C if and only if there is $(n-t)$-reconstruction in $C^{\perp}$.

Finally, the conditions on minimum distance of C, resp. $C^{\perp}$, imply that $w_0(C) \geq 2$. To conclude, note that $d_{\min}(C) \leq w_0(C)$ and $d_{\min}(C^{\perp}) \leq w_0(C^{\perp})$. □

This theorem is not stated in [134, 135], but it follows from the results. Moreover, it appears to have been part of the "folklore". See also the discussion and constructions in [56, 43, 90].

Lemma 11.70 (Uniformity) *Let $C \subset \mathbb{F}_q^{\ell}$ be an $\mathbb{F}_q$-linear code with $\ell \geq 1$. Fix some index set $\mathcal{I}$ for $\mathbb{F}_q^{\ell}$. Suppose*

$$d_{\min}(C^{\perp}) \geq 2.$$

Let $B \subset \mathcal{I}$ with $|B| = d_{\min}(C^{\perp}) - 1$. Then

$$\pi_B : C \longrightarrow \mathbb{F}_q^{|B|}$$

is a surjective $\mathbb{F}_q$-vector space morphism.

Proof For any $\mathbf{x}^* \in \mathbb{F}_q^{\ell}$ with

$$\mathbf{x}^*_B \in (\pi_B(C))^{\perp} \subset \mathbb{F}_q^{|B|} \text{ and } \mathbf{x}^*_{\overline{B}} = \mathbf{0} \in \mathbb{F}_q^{\ell - |B|},$$

it holds that

$$\mathbf{x}^* \in C^{\perp}.$$

Since

$$w_{\mathrm{H}}(\mathbf{x}^*) \leq |B| < d_{\min}(C^{\perp}),$$

it must hold that $\mathbf{x}^* = \mathbf{0}$. Thus, $(\pi_B(C))^{\perp} = \{\mathbf{0}\}$, or equivalently, $\pi_B(C) = \mathbb{F}_q^{|B|}$. This proves the lemma. □

Corollary 11.71 (Uniformity of Shares) *Let $C \subset \mathbb{F}_q^{n+1}$ be a linear secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$ on $n \geq 1$ players. Suppose $d_{\min}(C^{\perp}) > 2$. Write $t = d_{\min}(C^{\perp}) - 2$. Then each set of t shares in the corresponding secret sharing scheme is uniformly random in $\mathbb{F}_q^t$.*

Remark 11.8 In applications, it is convenient to employ a generator matrix for the code C of a linear secret sharing scheme, i.e., a matrix G such that its (say, $e \geq 2$) columns form a basis of C (or at least generate C). This way, it is straightforward to sample shares for a given secret. Without loss of generality, G has the vector $(1, 0, \ldots, 0)$ as its first row. Then

$$G(s, \rho_1, \ldots, \rho_{e-1})^T = (s, x_1, \ldots, x_n)^T,$$

where $s \in \mathbb{F}_q$ is the secret and $\rho_1, \ldots, \rho_{e-1} \in \mathbb{F}_q$ are chosen independently and uniformly at random. The elements $x_1, \ldots, x_n$ are shares for the secret s.

Definition 11.72 (Ideal Linear Threshold Schemes) *Let t, n be integers with $0 \leq t < n$. Then $\mathrm{ITSS}_{\mathrm{lin}}(t, n, q)$ denotes the set of linear secret sharing schemes for $\mathbb{F}_q$ over $\mathbb{F}_q$ with t-privacy and $(t+1)$-reconstruction.*

The theorem below summarizes the results presented so far.

Theorem 11.73 *Let t, n be integers with $0 \leq t < n$. Then $\mathrm{ITSS}_{\mathrm{lin}}(t, n, q) \neq \emptyset$ if one of the following two conditions is satisfied.*

- $t = 0$ *or* $t = n - 1$.
- $1 \leq t < n - 1$ *and* $n \leq q$.

Proof The first claim follows by combining Theorem 11.58 with Lemma 11.3. The second claim follows by combining Theorem 11.58 with Theorem 11.12. □

11.10.3 Brickell's Scheme

Cast in Massey's framework, Brickell's earlier (but equivalent) *vector space construction* [32] of linear secret sharing schemes is as follows.

Definition 11.74 *Let V be an $\mathbb{F}_q$-vector space with $\dim_{\mathbb{F}_q} V < \infty$. Let V^* denote the dual space of V, i.e., space of $\mathbb{F}_q$-vector space morphisms $\phi : V \longrightarrow \mathbb{F}_q$. Let $z_0, z_1, \ldots, z_n$ vectors in V, for some positive integer n. Then the $\mathbb{F}_q$-linear code $C \subset \mathbb{F}_q^{n+1}$ is defined as*

$$C = \{(\phi(z_0), \phi(z_1), \ldots, \phi(z_n)) \mid \phi \in V^*\} \subset \mathbb{F}_q^{n+1}.$$

Clearly, this notion captures exactly the $\mathbb{F}_q$-linear codes. However, this definition enables an instructive analysis of the associated secret sharing scheme in terms of linear algebra. First we state a lemma from basic linear algebra.

Lemma 11.75 *Let V be a finite-dimensional vector space over a field K. Let $U \subset V$ be a subspace and let $x \in V$. Then $x \notin U$ if and only if there is a linear form $\phi \in V^*$ such that ϕ vanishes on U (i.e., ϕ is identically 0 on U) but $\phi(x) = 1$.*

Proof The claim in the forward direction can be verified as an immediate consequence of the fact that there is a basis of V that contains x as well as a basis of U. The other direction is trivial. □

LEMMA 11.76 *Let C be as defined above. It is an $\mathbb{F}_q$-linear secret sharing scheme on n players if and only if $z_0 \neq 0$ and the $\mathbb{F}_q$-span of $z_1, \ldots, z_n$ includes z_0.*

PROOF As to the proof that it is an $\mathbb{F}_q$-linear secret sharing scheme, it should be clear that the condition $\pi_0(C) = \mathbb{F}_q$ holds if and only if $z_0 \neq 0$. By inspection, the condition that for all $\mathbf{x} \in C$ it holds that if $\mathbf{x}_{\mathcal{I}^*} = \mathbf{0} \in \mathbb{F}_q^n$ then $z_0 = 0 \in \mathbb{F}_q$, is equivalent to the condition that for all $\phi \in V^*$ such that $\phi(z_i) = 0$ for $i = 1, \ldots, n$ it holds that $\phi(z_0) = 0$. By Lemma 11.75, the latter is equivalent to z_0 being in the $\mathbb{F}_q$-span of $z_1, \ldots, z_n$. $\square$

LEMMA 11.77 *Let C be as in Definition 11.74. Suppose $z_0 \neq 0$ and the $\mathbb{F}_q$-span of $z_1, \ldots, z_n$ includes z_0. Let $B \subset \{1, \ldots, n\}$ be nonempty. Then:*

1. *$B \in \Gamma(C)$ if and only if the $\mathbb{F}_q$-span of the z_i's with $i \in B$ includes z_0.*
2. *$B \in \mathcal{A}(C)$ if and only if there is $\phi \in V^*$ such that $\phi(z_i) = 0$ for all $i \in A$ and $\phi(z_0) = 1$.*

PROOF The proof of the first claim follows from the combination of Lemmas 11.64 and 11.75. The proof of the third claim follows directly from Lemma 11.65.

In the other direction, a linear secret sharing scheme in the sense of Massey implies one in the sense of Brickell. Namely, choose some matrix such that the columns form a basis for the linear code and take its $n + 1$ rows as $z_0, z_1, \ldots, z_n$. Thus, the two definitions are equivalent. However, Massey's formulation makes it easier to draw on results from coding theory when constructing secret sharing schemes with sufficient control over the t-privacy and r-reconstruction parameters. On the other hand, for certain specialized classes of access structures, it is often more convenient to think in terms of Brickell's original definition. $\square$

11.11 Generalizations of Linear Secret Sharing

We now discuss some generalizations of linear secret sharing schemes.

11.11.1 Linear Secret Sharing with Large Secret Space

DEFINITION 11.78 (LINEAR SECRET SHARING FOR $\mathbb{F}_q^k$ OVER $\mathbb{F}_q$) *Let $C \subset \mathbb{F}_q^{n+k}$ be an $\mathbb{F}_q$-linear code, where n, k are positive integers. Define $\mathcal{I} = \{-k+1, \ldots, 0, 1, \ldots, n\}$ as the index set. Define $\mathcal{I}^* = \{1, \ldots, n\}$ and define $\mathcal{Z} = \{-k + 1, \ldots, 0\}$. Then C is a linear secret sharing scheme for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ if the following conditions are satisfied.*

1. *$\pi_{\mathcal{Z}}(C) = \mathbb{F}_q^k$.*
2. *For all $\mathbf{x} \in C$ it holds that if $\mathbf{x}_{\mathcal{I}^*} = \mathbf{0} \in \mathbb{F}_q^n$ then $\mathbf{x}_{\mathcal{Z}} = \mathbf{0} \in \mathbb{F}_q^k$.*

By a similar analysis to the one following Definition 11.60, this gives rise to a secret sharing scheme.

As before, if $B \subset \mathcal{I}^*$, then $\overline{B} := \mathcal{I}^* \setminus B$. For $i = 1, \dots, k$, let $\mathbf{u}_i \in \mathbb{F}_q^k$ denote the i-th vector in the standard basis of $\mathbb{F}_q^k$, i.e., there is 1 in the i-th coordinate and 0 in all others. An easy adaptation of the arguments in the proofs of Lemmas 11.64 and 11.65, implies the following.

Lemma 11.79 *Let $B \subset \mathcal{I}^*$ be a nonempty set.*

- *$B \in \Gamma(C)$ if and only if, for each integer i with $1 \le i \le k$, there is $\mathbf{x}^* \in C^\perp$ such that*
$$\mathbf{x}^*_{\mathcal{Z}} = \mathbf{u}_i \quad \textit{and} \quad \mathbf{x}^*_{\overline{B}} = \mathbf{0} \in \mathbb{F}_q^{n-|B|}.$$
- *$B \in \mathcal{A}(C)$ if and only if, for each integer i with $1 \le i \le k$, there is $\mathbf{x} \in C$ such that*
$$\mathbf{x}_{\mathcal{Z}} = \mathbf{u}_i \quad \textit{and} \quad \mathbf{x}_B = \mathbf{0} \in \mathbb{F}_q^{|B|}.$$

Theorem 11.67 (dualization) does not hold for a scheme satisfying Definition 11.78 if $k > 1$ as it is not a perfect secret sharing scheme in that case. Indeed, the second condition above is not the negation of the first. In fact, using these two conditions it is easy to prove that if B is a privacy set and a set B' with $B \subset B'$ is a reconstructing set, then $|B'| - |B| \ge k$.

Remark 11.9 Using similar reasoning as in the proof of Theorem 11.67, the claim in Corollary 11.66 still holds.

Remark 11.10 There is an immediate translation of Definition 11.78 into the language of Brickell's scheme from Section 11.10.3. Clearly, this involves elements $z_{-k+1}, \dots, z_0, z_1, \dots, z_n$, instead of just the elements $z_0, z_1, \dots, z_n$. The characterizations of $\Gamma(C)$ and $\mathcal{A}(C)$ in Lemma 11.79 translate in the obvious way. The condition that $\pi_{\mathcal{Z}}(C) = \mathbb{F}_q^k$ translates to the condition that $z_{-k+1}, \dots, z_0$ are linearly independent.

Theorem 11.69 in combination with Lemma 11.70 implies (see also [56])

Theorem 11.80 (Generalized Parameter Control from Linear Codes) *Let n, k be positive integers. Let $C \subset \mathbb{F}_q^{n+k}$ be an $\mathbb{F}_q$-linear code. If*
$$d_{\min}(C) \ge k+1 \quad \textit{and} \quad d_{\min}(C^\perp) \ge k+1,$$
then it is a linear secret sharing scheme for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ with t-privacy and r-reconstruction such that
$$t = d_{\min}(C^\perp) - k - 1 \quad \textit{and} \quad r = n - d_{\min}(C) + k + 1.$$
Moreover, in the corresponding secret sharing scheme, any t shares are uniformly random in $\mathbb{F}_q^t$.

A more precise version of this theorem can be obtained by considering an appropriate, straightforward generalization of the approach in Theorem 11.70 based on the weights from Definition 11.68, but we refrain from spelling out the details here.

11.11.2 Further Generalizations

We now turn to a generalization in a different direction.

Definition 11.81 *A* generalized linear secret sharing scheme $\Sigma := (C, \Pi)$ *for* $\mathbb{F}_q$ *over* $\mathbb{F}_q$ *on* $n \geq 1$ *players consists of a linear secret sharing scheme* C *on* $m \geq n$ *players, indexed by the set* $\mathcal{J} = \{0, 1, \ldots, m\}$*, and a disjoint partition* Π *of the set* $\mathcal{J}^* = \{1, \ldots, m\}$ *by* n *nonempty subsets* $I_1, \ldots, I_n \subset \mathcal{J}$*. Define* $\mathcal{I} = \{0, 1, \ldots, n\}$ *and* $\mathcal{I}^* = \{1, \ldots, n\}$*, the* player set*.*

The idea is that in a linear secret sharing scheme according to Definition 11.60 the m share-coordinates are subdivided into n consecutive blocks, each of which corresponds to a different player in the generalization. In particular, the number of field elements a player gets as a share equals the cardinality of the block corresponding to that player.

In Chapter 6 this generalization was also considered and there we showed that such schemes can be used for multiparty computation in much the same way as Shamir's scheme. The advantage of the generalized schemes is that they can be constructed for any monotone access structure and over any field, as we shall see below.

We denote the access structure on the total of n players in the generalization as $\Gamma(\Sigma)$. Let $B \subset \mathcal{I}^*$ be a nonempty set. Observe that Σ is a *perfect* secret sharing scheme with

$$B \in \Gamma(\Sigma) \text{ if and only if } \cup_{j \in B} I_j \in \Gamma(C).$$

If we cast this generalization in the language of Brickell's scheme, we obtain *monotone span programs*, a notion introduced by Karchmer and Wigderson [120] that is well-studied in complexity theory. This connection was first noted in [12]. Even further generalization of this notion where the secret-space can have dimension greater than 1 were studied in [22] and [175].

Theorem 11.82 *Let* $\Sigma = (C, \Pi)$ *be a generalized linear secret sharing scheme for* $\mathbb{F}_q$ *over* $\mathbb{F}_q$*. Define* $\Sigma^* = (C^\perp, \Pi)$*. Then:*

1. Σ *is perfect.*
2. $\Gamma(\Sigma^*) = (\Gamma(\Sigma))^*$*.*

This theorem follows from the characterization of $\Gamma(\Sigma)$ above in combination with the results in Section 11.10.2. It is equivalent to the well-known fact that for any monotone Boolean function and for any finite field it holds that its monotone span program complexity equals that of its dual [120].

The relevance of generalized linear secret sharing schemes becomes apparent from the following theorem.

Theorem 11.83 *[118, 20] Let* Γ *be a monotone access structure on* $n \geq 1$ *players. Then there is a generalized linear secret sharing scheme* Σ *for* $\mathbb{F}_q$ *over* $\mathbb{F}_q$ *on* n *players such that* $\Gamma(\Sigma) = \Gamma$*.*

There are several ways in which this result can be shown. Typically, the access structure is decomposed into smaller pieces for which schemes are known and then these schemes are glued together to get a scheme for the composition. [14] The simplest (but "expensive") approach is that of [118], which we explain here [15]:

- Enumerate all the minimal sets of Γ, say this results in the sets $B_1, \ldots, B_\ell$.
- Let $s \in \mathbb{F}_q$ be the secret. For $i = 1, \ldots, \ell$ do the following.
  Select a fresh random $|B_i|$-out-of-$|B_i|$ additive secret sharing over $\mathbb{F}_q$ (see Section 11.1) of the secret s. Call this the *i-th sharing.* Assign each of the $|B_i|$ resulting shares to a different player in B_i.

Note that the length of the share-vector for a player is exactly the number of minimal sets this player sits in. The access structure is what it should be, as we verify below. By definition of a minimal set it holds that $B \in \Gamma$ if and only if there is some integer j with $1 \leq j \leq \ell$ such that $B_j \subset B$. Therefore, if $B \notin \Gamma$, then $B_i \setminus B$ is nonempty for all $1 \leq i \leq \ell$. Therefore, B *lacks* at least one out of the $|B_i|$ shares resulting from the i-th sharing (for all $1 \leq i \leq \ell$). Since these sharings are independently generated, this means that B gets no information on the secret s from the total information available to B. On the other hand, if $B \in \Gamma$, then there is some integer j with $1 \leq j \leq \ell$ such that $B_j \subset B$. Hence, B has in particular all the $|B_j|$ shares from the j-th sharing of s and can reconstruct the secret s.

Generally, this leads to very inefficient secret sharing schemes. For instance, consider the (t, n)-threshold access structure with, say, $t \approx n/2$. Then the share-length in the scheme above is *exponential* in n due to the sheer magnitude of the quantity $\binom{n}{t}$ for these values of the parameters.

An approach [20] which is in several cases much better in this respect is based on *monotone Boolean formulas* from complexity theory. A monotone Boolean formula is a finite directed acyclic graph such that:

- At the top, there are precisely $n \geq 1$ vertices without any incoming edges, the *input vertices.* These vertices are numbered $1, \ldots, n$ and each of them may have an arbitrary number of outgoing edges going down.
- At the bottom, there is precisely one vertex without any outgoing edges. It has just a single incoming edge from above. This is the *output vertex.*
- In between, each *computation vertex* has two incoming edges from above and one outgoing edge going down. Such a vertex is labeled either with the Boolean logical AND-operator or with the Boolean logical OR-operator.

By definition, the logical AND and OR operate on bits $b, b' \in \{0, 1\}$. The AND produces $c = 1$ as the outgoing bit if the incoming bits $b, b' \in \{0, 1\}$ satisfy $b = b' = 1$ and it produces $c = 0$ as the outgoing bit otherwise. An OR operator

[14] See [170] for results on composition.
[15] In Chapter 6 one can find an alternative proof based on so-called replicated secret-sharing. This is not more efficient in general, however.

produces $c = 1$ if $b = 1$ or $b' = 1$ and it produces $c = 0$ otherwise. If a bit is represented as an element of, say, $\mathbb{F}_2$, the AND corresponds to $c := bb' \in \mathbb{F}_2$ and the OR corresponds to $c := bb' + b + b' \in \mathbb{F}_2$.

Under this definition, a monotone Boolean formula represents a Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ as follows. Given $(b_1, \ldots, b_n) \in \{0,1\}^n$, label the i-th input vertex with b_i for $i = 1, \ldots, n$. Now start the "computation" by pushing the bits $b_1, \ldots, b_n$ inductively down the graph: each time a logical operator is encountered, the incoming bits b, b' are converted to an outgoing bit c that represents the result of the respective logical operation on b, b'. This process is continued until the output vertex receives a bit, which is declared $f(b_1, \ldots, b_n)$.

Note that the simplest examples are just those formulas that consist of a single AND or a single OR: two input vertices, one computation vertex, and one output vertex. In general, formulas are just compositions of these two basic building blocks, leading to a single output vertex. Disregarding the directedness, as a graph it is a tree. By basic graph theory this means that the total number of edges equals the number of vertices minus 1. The size of a formula is the number of computation vertices.

This function f computed by the formula is *monotone* in the sense that if $(b_1, \ldots, b_n) \in \{0,1\}^n$ is such that $f(b_1, \ldots, b_n) = 1$, then flipping any 0-bit among $b_1, \ldots, b_n$ (if there is any 0-bit there) to a 1-bit does not affect the f-value, i.e., it remains 1. Each monotone Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ are computed by some monotone Boolean formula.

One way to see this is to note first that monotone access structures correspond naturally with monotone Boolean functions simply by representing a set by its characteristic bit-vector and declaring the f-value to be 1 if and only if the set in question sits in the access structure. It is an easy task to construct a monotone Boolean formula computing a given monotone access structure (i.e, if one doesn't care too much about the resulting size). For instance, a construction can be based on a slight generalization of the idea behind the scheme above [118]: construct the formula as "a (multi-input) OR over the minimal sets, each of which is in turn given as a (multi-input) AND". The conversion to a formula in which the computation gates have incoming degree 2 is straightforward.

Working inductively in the *reverse* direction, i.e., from the bottom to the top in a monotone Boolean formula, a linear secret sharing scheme can be constructed whose access structure corresponds to the monotone Boolean function computed by the formula in question. Place the secret s at the output vertex. If the output vertex receives the output of a computation from an AND computation vertex directly above it, generate a random 2-out-of-2 additive secret sharing for s and send share s_0 up the left input-edge of that preceding computation vertex and send s_1 up the right one. If it was an OR instead, just send the secret s up instead, both on the left and the right. Now repeat this process for s_0 and, independently, for s_1, etc. until each input vertex at the top has received an element through each one of its outgoing edges. The vector of elements an input vertex thus receives constitutes its share in the secret s. It is not so hard to prove by induction that this generalized linear secret sharing scheme has the access structure as claimed.

To illustrate why this approach is more powerful than that of [118], consider the following example. As it is known [174] that the majority function, i.e., the Boolean function $f : \{0,1\}^n \longrightarrow \{0,1\}$ that gives 1 if and only if the Hamming-weight of the argument is strictly greater than $n/2$ can be computed by a polynomial (in n) size monotone Boolean formula, this results in a generalized linear secret sharing scheme for the majority access structure where each of the n individual share-vector has length polynomial in n. [16]

The approach can be further extended by allowing more general types of monotone gates in the formula, not just ORs and ANDs. For instance, if in addition small threshold gates are allowed (such as 2-out-of-3 or 2-out-of-4), the formula-based approach sketched here has applications to multi-party computation secure against a *general adversary*, i.e., one that is not necessarily constrained by a threshold condition on which subsets of the network may be corrupted. [17]

By a counting argument, most access structures admit only exponentially inefficient linear secret sharing schemes. This is generally believed to be true for nonlinear schemes as well, though a proof is lacking at present. More precisely, the minimal length of the shares in secret sharing schemes for general (non-threshold) access structures has been conjectured to grow exponentially on the number of players. Nevertheless, the best known lower bounds are still very far off from establishing this. Another related open problem is to determine which access structures admit an ideal secret sharing scheme. These open problems have attracted considerable attention during the last few decades, and a rich theory has been developed, in which matroid theory plays a pivotal role. For more information on this, refer to [13, 148].

11.11.3 Extension Field Interpolation

Towards an improvement of multi-secret sharing scheme from Section 11.4.2, we generalize Theorem 11.6, Lagrange's Interpolation Theorem. Other applications are given in Chapter 12.

THEOREM 11.84 *Fix an algebraic closure $\overline{K}$ of K. Suppose $\alpha_1, \ldots, \alpha_m \in \overline{K}$ have the property that if $m > 1$ then their respective minimal polynomials $h_i(X) \in K[X]$ are pair-wise distinct.* [18] *For $i = 1, \ldots, m$, define*

$$\delta_i = \deg h_i(X) \ (= \dim_K K(\alpha_i)).$$

[16] Of course, Shamir's scheme gives a much better result. But our point here is just to show a separation.

[17] For applications, see [68]. An application to security against threshold adversaries is given in [58].

[18] Equivalently, α_i, α_j are not Galois-conjugate over K if $i \neq j$.

Moreover, define

$$M = \sum_{i=1}^{m} \delta_i.$$

As before, let $K[X]_{\leq M-1}$ *denote the* K*-vector space of polynomials* $f(X) \in K[X]$ *such that* $\deg f \leq M - 1$.

Then the evaluation map

$$\mathcal{E} : K[X]_{\leq M-1} \longrightarrow \bigoplus_{i=1}^{m} K(\alpha_i)$$

$$f(X) \mapsto (f(\alpha_i))_{i=1}^{m}$$

is an isomorphism of K*-vector spaces.*

Proof Since the K-dimensions on both sides are identical, it is sufficient to argue injectivity. Let $f(X) \in K[X]_{\leq M-1}$ with $f(X) \not\equiv 0$. Suppose that, for $i = 1, \ldots, m$, it holds that

$$f(\alpha_i) = 0.$$

Since the polynomials $h_1(X), \ldots, h_m(X)$ are pairwise co-prime, it is the case that

$$h_1(X) \cdots h_m(X) \mid f(X),$$

i.e., their product divides $f(X)$ in $K[X]$. But this implies

$$\deg f \geq M,$$

a contradiction. Alternatively, the theorem can also be shown easily using the Chinese Remainder Theorem.

For completeness we give a direct construction of the inverse $\mathcal{E}^{-1}$. Suppose first that $m = 1$, and set $\alpha_1 = x$. If $x \in K$, then $K(x) = K$ and $M = 1$. Hence, the claim is a special case of Theorem 11.6. If $x \notin K$, then the claim follows directly from the fact that there is the basis

$$1, x, \ldots, x^{k-1} \in K(x),$$

of $K(x)$ as a K-vector space, where

$$k = \dim_K K(x) > 1.$$

If $y \in K(x)$, then $\mathcal{E}(f) = y$ where $f(X)$ is the polynomial whose coefficient-vector is the coordinate-vector of y according to the basis above.

Now suppose $m > 1$. For $i = 1, \ldots, m$, define

$$\delta_i'(X) = \prod_{1 \leq k \leq m, k \neq i} h_k(X) \ \in K[X]$$

and

$$z_i = \delta_i'(\alpha_i) = \prod_{1 \leq k \leq m, k \neq i} h_k(\alpha_i) \ \in K(\alpha_i).$$

For all $1 \leq i, j \leq m$ with $i \neq j$, it follows that

$$\delta'_i(\alpha_j) = 0 \ , \ \deg(\delta'_i) = M - \dim_K K(\alpha_i).$$

Moreover, by assumption on the α_i's, it holds that $h_j(\alpha_i) \neq 0$ if $i \neq j$. Therefore,

$$z_i \neq 0,$$

for $i = 1, \ldots, m$.

Let

$$\mathbf{y} \in \bigoplus_{i=1}^{m} K(\alpha_i),$$

For $i = 1, \ldots, m$, select

$$f_i(X) \in K[X]$$

such that

$$\deg(f_i) \leq \dim_K K(\alpha_i) - 1 \ , \ f_i(\alpha_i) = \frac{y_i}{z_i}.$$

By the claim for the case $m = 1$, these exist. Then define

$$f(X) = \sum_{i=1}^{m} \delta'_i(X) f_i(X) \in K[X].$$

It follows that

$$\deg(f) \leq \max_{1 \leq i \leq m} ((M - \dim_K K(\alpha_i)) + (\dim_K K(\alpha_i) - 1)) = M - 1.$$

and that, for $j = 1, \ldots, m$,

$$f(\alpha_j) = \sum_{i=1}^{m} \delta'_i(\alpha_j) f_i(\alpha_j) = \delta'_j(\alpha_j) f_j(\alpha_j) = z_j \frac{y_j}{z_j} = y_j.$$

□

11.11.4 Multi-Secret Sharing Revisited

We give an example of a secret sharing scheme in the sense of Definition 11.78. In the "multi-secret sharing scheme" from Section 11.4.2, the length of the secret can be increased at the expense of the following two consequences.

- A growing gap between privacy and reconstruction.
- A decrease in the maximum number of players that can be accommodated.

We show that the latter of the two consequences above can be avoided.

Theorem 11.84 gives rise to the following improvement [55] of the "multi-secret sharing scheme" from Section 11.4.2.

Suppose t, k, n are integers such that with $t \geq 0$, $k > 1$ and $t + k < n \leq q$. Let $\alpha_0 \in \mathbb{F}_{q^k}$ be such that $\mathbb{F}_q(\alpha_0) = \mathbb{F}_{q^k}$, and let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ be pairwise distinct.

Consider

$$C = \{(f(\alpha_0), f(\alpha_1), \ldots, f(\alpha_n)) \mid f(X) \in \mathbb{F}_q[X], \deg(f) \leq t+k\} \subset \mathbb{F}_{q^k} \oplus \left(\bigoplus_{i=1}^{n} \mathbb{F}_q \right).$$

Take $\{0, 1, \ldots, n\}$ as the index set $\mathcal{I}$ of C. Let $s \in \mathbb{F}_{q^k}$ be the secret. Select $\mathbf{x} \in C$ uniformly random such that $x_0 = s$, and declare $x_1, \ldots, x_n$ as the shares.

By Theorem 11.84, it follows that, for each $s \in \mathbb{F}_{q^k}$, there exists $\mathbf{x} \in C$ with $x_0 = s$ and that and that there is $(t+k+1)$-reconstruction. On the same account, for each triple $(s, B, \mathbf{u})$ where $s \in \mathbb{F}_{q^k}$, $B \subset \mathcal{I}^*$ with $|B| = t$, and $\mathbf{u} \in \mathbb{F}_q^t$, there is a unique $\mathbf{x} \in C$ such that $x_0 = s$ and $\mathbf{x}_B = \mathbf{u}$. On account of Lemma 11.46, it follows that B is a privacy set. Hence, there is t-privacy.

Finally, it should be clear that this scheme fits Definition 11.78: even though the secret space is the finite field $\mathbb{F}_{q^k}$, this is isomorphic to $\mathbb{F}_q^k$ as an $\mathbb{F}_q$-vector space.

11.12 Bounds for Secret Sharing

We show several limitations on the parameters of a secret sharing scheme.

11.12.1 Lower Bound on the Length of Shares

Shortening gives an immediate lower bound on the length of the shares in a perfect, connected secret sharing scheme $\mathbf{S}$, as follows.

THEOREM 11.85 *([121]) Suppose* $\mathbf{S}$ *is perfect and connected. Let* $j \in \mathcal{I}^*$. *Then*

$$H(S_j) \geq H(S_0) = \log_2 |\mathcal{S}_0|.$$

In particular,

$$|\mathcal{S}_j| \geq |\mathcal{S}_0|.$$

PROOF Let $j \in \mathcal{I}^*$. If $\{j\} \in \Gamma(\mathbf{S})$, then $H(S_0|S_j) = 0$ and the claim follows directly by Lemma 11.34. Now suppose $\{j\} \notin \Gamma(\mathbf{S})$ and let $B \in \Gamma(\mathbf{S})$ be a minimal set such that $j \in B$. Then

$$H(S_0|\mathbf{S}_B) = 0$$

and that

$$H(S_0) = H(S_0|\mathbf{S}_{B\setminus\{j\}}).$$

By Lemma 11.34, it follows that

$$H(S_0) = H(S_0|\mathbf{S}_{B\setminus\{j\}}) \leq H(S_0 S_j|\mathbf{S}_{B\setminus\{j\}}) =$$

$$H(S_j|\mathbf{S}_{B\setminus\{j\}}) + H(S_0|S_j\mathbf{S}_{B\setminus\{j\}}) = H(S_j|\mathbf{S}_{B\setminus\{j\}}) \leq H(S_j).$$

$\square$

Remark 11.11 This theorem motivates Definition 11.57.

Again by Lemma 11.34, we have the following corollary.

Corollary 11.86 *Suppose* $\mathbf{S}$ *is an ideal secret sharing scheme defined over an alphabet* $\mathcal{F}$ *of cardinality* $\bar{q}$. *Then* S_j *gives the uniform distribution on* $\mathcal{F}$ *for all* $j \in \mathcal{I}$.

11.12.2 Shortening of a Secret Sharing Scheme

We explain the general principle of *shortening* of secret sharing schemes. This is often a useful handle for proofs by induction. Throughout this section, let $\mathbf{S}$ be a secret sharing scheme indexed by a set $\mathcal{I}$.

Definition 11.87 (Shortening) *Suppose* $B \in \mathcal{A}(\mathbf{S})$ *with* $B \neq \emptyset$ *and* $\mathbf{u} \in \mathcal{S}_B$ *satisfy* $P(\mathbf{S}_B = \mathbf{u}) > 0$. *Then* $\mathbf{S}^{B;\mathbf{u}}$ *is the secret sharing scheme obtained by conditioning the random variable* $\mathbf{S}$ *on the event* $\mathbf{S}_B = \mathbf{u}$ *and deleting the random variables corresponding to* B. *Thus, there are* $|\mathcal{I}^*| - |B| \geq 1$ *players left. We say that* $\mathbf{S}$ *is shortened at* $(B; \mathbf{u})$.

Another way of describing the construction of $\mathbf{S}^{B;\mathbf{u}}$ is as follows. [19] Suppose B is a nonempty privacy set and suppose that $\mathbf{u}$ is a share-vector for B that is assigned with non-zero probability when sampling from $\mathbf{S}$. The shortened scheme then samples from $\mathbf{S}$ conditioned on the event that B gets $\mathbf{u}$. The corresponding shares for $\mathcal{I}^* \setminus B$ are assigned but the ones for B are deleted.

It is easy to see that this is in fact a secret sharing scheme. Since S_0 is independently distributed from $\mathbf{S}_B$, it holds that $P(S_0 = s|\mathbf{S}_B = \mathbf{u}) = P(S_0 = s)$ for all $s \in \mathcal{S}_0$. Thus, the secret space is unchanged and there is still the uniform probability distribution on it. The share-spaces are unchanged as well.

Finally, the set $\mathcal{I}^* \setminus B$ clearly reconstructs in $\mathbf{S}^{B;\mathbf{u}}$: by construction, any "full share vector" according to $\mathbf{S}^{B;\mathbf{u}}$ determines a unique "full share vector" according to $\mathbf{S}$ with $\mathbf{S}_B = \mathbf{u}$. This in turns determines the secret since $\mathbf{S}$ is a secret sharing scheme. In particular, $r(\mathbf{S}^{B;\mathbf{u}}) \leq r(\mathbf{S}) - |B|$ for this reason. We argue about privacy below.

Lemma 11.88 *([48]) If* $t(\mathbf{S}) \geq 1$ *and* $1 \leq |B| \leq t(\mathbf{S})$, *then*

- $r(\mathbf{S}^{B;\mathbf{u}}) \leq r(\mathbf{S}) - |B|$,
- $t(\mathbf{S}^{B;\mathbf{u}}) \geq t(\mathbf{S}) - |B|$.

[19] For secret sharing this notion was first introduced in [133, 92] under the name of *contraction*. Our treatment here is from [48] and has been inspired by a classic notion from the theory of linear codes that goes by the same name (not to be confused with *puncturing*, which just means ignoring selected coordinates). See e.g. [112].

Proof The statement about $r(\mathbf{S}^{B;\mathbf{u}})$ has been argued above. If $|B| = t(\mathbf{S})$, then $t(\mathbf{S}^{B;\mathbf{u}}) \geq 0$ by definition. Now suppose $|B| < t(\mathbf{S})$. Let $B' \subseteq \mathcal{I}^* \setminus B$ be such that $|B'| = t(\mathbf{S}) - |B|$. Since $|B \cup B'| = t(\mathbf{S})$, it holds that $\mathbf{S}_{B \cup B'}$ is independent from S_0. Then, for each $\mathbf{v} \in \mathcal{S}_B$ satisfying $P(\mathbf{S}_{B'} = \mathbf{v} | \mathbf{S}_B = \mathbf{u}) > 0$, it holds that

$$P(S_0 = s, \mathbf{S}_{B'} = \mathbf{v} | \mathbf{S}_B = \mathbf{u}) =$$

$$P(S_0 = s | \mathbf{S}_{B'} = \mathbf{v}, \mathbf{S}_B = \mathbf{u}) \cdot P(\mathbf{S}_{B'} = \mathbf{v} | \mathbf{S}_B = \mathbf{u}) =$$

$$P(S_0 = s | \mathbf{S}_B = \mathbf{u}) \cdot P(\mathbf{S}_{B'} = \mathbf{v} | \mathbf{S}_B = \mathbf{u}),$$

where the first equality follows from elementary calculus of conditional probabilities, and the second because $B, B \cup B'$ are privacy sets of $\mathbf{S}$. We have proved that S_0 and $\mathbf{S}_{B'}$ are independent when conditioned on the event $\mathbf{S}_B = \mathbf{u}$, and this is the same as saying that B' is a privacy set in the scheme $\mathbf{S}^{B;\mathbf{u}}$. $\square$

Remark 11.12 More generally, suppose the pair $(B, \mathbf{u})$ with $B \in \mathcal{A}(\mathbf{S})$ and $B \neq \emptyset$ and with $\mathbf{u} \in \mathcal{S}_B$ satisfies $P(\mathbf{S}_B = \mathbf{u}) > 0$. Then $\Gamma(\mathbf{S}^{B;\mathbf{u}})$ includes all sets $B' \subset \mathcal{I}^* \setminus B$ such that $B \cup B' \in \Gamma(\mathbf{S})$ and $\mathcal{A}(\mathbf{S}^{B;\mathbf{u}})$ includes all sets $B' \subset \mathcal{I}^* \setminus B$ such that $B \cup B' \in \mathcal{A}(\mathbf{S})$.

11.12.3 ITSS–IC–MDS Equivalence

We now show the well-known result that ideal threshold secret sharing, interpolation codes and so-called *maximum distance separable (MDS) codes* are all essentially equivalent. We will use these equivalences later on to show bounds on ideal threshold secret sharing.

Theorem 11.89 *([132, 26]) Let $t, n, \bar{q}$ be integers with $0 \leq t < n$ and $\bar{q} \geq 2$. Suppose $\mathbf{S} = (S_0, S_1, \ldots, S_n)$ is an ideal (t, n)-threshold secret sharing scheme defined over an alphabet $\mathcal{F}$ of cardinality $\bar{q}$. Then:*

1. *The code $C := \{\mathbf{x} \in \mathcal{F}^{n+1} : P(\mathbf{S} = \mathbf{x}) > 0\}$ corresponding to $\mathbf{S}$ is a $(t+1, n+1, \bar{q})$-interpolation code.*
2. *$\mathbf{S}_B$ gives the uniform distribution on $\mathcal{F}^{t+1}$ for each $B \subset \{0, 1, \ldots, n\}$ with $|B| = t+1$. Consequently, since C is a $(t+1, n+1, \bar{q})$-interpolation code, $\mathbf{S}$ gives the uniform distribution on C.*

Proof The proof is by induction on t. If $t = 0$, then similar reasoning as in the proof of Corollary 11.86 shows that, for each $j \in \mathcal{I}^*$, the variables S_0 and S_j mutually determine each other. Therefore, if $n > 1$, it holds that, for each $i, j \in \mathcal{I}^*$ with $i \neq j$, there is a similar one-to-one correspondence between S_i and S_j. Since S_0 has the uniform distribution on $\mathcal{F}$, each of the claims follows. Now suppose $t > 0$.

As to the first claim, select arbitrary $j \in \mathcal{I}^*$, $u \in \mathcal{F}$. Now shorten the scheme at $(\{j\}, u)$. By Corollary 11.86, this is well-defined. By Lemma 11.88 this results in an ideal $(t-1, n-1)$-threshold scheme over the alphabet $\mathcal{F}$ of cardinality $\bar{q}$.

By the induction hypothesis, the code $C^{\{j\};u}$ corresponding to this scheme is a $(t, n, \bar{q})$-interpolation code. It follows that C is a $(t+1, n+1, \bar{q})$-interpolation code, since $j \in \mathcal{I}^*$ and $u \in \mathcal{F}$ are arbitrary and $t > 0$. As to the second claim, this follows similarly by induction. Briefly, S_j gives the uniform distribution on $\mathcal{F}$ by Corollary 11.86 and after shortening there is "t-wise uniformity" on $\mathbf{S}^{\{j\};u}$ by the induction hypothesis. This implies "$(t+1)$-wise uniformity" for $\mathbf{S}$. □

Combining this with Theorem 11.58 gives a strong equivalence between ideal threshold secret sharing and interpolation codes.

Lemma 11.90 (Singleton Bound) *Let C be a code of length $\ell \geq 1$ over an alphabet of cardinality $\bar{q} \geq 2$. Suppose $|C| > 1$. Then*

$$|C| \leq \bar{q}^{\ell-d+1},$$

where $d := d_{\min}(C)$.

Proof Write $\mathcal{F}$ for the alphabet and write $\mathcal{I}$ for an index set. If $d = 1$, the bound is trivial since $C \subset \mathcal{F}^{\ell}$. Now suppose $d > 1$. Fix $B \subset \mathcal{I}$ arbitrarily with $|B| = \ell - (d-1)$. Consider the projection map that sends $\mathbf{x} \in C$ to $\mathbf{x}_B \in \mathcal{F}^{\ell-d+1}$. This map is injective; otherwise there are $\mathbf{x}, \mathbf{y} \in C$ with $\mathbf{x} \neq \mathbf{y}$ and $d_{\mathrm{H}}(\mathbf{x}, \mathbf{y}) \leq d-1$, which is nonsense. The bound follows. □

Definition 11.91 (MDS Code) *A code C with $|C| > 1$ of length ℓ and minimum distance d over an alphabet of cardinality $\bar{q}$ is* Maximum Distance Separable (MDS) *if it attains the Singleton Bound, i.e., $|C| = \bar{q}^{\ell-d+1}$. The class of all MDS codes with length ℓ and minimum distance d, defined over an alphabet of cardinality $\bar{q}$, is denoted* $\mathrm{MDS}(k, \ell, \bar{q})$*, where $k := \ell - d + 1$.*

It is straightforward that interpolation codes and MDS codes are equivalent, as shown in the following lemma.

Lemma 11.92 *Let $r, \ell, \bar{q}$ be integers with $1 \leq r \leq \ell$ and $\bar{q} \geq 2$. Then $C \in \mathrm{IC}(r, \ell, \bar{q})$ if and only if $C \in \mathrm{MDS}(r, \ell, \bar{q})$.*

Proof In the forward direction, let $C \in \mathrm{IC}(r, \ell, \bar{q})$. Then $|C| = \bar{q}^r$. Moreover, any two distinct words in C agree in at most $r-1$ coordinates. Hence, $d \geq \ell - r + 1$. This is sharp, as there exist two distinct words in C that agree in exactly $r-1$ coordinates.

In the other direction, let $C \in \mathrm{MDS}(r, \ell, \bar{q})$. Write $d = \ell - r + 1$. By definition, it holds that $d = d_{\min}(C)$. The case $r = \ell$ is trivial since this means $C = \mathcal{F}^{\ell}$. Now suppose $r < \ell$, so that $d > 1$. Select $B \subset \mathcal{I}$ arbitrarily such that $|B| = \ell - (d-1)$. Consider the projection map that sends $\mathbf{x} \in C$ to $\mathbf{x}_B \in \mathcal{F}^{\ell-d+1}$. This map is injective for the same reason as in the proof of Lemma 11.90. Since $|C| = \bar{q}^r = \bar{q}^{\ell-d+1}$ by definition, this projection map constitutes a bijection. □

On account of Theorem 11.89, Theorem 11.58, and Lemma 11.92, we have the following equivalences.

Theorem 11.93 (ITSS-IC-MDS Equivalence) *Let $t, n, \bar{q}$ be integers with $0 \leq t < n$ and $\bar{q} \geq 2$. Then the following three statements are equivalent.* [20]

- $\text{ITSS}(t, n, \bar{q}) \neq \emptyset$. [21]
- $\text{IC}(t+1, n+1, \bar{q}) \neq \emptyset$.
- $\text{MDS}(t+1, n+1, \bar{q}) \neq \emptyset$.

This result also holds when restricted to the $\mathbb{F}_q$-linear case.

11.12.4 Upper Bounds on Length for Fixed Alphabet Size in ITSS

We show the fundamental limitation that the number of players n in an ideal (t, n)-threshold secret sharing scheme is bounded as a function of the cardinality $\bar{q}$ of the alphabet if the trivial cases $t = 0, n-1$ are excluded.

We present two results. The first, which asserts that $n < 2\bar{q} - 1$ whether the scheme is linear or not, combines the fact that ideal threshold secret sharing schemes are *equivalent* to interpolation codes with an application of the classical Bush bound [35] for orthogonal arrays $O(r, \ell, \bar{q})$ (= interpolation codes) and a bound on pairwise orthogonal Latin squares.

The second result is a direct application of S. Ball's breakthrough result [3] in combinatorics and coding theory which affirms the longstanding linear MDS Conjecture *in the case of prime fields*, in combination with the fact that ideal (linear) threshold secret sharing schemes are equivalent to (linear) *maximum distance separable (MDS) codes*. As a consequence, the existence of ideal $\mathbb{F}_p$-linear threshold schemes (p prime) is completely understood. Namely, excluding the trivial case $t = 0, n-1$, a (t, n)-scheme exists if *and only if* $n \leq p$.

We start with some preparations for the first result.

Definition 11.94 *Let $\bar{q} \geq 2$ be an integer. A* Latin square *of order $\bar{q}$ is a square matrix M with entries in the set $[\bar{q}] := \{1, \ldots, \bar{q}\}$ such that each element of $[\bar{q}]$ occurs exactly once in each column (resp., row).*

Note that M is a $\bar{q} \times \bar{q}$ matrix.

Definition 11.95 *Suppose M, M' are Latin squares of order $\bar{q}$. Then M, M' are* orthogonal *if the $\bar{q}^2$ ordered pairs (M_{ij}, M'_{ij}) with $1 \leq i, j \leq \bar{q}$ are all distinct, i.e., they are in one-to-one correspondence with the set $[\bar{q}] \times [\bar{q}]$.*

There is the following classical bound for orthogonal Latin squares (see e.g. [178]).

Lemma 11.96 *Suppose $M_1, M_2, \ldots, M_w$ are pairwise orthogonal Latin squares of order $\bar{q}$. Then $w < \bar{q}$.*

[20] We have refrained from repeating the explicit correspondences between the equivalent classes below.
[21] For the definition, see Section 11.9.4.

PROOF Having a permutation act on the symbols $\{1,\dots,\bar{q}\}$ of some M_i and replacing M_i by the resulting Latin square does not affect pairwise orthogonality. Therefore, we may assume without loss of generality that the top row of each M_i is the vector $(1,2,\dots,\bar{q})$. Now consider the entries $x_1,\dots,x_w$, where x_i is the $(2,1)$-entry in M_i $(1 \le i \le w)$. The following two observations together justify the claimed bound. First, by the Latin-square property, none of the x_i's is equal to 1, since the top left corner of each M_i is 1. Second, since the top rows are identical, pairwise orthogonality implies that $x_i \neq x_j$ for all $1 \le i < j \le w$. □

THEOREM 11.97 (BUSH BOUND [35]) *Suppose* $\mathrm{IC}(r,\ell,\bar{q}) \neq \emptyset$*, for some integers* $r,\ell,\bar{q}$ *with* $2 \le r \le \ell-2$ *and* $\bar{q} \ge 2$*. Then* $r < \bar{q}$*.*

PROOF By application of Lemma 11.4 (puncturing), we may assume that $\mathrm{IC}(r,r+2,\bar{q}) \neq \emptyset$ as well. Select some code in this class and consider the subcode consisting of the following words. Without loss of generality the symbols $0,1$ are in the alphabet.

1. (*Category I*) Among the first $r-1$ coordinates there are exactly $r-2$ occurrences of 0, whereas the r-th coordinate is 0.
2. (*Category II*) The first $r-1$ coordinates are 0 and the r-th coordinate is 0 as well.
3. *(Category III)* The first $r-1$ coordinates are 0, whereas the r-th coordinate is 1.

For each of these categories, it is by the interpolation property that the "tail", i.e., the pair consisting of the last two coordinates (the $(r+1)$-st and the $(r+2)$-nd), is uniquely determined by the first r coordinates. So, categories II and III each contain a single element. Write (a,b) and (a',b') for their respective tails. Again by interpolation, $a \neq a'$ and $b \neq b'$. By counting, category I consists of exactly $(\bar{q}-1)(r-1)$ words. For each of those, if its tail is given by (x,y), then $x \neq a$ and $y \neq b$ and $(x,y) \neq (a',b')$ by interpolation. Moreover, within these categories, two tails cannot be equal, once again by interpolation. It follows that

$$(\bar{q}-1)(r-1) \le (\bar{q}-1)^2 - 1,$$

which is equivalent to the claimed bound. □

LEMMA 11.98 *Let* $\ell,\bar{q}$ *be integers* $\ell \ge 1$ *and* $\bar{q} \ge 2$*. Suppose* $\mathrm{IC}(2,\ell,\bar{q}) \neq \emptyset$*. Then* $\ell < \bar{q}+2$*.*

PROOF If $\ell < 4$, then the claim is trivially true. So assume $\ell \ge 4$. Consider any code in $\mathrm{IC}(2,\ell,\bar{q})$. Such a code gives rise to $\ell-2 \ge 2$ pairwise orthogonal Latin squares of order $\bar{q}$ as follows. Interpret its words $\mathbf{x}$ as vectors of the form $(i,j,L_1(i,j),\dots,L_{\ell-2}(i,j))$, where (i,j) indicates the (i,j)-position in a $\bar{q}\times\bar{q}$ matrix and $L_u(i,j)$ gives the entry in the u-th matrix at the (i,j)-position $(u = 1,\dots,\ell-2)$. The interpolation property implies that these are well-defined, are

Latin squares and are pairwise orthogonal. The claimed bound now follows by application of Lemma 11.96. □

We are now in the position to state and prove the first bound.

Theorem 11.99 (Upper Bound on Length in ITSS) *Let $t, n, \bar{q}$ be integers with* $1 \leq t < n-1$ *and* $\bar{q} \geq 2$. *Suppose* $\mathrm{ITSS}(t, n, \bar{q}) \neq \emptyset$. *Then* $n < 2\bar{q} - 2$.

Proof Apply Theorem 11.89 and let $C \in \mathrm{IC}(t+1, n+1, \bar{q})$. Since $(n+1) - (t+1) \geq 2$ and $t+1 \geq 2$, by Theorem 11.97 it follows that

$$t + 1 < \bar{q}.$$

Next, apply Lemma 11.4 (shortening) to obtain a code $C'' \in \mathrm{IC}(2, n+2-t, \bar{q})$, where $n+2-t \geq 4$. This is possible since $t+1 \geq 2$. By Lemma 11.98,

$$n + 2 - t < \bar{q} + 2.$$

The claimed bound follows by combining these two bounds. □

We remark that *in the $\mathbb{F}_q$-linear case*, there is the following alternative proof. It is straightforward that existence of an $\mathbb{F}_q$-linear MDS code of length ℓ and dimension k implies existence of a set of ℓ vectors in $\mathbb{F}_q^k$ such that each k of them constitute an $\mathbb{F}_q$-basis of $\mathbb{F}_q^k$, and vice-versa. Namely, in the forward direction, construct a matrix whose collection of rows constitutes an $\mathbb{F}_q$-basis for the given MDS code. The set of column vectors has the claimed property. In the other direction, appropriately reverse this process.

Suppose we are given a set of ℓ vectors in $\mathbb{F}_q^k$ such that each k of them constitute an $\mathbb{F}_q$-basis of $\mathbb{F}_q^k$, with $2 \leq k \leq \ell$. Then there is the fundamental bound that $\ell \leq q+k-1$. Indeed, select any $k-2 \geq 0$ vectors and observe (from [3]) that there are exactly $q+1$ hyperplanes H containing those $k-2$ vectors (select $\mathbf{0}$ when $k=2$). Namely, complement the selected $k-2$ vectors with two extra vectors to get an $\mathbb{F}_q$-basis of $\mathbb{F}_q^k$; the hyperplanes H in question correspond one-to-one with the $q+1$ lines (passing through the origin) in the plane spanned by those two extra vectors. Moreover, each of those $q+1$ hyperplanes H contains at most one of the other $\ell-k+2$ vectors (as otherwise the dimension of H would be k instead of $k-1$), while for each of those vectors there is clearly such a hyperplane H containing it, by construction. In conclusion, $\ell - k + 2 \leq q + 1$, or equivalently, $\ell \leq q + k - 1$.

Taking the dual of a linear MDS code gives a linear MDS code, as is easy to see. [22] Assuming that the condition $k < \ell - 1$ also holds, the dimension of the dual is $\ell - k \geq 2$. Applying the bound derived above to the dual of C, it follows that $\ell \leq q + (\ell - k) - 1$. Combining the two, it follows that $\ell \leq 2q - 2$. [23]

We will state (but not prove – this is beyond the present scope) a bound for

[22] See for instance [176], p. 101, Theorem 6.8.3.

[23] Yet another proof is by inspection of the weight distribution of linear MDS codes (which is known) in combination with a similar dualization argument. See [176], p. 102–103 (combine Theorem 6.8.8 with Corollary 6.8.9).

ideal linear threshold sharing that is a direct consequence of S. Ball's recent breakthrough result [3] in combinatorics and coding theory which affirms the longstanding linear MDS Conjecture *in the case of prime fields.*

CONJECTURE 11.100 *(see e.g. [112], p. 265) Suppose C is an $\mathbb{F}_q$-linear code of length $\ell \geq 4$ and dimension k with $1 < k < \ell - 1$. If C is an MDS code then $\ell \leq q+1$ except when q is even and ($k = 3$ or $k = q-1$), in which case $\ell \leq q+2$.*

THEOREM 11.101 ([3]) *The linear MDS Conjecture is true if q is a prime number.*

Combining this with Theorems 11.93 and 11.73, the case of ideal linear secret sharing over prime fields is completely solved.

THEOREM 11.102 (FULL CHARACTERIZATION OF THE $\mathbb{F}_p$-LINEAR CASE) *Let t, n, p be integers with $0 \leq t < n$ and $p \geq 2$ a prime number. Then* $\mathrm{ITSS}_{\mathrm{lin}}(t, n, p) \neq \emptyset$ if and only if *one of the following two conditions is satisfied:*

1. $t = 0$ *or* $t = n - 1$.
2. $1 \leq t < n - 1$ *and* $n \leq p$.

11.12.5 A Lower Bound on the Threshold Gap

Theorem 11.97 shows that the number n of players in an ideal (t, n)-threshold secret sharing scheme is bounded as a function of the cardinality $\bar{q}$ of the alphabet as long as $1 \leq t < n - 1$. Therefore, if the number of players in a secret sharing scheme is sufficiently large compared to the cardinality of the alphabet, there must be a "gap" between privacy and reconstruction that is greater than 1. We now show some general bound on the magnitude of this gap [48]. In Section 12.6.4 we show some consequences of this bound.

Let $\mathbf{S}$ be a secret sharing scheme, indexed by some set $\mathcal{I}$. As in Section 11.9.2, let $r(\mathbf{S})$ (resp., $t(\mathbf{S})$) denote the minimum r (resp., maximum t) such that $\mathbf{S}$ has r-reconstruction (resp., t-privacy). Furthemore, let $n(\mathbf{S})$ denote the cardinality of the player set $\mathcal{I}^*$. Also, $\lambda^*(\mathbf{S})$ denotes the average Shannon-entropy of the shares.

DEFINITION 11.103 (THRESHOLD GAP) *The* threshold gap *of* $\mathbf{S}$ *is*

$$g(\mathbf{S}) := r(\mathbf{S}) - t(\mathbf{S}).$$

Note that $\mathbf{S}$ is a threshold secret sharing scheme if $g(\mathbf{S}) = 1$.

THEOREM 11.104 *[48] Suppose $t(\mathbf{S}) \geq 1$. Then*

$$g(\mathbf{S}) \geq \frac{n(\mathbf{S}) - t(\mathbf{S}) + 1}{2^{\lambda^*}}.$$

REMARK 11.13 This bound becomes invalid if $t(\mathbf{S})$ is substituted by a lower bound. For instance, the trivial lower bound $1 \leq t(\mathbf{S})$ would give $g(\mathbf{S}) \geq n(\mathbf{S})/\bar{q}$. This is false in the case of ideal $(n-1, n)$-threshold secret sharing schemes with $n > \bar{q}$.

We will prove the following weaker corollary.

COROLLARY 11.105 *[48] Suppose $t(\mathbf{S}) \geq 1$. Furthermore, suppose that each share-space has cardinality $\bar{q}$, for some positive integer $\bar{q}$. Then*

$$g(\mathbf{S}) \geq \frac{n(\mathbf{S}) - t(\mathbf{S}) + 1}{\bar{q}}.$$

PROOF For the sake of this proof, write $t := t(\mathbf{S}),\ r := r(\mathbf{S}),\ g := g(\mathbf{S})$. Write $\mathbf{S}^*$ for the joint shares, i.e., the vector of random variables $(S_j)_{j \in \mathcal{I}^*}$. We first show that

$$r \geq \frac{n}{\bar{q}} + 1.$$

Let s, s' be distinct elements of the secret-space $\mathcal{S}_0$. Since $t \geq 1$, the probability distributions of $S_j|(S_0 = s)$ and $S_j|(S_0 = s')$ coincide for each $j \in \mathcal{I}^*$.

For each $j \in \mathcal{I}^*$, this implies that if we sample $x_j \in \mathcal{S}_j$ according to $S_j|(S_0 = s)$ and $y_j \in \mathcal{S}_j$ according to $S_j|(S_0 = s')$, then the collision probability satisfies [24]

$$P(x_j = y_j) \geq \frac{1}{|\mathcal{S}_j|} = \frac{1}{\bar{q}}.$$

Therefore, if $\mathbf{x} \in \prod_{j \in \mathcal{I}^*} \mathcal{S}_j$ is sampled according to $\mathbf{S}^*|(S_0 = s)$ and, independently, $\mathbf{y} \in \prod_{j \in \mathcal{I}^*} \mathcal{S}_j$ is sampled according to $\mathbf{S}^*|(S_0 = s')$, then the expected Hamming distance between $\mathbf{x}$ and $\mathbf{y}$ satisfies

$$\mathrm{E}(d_{\mathrm{H}}(\mathbf{x}, \mathbf{y})) \leq (1 - \frac{1}{\bar{q}}) \cdot n,$$

by linearity of expectation.

Consequently, there are $\mathbf{x}, \mathbf{y}$ such that

1. $\mathbf{x}$ (resp., $\mathbf{y}$) has positive probability of showing up when sampling from $\mathbf{S}^*|(S_0 = s)$ (resp., $\mathbf{S}^*|(S_0 = s')$).
2. the Hamming distance between $\mathbf{x}, \mathbf{y}$ is at most $(1 - \frac{1}{\bar{q}}) \cdot n$, i.e., $\mathbf{x}, \mathbf{y}$ have at least $\frac{n}{\bar{q}}$ coordinates in common.

Since their respective secrets differ, it must hold that $r \geq \frac{n}{\bar{q}} + 1$, as desired. Note that $r \geq \frac{n}{\bar{q}} + 1$ is equivalent to $g \geq \frac{n-t+1}{\bar{q}}$ if $t = 1$.

Now assume $t > 1$. Shorten the scheme arbitrarily at some $t(\mathbf{S}) - 1$ coordinates, resulting in a scheme $\mathbf{S}'$ on $n(\mathbf{S}) - t(\mathbf{S}) + 1$ players. By Lemma 11.88,

$$r(\mathbf{S}') \leq r(\mathbf{S}) - t(\mathbf{S}) + 1$$

[24] This is verified at once using the Cauchy-Schwarz Inequality, a special case of which says that, for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, it holds that $\langle \mathbf{x}, \mathbf{y} \rangle \leq \langle \mathbf{x}, \mathbf{x} \rangle \cdot \langle \mathbf{y}, \mathbf{y} \rangle$.

and

$$t(\mathbf{S}') \geq 1.$$

Application of the bound above then yields

$$r(\mathbf{S}') \geq \frac{n(\mathbf{S}) - t(\mathbf{S}) + 1}{\bar{q}} + 1.$$

Therefore,

$$g(\mathbf{S}) = r(\mathbf{S}) - t(\mathbf{S}) = (r(\mathbf{S}) - t(\mathbf{S}) + 1) - 1 \geq r(\mathbf{S}') - 1 \geq \frac{n(\mathbf{S}) - t(\mathbf{S}) + 1}{\bar{q}},$$

which is the claimed bound. □

We note that some earlier lower bounds on the gap are implied by [27, 92, 147, 87] (see [48] for a discussion). The bound given here, however, gives a better result for small alphabets (which is what we are interested in here). [25]

The bound from Theorem 11.104 can be improved for linear secret sharing schemes, as follows.

Corollary 11.106 *[48] Suppose* $\mathbf{S}$ *is an* $\mathbb{F}_q$*-linear secret sharing scheme. Furthermore, suppose that* $1 \leq t(\mathbf{S}) < r(\mathbf{S}) \leq n - 1$*. Then*

$$g(\mathbf{S}) \geq \frac{n(\mathbf{S}) + 2}{2q - 1}.$$

Proof Let $C \subset \mathbb{F}_q^{n+1}$ be an $\mathbb{F}_q$-linear code realizing the scheme. By Theorem 11.67, the gap of the scheme $\mathbf{S}^\perp$ induced by $C^\perp$ equals $g(\mathbf{S})$ and $t(\mathbf{S}^\perp) = n(\mathbf{S}) - r(\mathbf{S}) \geq 1$. Application of the bound from Corollary 11.105 gives

$$g(\mathbf{S}) \geq \frac{r(\mathbf{S}) + 1}{q}.$$

Combining this the bound from Corollary 11.105 applied to $\mathbf{S}$, gives

$$2g \geq \frac{n - t + 1 + r + 1}{q} = \frac{n + 2 + g}{q}$$

and the desired bound follows. □

Remark 11.14 (Generalizations) The bound from this corollary also applies to the linear secret sharing schemes for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ from Definition 11.78 on account of Remark 11.9. It also applies to the generalized linear secret sharing schemes for $\mathbb{F}_q$ over $\mathbb{F}_q$ from Definition 11.81, on account of Theorem 11.82 (dualization). In the latter case the share-spaces do not necessarily have the same cardinality. Therefore, in Corollary 11.106, the definition of the parameter q must be replaced by the average over the values q^{m_i}, where m_i is the cardinality of the i-th set from the disjoint partition Π defined there ($i = 1, \ldots, n$).

[25] A special case of this bound, namely when $g(\mathbf{S}) = 1$, was previously shown in unpublished work by Joe Kilian and Noam Nisan in 1990. See [48].

This leads, for instance, to an alternative proof of the fact [120] that the binary monotone span program complexity of the majority function on n input bits is $\Omega(n \log n)$.

11.13 Interpolation over Rings and Blackbox Secret Sharing

So far we have considered interpolation over fields and secret sharing over finite fields. We now study interpolation over general commutative rings and present an application of interpolation over algebraic number fields to *blackbox secret sharing schemes.*

11.13.1 Interpolation and Admissible Sets

Throughout this section, let A be a nontrivial commutative ring.

The main purpose of this section is to show that Lagrange interpolation over general commutative rings can be characterized exactly in terms of *admissible sets*, which are defined right below. This characterization will follow by a combination of Theorem 11.108 and Lemma 11.114 below. Moreover, we show some relevant bounds involving admissible sets.

Definition 11.107 (Admissible Set) *Let $T \subset A$ be a nonempty set. Then T is* admissible *if, for all $a, b \in T$, it holds that if $a \neq b$, then $a - b$ is a unit in A. If $|T| = 1$, the set T is admissible by default.*

As a generalization of Definition 11.5, for each integer $m \geq 0$, let $A[X]_{\leq m}$ denote the set consisting of polynomials $f(X) \in A[X]$ such that $\deg f(X) \leq m$. Note that $A[X]_{\leq m}$ is a free A-module of rank $m+1$.

Theorem 11.108 (Lagrange Interpolation over Commutative Rings)

Suppose $T \subset A$ is admissible. Then the evaluation map

$$\mathcal{E} : A[X]_{\leq |T|-1} \longrightarrow \bigoplus_{\alpha \in T} A,$$

$$f(X) \mapsto (f(\alpha))_{\alpha \in T}$$

is an isomorphism of A-modules.

Proof The claim is trivial if $|T| = 1$. So assume $|T| > 1$. We follow the approach from Section 11.6.1, based on the CRT. We only point out the differences. Since T is admissible, the ideals $I_\alpha := (X - \alpha) \subset A[X]$ with $\alpha \in T$ are pairwise co-prime in $A[X]$. Namely, if $\alpha \neq \alpha'$, then the ideal $I_\alpha + I_{\alpha'} \subset A[X]$ contains the unit $\alpha - \alpha'$. By the CRT,

$$A[X] / \prod_{\alpha \in T} (X - \alpha) \simeq \prod_{\alpha \in T} A[X]/(X - \alpha).$$

It is sufficient to show the following.

1. The classes on the l.h.s. are in one-to-one correspondence with the polynomials in $A[X]_{\leq |T|-1}$.
2. For all $f(X) \in A[X]$ and for all $\beta \in A$, it holds that $f(X) \equiv f(\beta) \pmod{X-\beta}$.

As to the first property, by Theorem 11.109 below, each element on the l.h.s. is represented by some polynomial of degree at most $|T| - 1$. By Corollary 11.111 below, exploiting the fact that the difference of any two distinct elements of the admissible set T is, in particular, nonzero and not a zero divisor, it follows that distinct polynomials of degree at most $|T| - 1$ represent distinct classes. The second property follows by Corollary 11.110 below. This concludes the proof.

On the constructive side, for each $\alpha \in T$, consider the polynomial

$$\delta_\alpha(X) = \prod_{\substack{\alpha' \in T: \\ \alpha' \neq \alpha}} \frac{X - \alpha'}{\alpha - \alpha'} \in Q(A)[X],$$

where $Q(A)$ is the field of fractions of A. By assumption on T, the denominators are units in A. Therefore, for each $\alpha \in T$,

$$\delta_\alpha(X) \in A[X].$$

It follows that the inverse of $\mathcal{E}$ is the map

$$\mathcal{E}^{-1} : \bigoplus_{\alpha \in T} A \longrightarrow A[X]_{\leq |T|-1},$$

$$(z_\alpha)_{\alpha \in T} \mapsto \sum_{\alpha \in T} z_\alpha \cdot \delta_\alpha(X).$$

Note that by the discussion in Section 11.6.1, the definition of the inverse map can also be viewed as an immediate inheritance of the CRT. □

Of course, the theorem above can also be proved via Vandermonde-determinants in combination with basic results on linear systems of equations over rings. Moreover, it can easily be generalized so as to "include evaluations at the point at infinity," just as in Theorem 11.11.

Theorem 11.109 (Euclidean Algorithm) *Let $f(X), g(X) \in A[X]$ be such that $f, g \neq 0$. Suppose that the leading coefficient of $g(X)$ is a unit in A. Then there exist unique $q(X), r(X) \in A[X]$ such that*

1. $f(X) = r(X) + q(X)g(X)$.
2. $\deg r(X) < \deg g(X)$.

For a proof, see e.g. [126].

The corollary below is a special case of this theorem.

Corollary 11.110 *Suppose $f(X) \in A[X]$ and $a \in A$ are such that $f \neq 0$ and $f(a) = 0$. Then $f(X) = (X - a)q(X)$ for a unique polynomial $q(X) \in A[X]$.*

Note that $\deg q(X) = \deg f(X) - 1$ in the corollary above.

Corollary 11.111 *Let $f(X) \in A[X]$ with $f \neq 0$. Suppose $T' \subset A$ is a nonempty set such that the following holds.*

1. *If $a, b \in T'$ and $a \neq b$, then $a - b$ is not a zero divisor in A.*
2. *For each $a \in T'$, it holds that $f(a) = 0$.*

Then $|T'| \leq \deg f(X)$.

Proof Proof by induction on $|T'|$. The claim is trivial if $|T'| = 1$. So assume $|T'| > 1$. Let $b \in T'$. Let $q(X) \in A[X]$ be such that $f(X) = (X - b)q(X)$. Let $a \in T' \setminus \{b\}$. Then

$$0 = f(a) = (a - b)q(a).$$

Since $a - b$ is nonzero and not a zero divisor in A, it follows that $q(a) = 0$. Now apply the induction hypothesis to the pair $(q(X), T' \setminus \{b\})$. It follows that $|T'| - 1 \leq \deg q(X)$. Hence, $|T'| \leq \deg q(X) + 1 = \deg f(X)$. $\square$

Assuming that Corollary 11.111 is understood for the case of fields, here is how one can argue the remaining cases without further recourse to the Euclidean algorithm. If A is a field, then the corollary expresses nothing else than the fact that a polynomial of degree $n \geq 0$ has at most n distinct zeros. [26] If A is a domain (i.e., it has no zero divisors), the claim follows by passing to its quotient field $Q(A)$. If A is not a domain, let $Z(A) \subset A$ consist of the zero divisors together with 0. [27] Then $Z(A)$ contains a prime ideal P of A. [28] Consider the quotient-ring A/P. Write $\bar{f}(X)$ for the image of $f(X)$ in $(A/P)[X]$ under reduction modulo P. Similarly, write $\bar{T}'$ for the image of T' in A/P. Clearly, it holds that $\bar{f}(\bar{a}) = 0$ in A/P, for all $\bar{a} \in \bar{T}'$. By the conditions of the corollary, for all $a, b \in T'$ with $a \neq b$, it holds that $\bar{a} - \bar{b} \neq 0$ in A/P. Hence, each element of $\bar{T}'$ is a root of $\bar{f}(X)$ and $|\bar{T}'| = |T'|$. Since A/P is a domain and $\deg \bar{f}(X) \leq \deg f(X)$, the claim follows.

For applications to blackbox secret sharing, we need a slightly more general version of Theorem 11.108, which we present below. Let M be an A-module. [29] Since $M \otimes A[X]_{\leq w-1}$ is a free A-module of rank w, Lemma 10.16 implies that there is a bijection between the elements of $M \otimes A[X]_{\leq w-1}$ and the expressions of the form $\sum_{i=0}^{w-1} f_i \otimes X^i$ with $f_i \in M$ $(i = 0, \ldots, w - 1)$.

[26] Note that the zero-polynomial $f = 0$ has $|A|$ distinct zeroes, but its degree is $-\infty$ by definition. Hence, this does not contradict the claim.

[27] This is in general *not* an ideal. For instance, in $\mathbb{Z}/6\mathbb{Z}$, the element $\overline{2} + \overline{3}$ is a unit and, hence, not a zero divisor.

[28] If a set $S \subset A$ is closed under multiplication and $0 \notin S$, the set of ideals that avoid S is not empty, since it contains (0). Moreover, a maximal element P (in the sense of the partial ordering by inclusion) is a prime ideal of A. If A is finite, then existence of a such a maximal element is clear. Otherwise, it follows from Zorn's Lemma. For the details, see e.g. [126] (associated primes on p. 416). Now set $S = A \setminus Z(A)$ (the non-zero elements that are not zero-divisors), and note it satisfies the conditions.

[29] For our purposes, it will be convenient to write here the A-scalar multiplication of the A-module M "on the right," i.e., $m \cdot a$, instead of "on the left," i.e., $a \cdot m$. Thus, M is (tacitly assumed to be a) right-A-module.

Definition 11.112 *Let M be an A-module. Let w be a positive integer. For all $\mathbf{f} \in M \otimes A[X]_{\leq w-1}$ and $x \in A$, define*

$$\mathbf{f}(x) = \sum_{i=0}^{w-1} f_i \cdot x^i \ \in M,$$

where

$$\mathbf{f} = \sum_{i=0}^{w-1} f_i \otimes X^i,$$

with $f_i \in M$ ($i = 0, \ldots, w-1$).

Theorem 11.113 (Lagrange Interpolation over A-Modules) *Let M be an A-module. Suppose $T \subset A$ is an admissible set. Consider the evaluation map*

$$\mathcal{E}_{\otimes} : M \otimes_A A[X]_{\leq |T|-1} \longrightarrow \bigoplus_{\alpha \in T} M,$$

$$\mathbf{f} \mapsto (\mathbf{f}(\alpha))_{\alpha \in T}.$$

Then $\mathcal{E}_{\otimes}$ is an isomorphism of A-modules.

Proof The proof of the theorem essentially comes down to a straightforward combination of Theorem 11.108 with Lemma 10.17: in the notation of that lemma, set $M = M'$, $N = A[X]_{\leq |T|-1}$ and $N' = \bigoplus_{\alpha \in T} A$, set the map ϕ to the identity map on M and set the map ψ to the isomorphism $\mathcal{E} : A[X]_{\leq |T|-1} \longrightarrow \bigoplus_{\alpha \in T} A$ from Theorem 11.108. This yields an isomorphism

$$\phi \otimes \psi : M \otimes_A A[X]_{\leq |T|-1} \longrightarrow M \otimes_A \bigoplus_{\alpha \in T} A.$$

Moreover, once again by Lemma 10.16, there is the isomorphism

$$\chi : M \otimes_A \bigoplus_{\alpha \in T} A \longrightarrow \bigoplus_{\alpha \in T} M,$$

by A-linear extension of the assignment

$$m \otimes (a_\alpha)_{\alpha \in T} \mapsto (m \cdot a_\alpha)_{\alpha \in T}.$$

Therefore, the map

$$\chi \circ (\phi \otimes \psi) : M \otimes_A A[X]_{\leq |T|-1} \longrightarrow \bigoplus_{\alpha \in T} M$$

is an isomorphism. By inspection,

$$\chi \circ (\phi \otimes \psi) = \mathcal{E}_{\otimes}.$$

This concludes the proof. Note that the inverse of $\mathcal{E}_{\otimes}$ is the map

$$\mathcal{E}_{\otimes}^{-1} : \bigoplus_{\alpha \in T} M \longrightarrow M \otimes_A A[X]_{\leq |T|-1},$$

$$(z_\alpha)_{\alpha\in T} \mapsto \sum_{\alpha\in T} z_\alpha \otimes \delta_\alpha(X),$$

where the $\delta_\alpha(X)$'s are as in the proof of Theorem 11.108. $\square$

As is the case with Theorem 11.108, the theorem above can also be proved via Vandermonde-determinants in combination with basic results on linear systems of equations over rings.

Lemma 11.114 (Necessity of Admissibility) *Let $\widetilde{T} \subset A$ be a nonempty set. Replace the admissible set $T \subset A$ in Theorem 11.108 by $\widetilde{T}$. Suppose the corresponding map $\widetilde{\mathcal{E}}$ is an isomorphism. Then $\widetilde{T}$ is admissible.*

Proof We may assume $|\widetilde{T}| > 1$. We give two proofs. The first one is only sketched. Let J be a maximal ideal in A. Using similar reasoning as in the proof of Theorem 11.113 (using the general version of base extension), the map $\widetilde{\mathcal{E}}$ induces an isomorphism

$$(A/J)[X]_{|\widetilde{T}|-1} \longrightarrow \bigoplus_{\alpha\in\widetilde{T}} A/J$$

that works just by reduction modulo J. This means that the pairwise differences of the elements in $\widetilde{T}$ are units. Indeed, if not, then some difference is contained in a maximal ideal J and the induced map above is not an isomorphism: the evaluations points are not pairwise distinct and, hence, the dimension of the image is strictly smaller than $|\widetilde{T}|$.

We now give a more explicit proof. Let $x \in \widetilde{T}$. Consider the polynomial

$$g(X) = \prod_{\substack{x'\in\widetilde{T}:\\ x'\neq x}} (X - x') \ \in A[X].$$

By the hypothesis, let $f(X) \in A[X]$ be the unique polynomial of degree at most $|\widetilde{T}| - 1$ such that, for all $x' \in \widetilde{T}$ with $x \neq x'$,

$$f(x) = 1 \text{ and } f(x') = 0.$$

Then, for all $z \in \widetilde{T}$,

$$g(x)\cdot f(z) = g(z).$$

By the hypothesis, this means that the polynomial identity

$$g(x)\cdot f(X) = g(X)$$

holds. Since $\deg g(X) = |\widetilde{T}| - 1$ and $\deg f(X) \leq |\widetilde{T}| - 1$, it holds that $\deg f(X) = |\widetilde{T}| - 1$ as well. Since $g(X)$ is monic, it follows that $g(x)$ is a unit. Hence, $x - x'$ is a unit, for all $x' \in \widetilde{T}$ with $x \neq x'$. Since $x \in \widetilde{T}$ is arbitrary, the claim follows. $\square$

We now show explicit bounds on the cardinality of admissible sets for some given rings A.

Definition 11.115 (Lenstra Constant) *The quantity $\lambda(A)$ denotes the supremum of $|T|$, where T ranges over all admissible subsets of A.*

We first give some straightforward examples. In general, $\lambda(A) \geq 2$ since $0, 1 \in A$ and $0 \neq 1$ by assumption. If A is a field, then A itself is admissible. Thus, for example, $\lambda(\mathbb{Q}) = \infty$ and $\lambda(\mathbb{F}_q) = q$. It is immediate that $\lambda(\mathbb{Z}) = 2$ since, by the pigeonhole principle, among any three integers there are at least two of them having the same residue class modulo 2. Hence, the difference of such two is an even number and therefore not a unit.

Let $p \geq 2$ be a prime and let $k \geq 1$ be an integer. Then $\lambda(\mathbb{Z}/p^k\mathbb{Z}) = p$. Namely, the set consisting of the classes of $0, 1, \ldots, p-1$ is admissible. Once again by the pigeonhole principle, any set of size $> p$ is not admissible. Applying the Chinese Remainder Theorem, it follows that $\lambda(\mathbb{Z}/n\mathbb{Z}) = p'$, where $n \geq 2$ is an integer and $p' > 0$ is the smallest prime number with $p'|n$. If A is an $\mathbb{F}_q$-algebra such that $\dim_{\mathbb{F}_q} A < \infty$ and such that 0 is its only nilpotent, then A is a product of finite extension fields of $\mathbb{F}_q$ by Theorem 10.11. Hence, $\lambda(A)$ equals the cardinality of the smallest of these fields.

In general, the quantity $\lambda(A)$ may be upper bounded as follows.

Lemma 11.116 *For each proper ideal I in A, it holds that*

$$\lambda(A) \leq |A/I|.$$

Proof Without loss of generality, assume that $|A/I| < \infty$. By the pigeonhole principle, a set $T \subset A$ with $|T| > |A/I|$ contains distinct elements $a, b \in T$ such that $a \equiv b \bmod I$, or equivalently, $a - b \in I$. Since I is a proper ideal, this implies that such $a - b$ is not a unit. $\square$

Remark 11.15 Since each proper ideal I is contained in some maximal ideal J and since $|A/J| \leq |A/I|$ for such I, J, it follows that attention may be restricted to the maximal ideals when upper bounding $\lambda(A)$ by the approach of Lemma 11.116.

We give an application to the theory of algebraic number fields. Let K be a number field of degree n. Since $\mathcal{O}_K$ is a free $\mathbb{Z}$-module of rank n, it follows that

$$|\mathcal{O}_K/2\mathcal{O}_K| \leq 2^n.$$

Hence,

$$\lambda(\mathcal{O}_K) \leq 2^n < \infty.$$

More specifically, let $p \geq 2$ be a prime number and let K be the p-th cyclotomic number field, with ring of integers $\mathcal{O}_K$. If $p = 2$, then $K = \mathbb{Q}$. Hence, $\lambda(\mathcal{O}_K) = 2$ in this case. If $p > 2$, then $\lambda(\mathcal{O}_K)$ can be determined as follows. Let $\omega \in \mathbb{C}$ be a primitive p-th root of unity. Consider the set

$$T = \{0, 1, 1 + \omega, \ldots, 1 + \omega + \cdots + \omega^{p-2}\} \subset \mathcal{O}_K.$$

This is an admissible set of cardinality p. Indeed, the difference between any two distinct elements in this set is of the form $1+\cdots+\omega^i$ for some i with $0 \leq i \leq p-2$ or of the form $\omega^j + \cdots + \omega^i$ for some i, j with $1 \leq j < i \leq p-2$. Dividing the latter by ω^j, they fall into the former category. By Lemma 10.5, it follows that T is an admissible set. Hence, $\lambda(\mathcal{O}_K) \geq p$.

Let P be the unique prime ideal of $\mathcal{O}_K$ that lies above p. Since p is totally ramified in $\mathcal{O}_K$ (see Section 10.7.4), the Fundamental Identity (see Section 10.7.3) gives $|\mathcal{O}_K/P| = p$. By Lemma 11.116, it follows that $\lambda(\mathcal{O}_K) \leq p$. Combining the two bounds, it follows that

$$\lambda(\mathcal{O}_K) = p.$$

Remark 11.16 Suppose we restrict, in Definition 11.115, the admissible sets T to those with the additional property that each element of T is a unit. Writing $\lambda'(A)$ for the corresponding constant, it is easy to see that

$$\lambda(A) = \lambda'(A) + 1.$$

Namely, let T be an admissible set with $|T| > 1$ and fix $a \in T$. Then the set

$$U_a = \{a - b \mid b \in T, b \neq a\}$$

fits the requirements. In the other direction, set $T = U \cup \{0\}$. In the case of the ring of integers in a number field, such sets of units whose pairwise differences are units are called (sets of) *exceptional units.* [30]

11.13.2 Blackbox Secret Sharing

A (t, n)*-threshold blackbox secret sharing* scheme provides a "universal" method for performing (t, n)-threshold secret sharing over an *arbitrary finite abelian group* G. Given a secret element of G and uniformly random elements of G, each share (or share vector) is obtained by $\mathbb{Z}$-linear combination(s) of the secret and those random elements. Similarly, reconstruction of the secret from shares is by taking $\mathbb{Z}$-linear combinations. This makes sense, since G is a $\mathbb{Z}$-module.

The salient feature of blackbox secret sharing, however, is that the $\mathbb{Z}$-linear combinations are *oblivious* of the actual finite abelian group over which the secret sharing is performed. In particular, there is no information required on further structural properties of the group, such as its order, decomposition, etc. to operate the scheme. It is in this sense that we have called such a scheme "universal" (with respect to the class of finite abelian groups).

Note that $\mathbb{Z}$-scalar multiplication can be achieved by repeated addition. [31] It is more efficient, however, to apply the "double-and-add" method, which is the additive version of the well-known "square-and-multiply" method for fast

[30] These play a significant role in the theory of *Euclidean number fields* [128].

[31] i.e., for each $a \in \mathbb{Z}$ and $g \in G$, the operation $a \cdot g$ can be computed as $g + \cdots + g$ (a times) if $a \geq 0$ and as $(-g) + \cdots + (-g)$ ($|a|$ times) if $a < 0$.

exponentiation. [32] This way, the number of group operations is logarithmic in the absolute value of the (nonzero) integer scalar.

A *toy example* to illustrate the principle is provided by the additive 2-out-of-2 secret sharing scheme discussed earlier. Let G be an arbitrary finite abelian group. Suppose $s \in G$ is the secret. Now get a uniformly random element $u \in G$. Subsequently, take the following $\mathbb{Z}$-linear combinations to determine shares s_0, s_1: $s_0 := u \in G$, and $s_1 := s + u \in G$. To reconstruct s from s_0, s_1, take the following $\mathbb{Z}$-linear combination: $s = s_1 - s_0$. Neither of s_0 and s_1 alone gives any information about s, while s can be reconstructed given both. So, there is 1-privacy and 2-reconstruction. This is a blackbox secret sharing scheme since the only property of G used is that it is a finite abelian group. This extends trivially to $(n-1, n)$-threshold blackbox secret sharing $(n \geq 2)$, where there is $(n-1)$-privacy and n-reconstruction.

Let t, n be arbitrary integers with $1 \leq t < n - 1$. The question is: *does there exist a (t, n)-threshold blackbox secret sharing scheme?* We give an affirmative answer. However, the equivalent of an ideal scheme does not exist: universality, i.e., a single instance works for any finite abelian group as discussed above, comes at the price of a share being necessarily "larger" than the secret. In other words, the *expansion factor*, i.e., the total number of group elements assigned to the n players in a secret-sharing divided by n, is *greater* than 1.

This can be argued as follows. Observe that a (t, n)-threshold blackbox secret sharing scheme in particular implies a generalized (t, n)-threshold secret sharing scheme over $\mathbb{F}_2$ in the sense of Definition 11.81. A result from [120] on the complexity of monotone span programs for threshold functions implies that in such secret sharing schemes, a full share vector has length $\Omega(n \log n)$ if $1 \leq t < n - 1$. For details, please refer to [72, 73]. This can also be verified using Corollary 11.106, taking into account Remark 11.14, and setting $g = 1$.

On the constructive side, as in the case of general linear secret sharing over finite fields, a straightforward adaptation of the approach from [118] gives a blackbox secret sharing scheme for each (t, n)-threshold access structure, where t is the privacy threshold and n is the number of players. However, unless t is close to 1 or close to n, the share vector of G-elements a player gets expands to very large length, namely *exponential* length in n if $t \approx n/2$. This can be brought down to *polynomial* (in n) expansion for any majority threshold access structure (i.e., where $t + 1 > n/2$) by using an approach outlined in Section 11.11.2, which combines [20] with [174].

However, there are vastly superior solutions. We first present a solution [86] which achieves expansion as small as *linear* (in n), which already substantially improves the approach referred to above. This solution is based on algebraic number theory. Then we present two solutions [72, 73] that achieve the minimal logarithmic expansion. These are also based on algebraic number theory, but

[32] Also known as *repeated-squaring algorithm*. See e.g. [164].

work around what seems to be a difficult obstacle in [86] to achieving smaller than linear expansion by that method.

Before doing so, we note that blackbox secret sharing was originally motivated by threshold public key cryptography [84, 85], i.e., crypto-systems where the decryption capability is securely distributed among a number of servers. In the case of the RSA crypto-system, the problem arises that the order of the group in which the decryption (or signing-) key lives has to remain secret from each quorum of t servers. Blackbox secret sharing provides a way to secret-share the RSA secret key in a way that the decryption (or signature-) process can be handled by an efficient secure computation among the servers. In the case of RSA better solutions were later found that avoid blackbox secret sharing altogether. However, for certain other relevant threshold cryptography scenarios, blackbox secret sharing is required [81, 82].

A General $\mathbb{Z}$-Module Theoretical Definition

We give a formal definition in terms of matrices (a reader who so prefers can move on to the coordinate-free definition we present afterwards). Let t, n be integers with $1 \leq t < n$. A (t, n)-blackbox secret sharing scheme is captured by a sequence of integer matrices $M, M_0, M_1, \ldots, M_n$ as follows.

There are positive integers $e, d_1, \ldots, d_n$ such that the matrix M has $e \geq 1$ columns and $d := d_1 + \cdots + d_n + 1$ rows. The matrix M_0 corresponds to the submatrix of M consisting of its first row, which is the vector $\mathbf{u}_1^T := (1, 0, \ldots, 0) \in \mathbb{Z}^e$. The matrix M_1 corresponds to the submatrix of M consisting of the next d_1 rows of M, the matrix M_2 corresponds to the submatrix of M consisting of the d_2 rows directly thereafter, etc. By extension, for a nonempty subset $B \subset \{1, \ldots, n\}$, let M_B be the submatrix M restricted to those blocks M_i with $i \in B$. Also, define $d_B = \sum_{i \in B} d_i$.

Definition 11.117 (Blackbox Secret Sharing) *Let t, n be integers with $1 \leq t < n$. The sequence of integer matrices*

$$(M_0, M_1, \ldots, M_n)$$

*introduced above is a (t, n)-*threshold blackbox secret sharing scheme *if, for each $B \subset \{1, \ldots, n\}$, the following holds.*

1. *If $|B| = t + 1$, then there is a vector $\mathbf{r} \in \mathbb{Z}^{d_B}$, depending on B, such that*

$$M_B^T \mathbf{r} = \mathbf{u}_1 \in \mathbb{Z}^e$$

2. *If $|B| = t$, then there exists a vector $\mathbf{k} \in \mathbb{Z}^e$ such that*

$$M_0 \mathbf{k} = \mathbf{u}_1^T \mathbf{k} = 1 \in \mathbb{Z} \quad \text{and} \quad M_B \mathbf{k} = \mathbf{0} \in \mathbb{Z}^{d_B}$$

The expansion factor of the scheme is the fraction

$$\frac{\sum_{i=1}^n d_i}{n} \in \mathbb{Q}.$$

With the $(n-1, n)$-threshold blackbox secret sharing scheme discussed before in mind, we now explain how schemes satisfying this definition can be similarly used, but now in the general case of (t, n)-threshold blackbox secret sharing.

For each finite abelian group G, the matrix M induces a morphism of finite $\mathbb{Z}$-modules

$$M : G^e \longrightarrow G \times G^{d_1} \times \cdots \times G^{d_n},$$

$$\mathbf{g} \mapsto (\mathbf{u}_1^T \mathbf{g}, M_1 \mathbf{g}, \ldots, M_n \mathbf{g}).$$

The first coordinate (the G-coordinate) will be viewed as the "secret" and the remaining n coordinates as the "shares." By Lemma 11.59, the map M is a regular map onto its image $M(G^e)$, i.e., each element in the image has the same (finite) number of preimages. Therefore, selecting a uniformly random element $\gamma \in M(G^e)$ is the same as selecting a uniformly random element $\mathbf{g} \in G^e$ and setting $\gamma := M\mathbf{g}$. Therefore, the uniform distribution on G is attained by the secret if $\gamma \in M(G^e)$ is selected uniformly at random: $M_0 \mathbf{g}$ just "picks" the first coordinate $\mathbf{u}_1^T \mathbf{g}$ of $\mathbf{g}$. In particular, the secret can take any value in G. If $|B| = t+1$, then

$$\mathbf{r}^T (M_B \mathbf{g}) = (\mathbf{r}^T M_B) \mathbf{g} = \mathbf{u}_1^T \mathbf{g} = M_0 \mathbf{g}.$$

Hence, there is $(t+1)$-reconstruction. If $|B| = t$, then for each $s \in G$ there is an element in $M(G^e)$ such that the secret equals s and such that, for each $i \in B$, the share for i is $\mathbf{0} \in G^{d_i}$. This is easy to see as follows. For a vector $\mathbf{x} = (x_1, \ldots, x_e)^T \in \mathbb{Z}^e$ and an element $g \in G$, let $\mathbf{x} * g$ denote the vector $(x_1 g, \ldots, x_e g) \in G^e$. Then

$$M(\mathbf{x} * g) = (M\mathbf{x}) * g.$$

By the properties of the vector $\mathbf{k}$, the secret of $M(\mathbf{k} * s) \in M(G^e)$ equals s and, for each $i \in B$, the share for i equals $\mathbf{0} \in G^{d_i}$. Combining this observation with the fact that the map induced by M is a regular map onto its image $M(G^e)$, it follows using Lemma 11.46 that there is t-privacy. Note that the distribution on t shares does not depend on the secret.

Note that the integer matrices and vectors involved in the definition of blackbox secret sharing schemes are the same for all finite abelian groups G. Please refer to [72] for a discussion arguing that the definition given above is "minimal" in a precise and meaningful way.

For completeness, we also give an equivalent, *coordinate-free* definition, as follows.

Definition 11.118 (Alternative, Equivalent Definition) *Let t, n be integers with $1 \leq t < n$. A (t, n)-*threshold blackbox secret sharing scheme *consists of a free $\mathbb{Z}$-module $\mathcal{M}$ and submodules*

$$\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n \subset \mathcal{M}.$$

For each nonempty subset $B \subset \{1, \ldots, n\}$, define

$$\mathcal{M}_B = \sum_{i \in B} \mathcal{M}_i,$$

i.e., $\mathcal{M}_B \subset \mathcal{M}$ is the submodule spanned by the $\mathcal{M}_i$ with $i \in B$. For each $B \subset \{1, \ldots, n\}$, the following is required.

1. $\mathrm{rank}(\mathcal{M}_0) = 1$.

2. If $|B| = t + 1$, then

$$\mathcal{M}_0 \subset \mathcal{M}_B.$$

3. If $|B| = t$, then there is a $\mathbb{Z}$-linear form

$$\phi^B : \mathcal{M} \longrightarrow \mathbb{Z},$$

depending on B, such that

$$\phi^B(\mathcal{M}_B) = 0 \textit{ and } \phi^B(\mathcal{M}_0) = \mathbb{Z}.$$

The expansion factor *of the scheme is the ratio*

$$\frac{\sum_{i=1}^{n} \mathrm{rank}(\mathcal{M}_i)}{n} \in \mathbb{Q}.$$

Lemma 11.119 *Definitions 11.118 and 11.117 are equivalent.*

Proof It is straightforward that a scheme in the sense of Definition 11.117 implies a scheme in the sense of Definition 11.118, with the same parameters. The other direction follows from an application of Theorem 10.1. Since $\mathrm{rank}(\mathcal{M}_0) = 1$ and since there is a $\mathbb{Z}$-linear form on $\mathcal{M}$ whose image of $\mathcal{M}_0$ equals $\mathbb{Z}$, namely the forms ϕ^B with $|B| = t$, there is $z \in \mathcal{M}_0$ such that z is a $\mathbb{Z}$-basis of $\mathcal{M}_0$ and such that there is a $\mathbb{Z}$-basis of $\mathcal{M}$ that includes z. Without loss of generality, we may assume that $\phi^B(z) = 1$ for each B with $|B| = t$ (by updating ϕ^B with a sign, where necessary). This means that when embedding $\mathcal{M}$ into $\mathbb{Z}^e$ for some large enough integer e, it is possible to choose $\mathbf{z} = (1, 0, \ldots, 0) \in \mathbb{Z}^e$ without loss of generality. To revert to the matrix based definition, choose a basis (or generating set) for each submodule $\mathcal{M}_i$. □

In comparison with the definition of general linear secret sharing over a finite field, one of the important differences is the technical handle for proving t-privacy. Whereas in the former case it is sufficient to require that if $|B| = t$, then $z \notin \mathcal{M}_B$, this is not sufficient in the latter: since we are working over $\mathbb{Z}$ here, all that can be concluded with certainty from $z \notin \mathcal{M}_B$ is that there is a $\mathbb{Z}$-linear form $\phi^B : \mathcal{M} \longrightarrow \mathbb{Z}$, depending on B, such that $\phi^B(\mathcal{M}_B) = 0$ and $\phi^B(z) = \lambda \neq 0$, for some integer λ. This is not sufficient for blackbox secret sharing. Namely, for fixed B with $|B| = t$, consider the ideal $I \subset \mathbb{Z}$ generated by all such integers λ

and write $I = (a)$ for some integer a. If $I \subsetneq \mathbb{Z}$ and if p is a prime number dividing a, then this means the set B can reconstruct the secret in the case $G = \mathbb{Z}/p\mathbb{Z}$. For a full technical discussion, please refer to [72].

11.13.3 Constructions Based on Algebraic Number Theory

We present the constructions from [86] and [72] and for each of those we address the central points in its analysis. [33] Suppose A is a nontrivial commutative ring which is free and of finite rank as a $\mathbb{Z}$-module. [34] Its $\mathbb{Z}$-rank is denoted $k(A)$. Let G be an arbitrary finite abelian group. Set

$$M = G \otimes_{\mathbb{Z}} A.$$

We view M as an A-module, by base extension; see Section 10.9.2.

Let t, n be integers with $0 \leq t < n$. Suppose

$$n + 1 \leq \lambda(A).$$

Fix an admissible set $T \subset A$ with $|T| = n + 1$. Let $\alpha_0, \ldots, \alpha_n$ denote the distinct elements of T. Consider the A-module morphism

$$\mathcal{E}' : M \otimes_A A[X]_{\leq t} \longrightarrow \bigoplus_{i=0}^{n} M$$

$$\mathbf{f} \mapsto (\mathbf{f}(\alpha_0), \ldots, \mathbf{f}(\alpha_n)),$$

see Definition 11.112.

On account of Lemma 10.16, it holds that

$$|M| = |G|^{k(A)},$$

since A is a free $\mathbb{Z}$-module of rank $k(A)$, and that

$$|M \otimes_A A[X]_{\leq t}| = |M|^{t+1} = |G|^{k(A)\cdot(t+1)},$$

since $A[X]_{\leq t}$ is a free A-module of rank $t + 1$.

For each $B \subset \{0, 1, \ldots, n\}$ with $|B| = t + 1$ and for each vector

$$(z_i)_{i \in B} \in \bigoplus_{i \in B} M,$$

there is a unique

$$\mathbf{f} = \sum_{i=0}^{t} f_i \otimes X^i \ \in \ M \otimes_A A[X]_{\leq t},$$

[33] Our exposition here differs significantly from the respective expositions in [86] and [72], though.

[34] Note that A contains $\mathbb{Z}$ as a subring in a unique way. Namely, 0 is mapped to 0_A and 1 is mapped to 1_A. Since A is free, this is an injective ring morphism. In particular, $k(A) \geq 1$.

with $f_i \in M$ $(i = 0, \ldots, t)$, such that, for each $i \in B$,

$$\mathbf{f}(\alpha_i) = z_i \in M.$$

This follows from Theorem 11.113, instantiated with the admissible set $T_B = \{\alpha_i\}_{i \in B}$. Note that this unique $\mathbf{f}$ can be computed from the vector $(z_i)_{i \in B}$ by applying the inverse of this isomorphism, whose explicit description is given at the very end of the proof of Theorem 11.113. In conclusion, the image of the map $\mathcal{E}'$ is a $(t+1, n+1, |G|^{k(A)})$-interpolation code over the finite alphabet M. This gives an ideal (t, n)-threshold secret sharing scheme defined over M by Theorem 11.93.

We now discuss how to base a (t, n)-blackbox secret sharing scheme upon these observations. Replacing α_i by $\alpha_i - \alpha_0$ for $i = 0, \ldots, n$, we may, without loss of generality, assume $\alpha_0 = 0$. Fix a basis $e_1, \ldots, e_k$ of A as a free $\mathbb{Z}$-module, where $k = k(A)$. Recall there is a one-to-one correspondence between $M = G \otimes_{\mathbb{Z}} A$ and G^k in that the elements of M have a unique representation as $\sum_{\ell=1}^{k} g_\ell \otimes e_\ell$ with $g_1, \ldots, g_k \in G$. Thus, the expansion factor is $k(A)$, the rank of A as a free $\mathbb{Z}$-module.

Since $\alpha_0 = 0$, it is immediately clear that a uniformly random $\mathbf{f} \in M \otimes_A A[X]_{\leq t}$ leads to $\mathbf{f}(0)$ having uniform distribution on M. Decomposing $\mathbf{f}(0) \in M$ as

$$\mathbf{f}(0) = f_0 = \sum_{\ell=1}^{k} g_\ell \otimes e_\ell,$$

the value

$$s \otimes e_1 \in M \text{ with } s := g_1 \in G$$

is declared to be the secret. Note that this is essentially the same as saying that the secret is the "first coordinate" of the vector in G^k that represents $\mathbf{f}(0) \in M$.

As to the computations, all inputs, intermediary results and outputs are represented as elements in G^k, keeping the one-to-one correspondence with M in mind. Choosing a uniformly random element in M then just comes down to blackbox sampling k independent uniformly random elements from G. Additions of elements in M amounts to blackbox addition of the respective vectors in G^k representing them.

By inspection of the proof of Theorem 11.113, particularly, the evaluation map and its inverse, it follows that the computation of the shares and the reconstruction of the secret is by A-scalar operations completely oblivious of the specific finite abelian group involved. By the discussion in Section 10.9.3, it follows that A-scalar multiplications are composed of $\mathbb{Z}$-scalar multiplications. Hence, this gives a blackbox secret sharing scheme as required. [35] These considerations lead to the following theorem.

[35] It is not hard to show that the conditions from Definition 11.117 are satisfied; but we omit the details.

THEOREM 11.120 *Suppose A is a nontrivial commutative ring such that as a $\mathbb{Z}$-module, A is free and of finite rank $k(A)$. Suppose $\lambda(A) \geq 4$. Let t, n be integers such that $1 \leq t < n-1 \leq \lambda(A) - 2$. Then there exists a (t,n)-threshold blackbox secret sharing scheme with expansion factor $k(A)$.*

REMARK 11.17 Recall that in the case $t = n-1$, there is blackbox secret sharing with expansion factor 1, i.e., no expansion. [36]

The solution from [86] is retrieved by selecting a prime number p with $p > n$ and by setting A as the ring of integers of the p-th cyclotomic number field. This ring A satisfies $\lambda(A) = p$, as shown in Section 11.13.1. By *Bertrand's Postulate* [37] from number theory, for all integers $n \geq 1$, there is a prime number p with $n < p \leq 2n$. This ensures that expansion $O(n)$ is achievable by this method.

THEOREM 11.121 *[86] Let t, n be integers such that $1 \leq t < n-1$. Then there exists a (t,n)-threshold blackbox secret sharing scheme with expansion factor $O(n)$.*

It appears to be a difficult problem whether or not there exist nontrivial commutative rings A that are free and of finite rank as $\mathbb{Z}$-modules and that achieve $\lambda(A)$ *exponential* in $k(A)$. [38] This is an obstacle to improving blackbox secret sharing via the method of [86].

This obstacle is circumvented in [72], exploiting the observation that blackbox secret sharing may as well be based on *admissible pairs*. After a suitable choice of rings A, this leads to $O(\log n)$ expansion, which is minimal.

DEFINITION 11.122 *Let R be a nontrivial commutative ring. Let $T \subset R$ be a finite set with $|T| > 1$. Then*

$$\Delta(T) = (-1)^{\frac{1}{2}|T|(|T|-1)} \cdot \prod_{\substack{\alpha,\alpha' \in T: \\ \alpha \neq \alpha'}} (\alpha - \alpha').$$

Note that $\Delta(T)$ is the *square* of a Vandermonde determinant defined by the elements in the set T.

DEFINITION 11.123 (ADMISSIBLE PAIRS) *Let R be a nontrivial commutative ring. Let $T, T' \subset R$ be sets. Then $\{T, T'\}$ is an* admissible pair *(in R) if*

1. $1 < |T| = |T'| < \infty$.
2. The ideals $(\Delta(T))$ and $(\Delta(T'))$ are co-prime in R.

If $|T| = |T'| = 1$, then $\{T, T'\}$ is by default *an admissible pair in R. The* cardinality *$c(T, T')$ of an admissible pair $\{T, T'\}$ is the integer $(|T|+|T'|)/2$ $(= |T| = |T'|)$.*

[36] Of course, the same holds for the pathetic case $t = 0$.
[37] See for instance [108].
[38] It appears that, for orders in number fields, the best known result is *linear* in the rank.

Clearly, $c(T, T') = |T| = |T'|$. We may assume, without loss of generality, that

$$0 \in T \text{ and } 0 \in T'.$$

Namely, select $\beta \in T$ and replace T by the set consisting of all elements $\alpha - \beta \in R$ with $\alpha \in T$ (similarly for T'). Note that the cardinality has not changed in the process.

Definition 11.124 *Let R be a nontrivial commutative ring. Then $\widehat{\lambda}(R)$ is the supremum of $c(T, T')$, where $\{T, T'\}$ ranges over all admissible pairs in R.*

Note that $\widehat{\lambda}(R) \geq 2$.

We now explain the construction from [72]. As before, A is a nontrivial commutative ring such that as a $\mathbb{Z}$-module, A is free and of finite rank $k(A)$. Select a $\mathbb{Z}$-basis $e_1, \ldots, e_k$ of A, where $k = k(A)$. Let G be an arbitrary finite abelian group and define

$$M = G \otimes_{\mathbb{Z}} A,$$

where M is viewed again as an A-module. Suppose t, n are integers with $0 \leq t < n$. For the moment, let $T \subset A$ be an arbitrary set such that $|T| = n + 1$ and such that $0 \in T$. Let $\alpha_0, \alpha_1, \ldots, \alpha_n$ denote the distinct elements of T, where $\alpha_0 = 0$. Define

$$\delta = \prod_{0 \leq j < i \leq n} (\alpha_i - \alpha_j) \ \in A.$$

Note that

$$\delta^2 = \Delta(T).$$

Slightly modify the above "polynomial evaluation" as

$$\mathbf{f}^{(\delta)}(x) := f_0 \cdot \delta + \sum_{i=1}^{t} f_i \cdot x^i \ \in M$$

for all $x \in A$. Note that the difference is in the "lowest order coefficient."

Define the A-module morphism

$$\mathcal{E}'' : M \otimes_A A[X]_{\leq t} \longrightarrow \bigoplus_{i=0}^{n} M$$

$$\mathbf{f} \mapsto (\mathbf{f}^{(\delta)}(\alpha_0), \ldots, \mathbf{f}^{(\delta)}(\alpha_n)).$$

Consider the (t, n)-threshold blackbox scheme discussed above but with T as selected here, taking into account these slight modifications and the fact that T is not promised to be admissible. Decomposing $f_0 \in M$ as

$$f_0 = \sum_{\ell=1}^{k} g_\ell \otimes e_\ell,$$

we declare the value

$$s \otimes e_1 \in M$$

with $s := g_1 \in G$ to be the secret. This is another modification, as previously the secret was declared from the decomposition of the evaluation at 0.

Using similar reasoning as before, this scheme is "blackbox" and has t-privacy. However, only a *weak form* of $(t+1)$-reconstruction holds. Namely, there is $(t+1)$-reconstruction only of a δ^2-multiple of the secret, i.e., the element $(s \otimes \delta^2 e_1)$, instead of $s \otimes e_1$. [39] If δ is not a unit in A, depending on the order of the group G, this may cause an irreversible loss of information about the secret s. [40] Clearly, this is in violation of the "blackbox principle."

To remedy this, suppose

$$\widehat{\lambda}(A) \geq n+1.$$

Let $\{T, T'\}$ be an admissible pair in A of cardinality $n+1$. Without loss of generality, assume that $0 \in T$ and $0 \in T'$. Since the ideals $(\Delta(T)), (\Delta(T')) \subset A$ are co-prime, there are $\rho, \rho' \in A$ such that

$$\rho \cdot \Delta(T) + \rho' \cdot \Delta(T') = 1.$$

Now consider two instances of the "weak scheme" in parallel: one instance based on T and another instance based on T', *independently* of one another under the constraint that the secret in one equals that in the other. Then the joint scheme is "blackbox" and has t-privacy. [41] This time, however, there is $(t+1)$-reconstruction; the secret $s \otimes e_1$ is recovered as

$$\rho \cdot (s \otimes \Delta(T) \cdot e_1) + \rho' \cdot (s \otimes \Delta(T') \cdot e_1) = (s \otimes (\rho \cdot \Delta(T) + \rho' \cdot \Delta(T'))e_1) = s \otimes e_1.$$

These considerations lead to the following theorem.

Theorem 11.125 *[72] Suppose A is a nontrivial commutative ring such that as a $\mathbb{Z}$-module, A is free and of finite rank $k(A)$. Suppose $\widehat{\lambda}(A) \geq 4$. Let t, n be integers such that $1 \leq t < n-1 \leq \widehat{\lambda}(A) - 2$. Then there exists a (t,n)-threshold blackbox secret sharing scheme with expansion factor at most $2 \cdot k(A)$.*

To circumvent the obstacle in the approach from [86] and to reach minimal $O(\log n)$ expansion, it remains to prove the claim that there exist nontrivial commutative rings A such that, as a $\mathbb{Z}$-module, the ring A is free of finite rank $k(A)$ and such that $\widehat{\lambda}(A)$ is exponential in $k(A)$. This can be shown as follows.

[39] Privacy as well as weak reconstruction can be easily verified by considering Lagrange interpolation over $Q(A)$ and by clearing denominators. See [72] for more details.

[40] This way one also sees that blackbox secret sharing based on "interpolation over the rational integers" fails.

[41] We skip the straightforward formalization of this parallelization as an appropriately defined product of blackbox secret sharing schemes.

Suppose $f(X) \in \mathbb{Z}[X]$ is monic and irreducible. Let $\nu \in \mathbb{C}$ be a root of $f(X)$ and define $K = \mathbb{Q}(\nu)$. Consider the order $\mathbb{Z}[\nu]$ in the number field K. Suppose $f(X)$ has the following two additional properties.

1. $d := \deg f(X) = \lceil \log_2(n+1) \rceil$.
2. For each prime number p with $2 \le p \le n$, it holds that the polynomial $\bar{f}(X) \in \mathbb{F}_p[X]$ is irreducible, where $\bar{f}(X)$ is the image of $f(X)$ under reduction modulo p.

Define

$$T = \{0, 1 \ldots, n\} \subset \mathbb{Z}$$

and

$$\delta = \prod_{0 \le j < i \le n} (i - j).$$

Note that

$$\delta^2 = \Delta(T).$$

Then there exists $T' \subset \mathbb{Z}[\nu]$ such that $|T'| = n + 1$ and such that $(\Delta(T))$ and $(\Delta(T'))$ are co-prime ideals in $\mathbb{Z}[\nu]$. Note that the $\mathbb{Z}$-rank of $\mathbb{Z}[\nu]$ equals $\lceil \log_2(n+1) \rceil$. [42]

The implication above can be argued as follows. As the number δ is a rational integer that factorizes over the prime numbers p with $2 \le p \le n$, it is sufficient to show the existence of distinct elements $\alpha'_0, \alpha'_1, \ldots, \alpha'_n \in \mathbb{Z}[\nu]$ such that the ideals (N) and (δ') are co-prime in $\mathbb{Z}[\nu]$, where

$$N = \prod_{\substack{2 \le p \le n, \\ p \text{ prime}}} p$$

and where

$$\delta' = \prod_{0 \le j < i \le n} (\alpha'_i - \alpha'_j) \in \mathbb{Z}[\nu].$$

Note that

$$(\delta')^2 = \Delta(T').$$

We now show the existence of such a set T'. By the assumptions on $f(X)$, it holds that, for each prime number p with $2 \le p \le n$,

$$\mathbb{Z}[\nu]/p\mathbb{Z}[\nu] \simeq \mathbb{F}_{p^d}$$

and that

$$p^d \ge n + 1.$$

[42] It is possible to slightly weaken the conditions on $f(X)$ while still guaranteeing the same conclusion. But the conditions stated here simplify the ensuing arguments somewhat.

Therefore,

$$\mathbb{Z}[\nu]/N\mathbb{Z}[\nu] \simeq \prod_{\substack{2\leq p\leq n,\\ p \text{ prime}}} \mathbb{F}_{p^d},$$

which follows by the Chinese Remainder Theorem, and, hence,

$$\lambda(\mathbb{Z}[\nu]/N\mathbb{Z}[\nu]) \geq n+1.$$

The latter observation means that there exists $T' = \{\alpha'_0, \alpha'_1, \ldots, \alpha'_n\} \subset \mathbb{Z}[\nu]$ with $|T'| = n+1$ such that (the class of) δ' is a unit in $\mathbb{Z}[\nu]/N\mathbb{Z}[\nu]$, i.e., (δ') and (N) are co-prime ideals in $\mathbb{Z}[\nu]$. It follows at once that $(\Delta(T))$ and $(\Delta(T'))$ are co-prime ideals in $\mathbb{Z}[\nu]$.

All that remains to be shown is the existence of polynomials $f(X)$ satisfying the hypotheses. For each prime number p with $2 \leq p \leq n$ choose an irreducible monic polynomial $f^{(p)}(X) \in \mathbb{F}_p[X]$ of degree $d = \lceil \log_2(n+1) \rceil$. Once again using the Chinese Remainder Theorem, it is easy to construct an integer monic polynomial $f(X)$ of degree d such that modulo each prime number p with $2 \leq p \leq n$ it equals $f^{(p)}(X)$. Since such a polynomial $f(X)$ is (automatically) irreducible in $\mathbb{Z}[X]$, each of the hypotheses is satisfied. Note that the expansion factor of the resulting scheme equals $d = O(\log n)$, which is minimal.

Remark 11.18 If the constructions based on algebraic number theory from [86, 72] are represented by integer matrices in the way indicated above, the integers occurring in these matrices have bit-length polynomial in n.

The constant in the $O(\log n)$ expansion factor has been improved in [73] by using a more refined approach. It is based on sets T defining a Vandermonde determinant that is *primitive*, i.e., its only rational integer divisors are $-1, 1$. The existence of an order A in a number field of degree $k(A)$ such that A admits a set T with a primitive Vandermonde-determinant and such that $|T|$ is exponential in $k(A)$ has been shown in [73] using a combination of arguments from algebraic number theory and algebraic geometry.

Theorem 11.126 *([72, 73]) There exists a positive real constant c such that the following holds. Let t, n be integers with $1 \leq t < n-1$. Then there exists a (t,n)-threshold blackbox secret sharing scheme with expansion factor at most $c + \log_2 n$.*

This is tight up to an additive constant.

12

Codices

Contents

12.1 Introduction

We introduce the notion of an *arithmetic codex*, or *codex* for short [63, 48]. Codices encompass several well-established notions from cryptography (various types of *arithmetic* secret sharing schemes, which all enjoy additive as well as multiplicative properties) and from algebraic complexity (bilinear complexity of multiplication in algebras) in a single mathematical framework. Arithmetic secret sharing schemes have important applications to secure multiparty computation and even to *two*-party cryptography. One such application, known as "MPC-in-the-head" can be found in Chapter 8.

Interestingly, several recent applications to two-party cryptography (including the one we present in this book) rely crucially on the existence of certain *asymptotically good* schemes. It is intriguing that their construction requires asymptotically good towers of algebraic function fields over a finite field: no elementary (probabilistic) constructions are known in these cases. Besides introducing the notion, we discuss some of the constructions, as well as some limitations.

Throughout this chapter, let K be a field. Let $p > 0$ be a prime number, let v be a positive integer, and define $q = p^v$. Let $\mathbb{F}_q$ be a finite field with q elements.

12.2 The Codex Definition

Recall that, for our purposes, a K-algebra is a commutative ring such that K is a subring. By definition of a subring, it is the case that the multiplicative unity of a K-algebra is precisely the multiplicative unity of K. It is henceforth denoted by 1. Also recall that a K-algebra is in particular a K-vector space. See Section 10.8.

12.2.1 Powers of a Space

Taking *powers* of a subspace of a K-algebra R plays a fundamental role in the definition and study of codices. Suppose $C \subset R$ is a K-vector subspace of R, with R viewed as a K-vector space. In general, of course, the space C is *not* closed under multiplication. For example, take $R = K[X]$, the polynomial ring over K, and let C be the subspace of polynomials of degree $\leq t$ for some positive integer t.

We define, for each positive integer d, the "closure" of C under "d-multiplication" as the subspace $C^{*d} \subset R$ generated by the elements of R that can be written as a product of d elements selected from C. Formally:

Definition 12.1 (Set of d-Products) *Let R be a K-algebra. Let $C \subset R$ be* *a K-vector subspace. Let $d \geq 0$ be an integer. The set $m_d(C) \subset R$ consists of all $z \in R$ such that*

$$z = \prod_{i=1}^{d} x^{(i)}$$

for some $(x^{(1)}, \ldots, x^{(d)}) \in C^d$.

By the convention that the empty product equals 1, it follows that $m_0(C) = \{1\}$.

Definition 12.2 (Powers of a Space) *Let R be a K-algebra. Let $C \subset R$ be a K-vector subspace. Let $d \geq 0$ be an integer. The space C^{*d} is the K-vector subspace of R spanned by the set $m_d(C)$.*

Concretely, the space C^{*d} consists of all finite sums $(\sum \lambda x) \in R$ with $\lambda \in K$ and $x \in m_d(C)$. In particular, $C^{*0} = K$.

Remark 12.1 If $1 \in C$, then

$$K \subset C \subset \ldots \subset C^{*d}.$$

If $K = \mathbb{F}_q$ and the *Frobenius map* (i.e., the q-th power map) fixes R (i.e., $x^q = x$ for all $x \in R$), then

$$C \subset C^{*q}.$$

We view K^n as a K-algebra in the natural way, i.e., as an n-fold direct product of the K-algebra K. Thus, ring-multiplication is component-wise. Moreover, K is embedded "diagonally", i.e., $\lambda \in K$ corresponds to $(\lambda, \ldots, \lambda) \in K^n$. In particular, $\mathbf{1} := (1, \ldots, 1)$ is the multiplicative unity. [1] The space K^n is indexed by the set $\{1, \ldots, n\}$. If $\mathbf{x} \in K^n$, then we write $\mathbf{x} = (x_i)_{i=1}^n$ for its coordinate vector. See the conventions in Section 11.2.

For each $\mathbf{x}, \mathbf{y} \in K^n$, their component-wise product in K^n is denoted by $\mathbf{x} * \mathbf{y}$. Thus

$$\mathbf{x} * \mathbf{y} = (x_1 y_1, \ldots, x_n y_n) \in K^n$$

for all $\mathbf{x}, \mathbf{y} \in K^n$. If $\mathbf{x} \in K^n$ and m is a nonnegative integer, then $\mathbf{x}^m$ denotes $\mathbf{x} * \cdots * \mathbf{x} \in K^n$ (m times). The empty product ($m = 0$) yields $\mathbf{1}$ by default. Thus,

$$\mathbf{x}^m = (x_1^m, \ldots, x_n^m)$$

for all $\mathbf{x}, \mathbf{y} \in K^n$ and for all integers $m \geq 0$. The standard inner product on K^n is denoted by $\langle \cdot, \cdot \rangle$. Thus,

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} x_i y_i \ \in K$$

for all $\mathbf{x}, \mathbf{y} \in K^n$.

[1] So, for example, if $K = \mathbb{F}_q$ and $R = \mathbb{F}_q^n$, then Remark 12.1 means that $C \subset C^{*q}$ always.

Let A be a K-algebra and let n be a positive integer. Let t, d, r be integers with $d \geq 1$ and $0 \leq t < r \leq n$. Informally, an (n, t, d, r)-*codex for* A *over* K consists of a K-vector subspace

$$C \subset K^n$$

and a K-vector space morphism

$$\psi : C \longrightarrow A$$

such that the following holds.

1. Each element $a \in A$ is "presented" in the sense that $\psi(\mathbf{x}) = a$ for some $\mathbf{x} \in C$, i.e., ψ is surjective. In particular, this means that

$$\dim_K A < \infty. \text{ [2]}$$

2. The product $\mathbf{z} \in K^n$ of d C-elements uniquely determines the product $a \in A$ of the d respective A-elements presented by them. Moreover, any selection of r coordinates of $\mathbf{z}$ suffices to determine this product a.

   In fact, for each such selection of r coordinates, there exists a K-*vector space morphism* from K^r to A such that, when applied to the selected r coordinates of $\mathbf{z}$, the element a is obtained.
3. If $t > 0$, any t coordinates of a generic ("random") C-element are jointly independent from ("give no information about") the A-element that this C-element presents.

Note that if $K = \mathbb{F}_q$, then the case $d = 1$ above essentially describes a linear secret sharing scheme for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ with t-privacy and r-reconstruction, where $k := \dim_K A$. Not surprisingly, we will be more interested in the case $d > 1$ here.

REMARK 12.2 Recall that if $\dim_K A < \infty$ and 0 is the only nilpotent in A, then A is isomorphic (as a K-algebra) to a finite direct product of finite-degree extension fields of K. Moreover, if the discriminant of A over K is nonzero, then, additionally, each of these extensions is separable over K. See Theorem 10.11. An example with $\dim_K A < \infty$ and with nontrivial nilpotents is the K-algebra $K[X]/(X^2)$, which has $\overline{X}^2 = 0$. It is easy to generate other elementary (multivariate) examples along these lines.

We shall now develop the formal definition of a codex. [3] For notations concerning codes and projection functions, please refer to Section 11.2. By default, we will assume that K^n is indexed by the set $\mathcal{I} = \{1, \ldots, n\}$.

[2] As K-vector spaces, $A \simeq C/\ker\psi$ since ψ is surjective. Therefore, $\dim_K A \leq n$. Note that $\dim_K A \geq 1$, since the field K is contained in A as a subring.

[3] Reading "multiplication" into the symbol "x", the term *codex* signals a code with multiplicative properties.

12.2.2 Multiplicativity

Definition 12.3 ((d, r)-multiplicativity) *Let A be a K-algebra. Let n, d, r be integers with $d \geq 1$ and $1 \leq r \leq n$. Let $C \subset K^n$ be a K-vector subspace and let*

$$\psi : C \longrightarrow A$$

be a K-vector space morphism. Then (C, ψ) has (d, r)-multiplicativity if there is a unique K-vector space morphism

$$\overline{\psi} : C^{*d} \longrightarrow A$$

such that the following conditions are satisfied.

1. *$\overline{\psi}$ satisfies the* multiplicative relation

$$\overline{\psi}(\mathbf{x}_1 * \cdots * \mathbf{x}_d) = \prod_{j=1}^{d} \psi(\mathbf{x}_j),$$

for all $\mathbf{x}_1, \ldots, \mathbf{x}_d \in C$.

2. *$\overline{\psi}$ is* r-wise determined, *i.e.,*

$$C^{*d} \quad \cap \bigcup_{\substack{B \subset \{1,\ldots,n\}: \\ |B|=r}} \ker \pi_B \quad \subset \quad \ker \overline{\psi},$$

Note that, at this point, the morphism ψ is not necessarily surjective. In particular, the trivial morphism is not excluded from the definition here. Also, strictly speaking, it is not necessary to state the uniqueness of $\overline{\psi}$ as a separate condition; since C^{*d} is generated as a K-vector space by $m_d(C)$ and since the multiplicative relation holds, mere existence means that the K-vector space morphism $\overline{\psi}$ is determined by its action on $m_d(C)$, which is, in turn, determined by ψ.

By inspection, the condition of being r-wise determined is equivalent to the condition that $\overline{\psi}(\mathbf{z}) = \overline{\psi}(\mathbf{z}')$ for all $\mathbf{z}, \mathbf{z}' \in C^{*d}$ with $\pi_B(\mathbf{z}) = \pi_B(\mathbf{z}')$ for some $B \subset \{1, \ldots, n\}$ with $|B| = r$. This motivates the terminology, as this means that "$\overline{\psi}(\mathbf{z})$ is uniquely determined by any r coordinates of $\mathbf{z}$, for each $\mathbf{z} \in C^{*d}$." In alternative language, the restriction of C^{*d} to those elements whose Hamming-weight is at most $n - r$, is contained in the kernel of $\overline{\psi}$.

In the case $d = 1$, the map ψ is r-wise determined by definition, since ψ and $\overline{\psi}$ coincide in this case. This is also true in the uninteresting case $d \geq 1$ and $\psi \equiv 0$. We now give a general condition that guarantees that (d, r)-multiplicativity implies that ψ is r-determined. This condition is trivially satisfied in the cases of our interest later on. See also Remark 12.3.

Definition 12.4 *The set consisting of all $z \in A$ such that $z = 0$ or z is a zero divisor in A is denoted $Z(A)$.*

Lemma 12.5 *The complement $A \setminus Z(A)$ is multiplicatively closed.*

Proof Let $a, a' \in A \setminus Z(A)$. Suppose $b \in A$ is such that $(aa')b = 0$. Since $(aa')b = a(a'b) = 0$, it follows that $a'b = 0$. This implies $b = 0$. Hence, $aa' \in A \setminus Z(A)$. $\square$

Lemma 12.6 *Suppose that at least one of the following conditions is satisfied.*

1. *The only nilpotent in A is 0.*
2. $\operatorname{im} \psi \not\subset Z(A)$.

Then (d, r)-multiplicativity in Definition 12.3 implies that ψ is r-determined as well, i.e.,

$$C \cap \bigcup_{\substack{B \subset \{1,\dots,n\}: \\ |B| = r}} \ker \pi_B \subset \ker \psi.$$

Proof We may assume $d > 1$. Suppose $\mathbf{x} \in C$ is such that $\mathbf{x}_B = \mathbf{0}$ for some $B \subset \{1, \dots, n\}$ with $|B| = r$. We show that $\psi(\mathbf{x}) = 0$. Since $\mathbf{x}^d \in C^{*d}$, $(\mathbf{x}^d)_B = \mathbf{0}$ and $|B| = r$ and, the identity

$$\overline{\psi}(\mathbf{x}^d) = (\psi(\mathbf{x}))^d = 0$$

holds by definition of (d, r)-multiplicativity. Hence, $\psi(\mathbf{x})$ is a nilpotent. If the first condition of the lemma is satisfied, it follows that $\psi(\mathbf{x}) = 0$.

Now assume that the second condition of the lemma is satisfied. Let $\mathbf{y} \in C$ be such that $\psi(\mathbf{y}) \in A \setminus Z(A)$. Define

$$\mathbf{z} = \mathbf{x} * \mathbf{y}^{d-1}.$$

Since $\mathbf{z} \in C^{*d}$, $\mathbf{z}_B = \mathbf{0}$ and $B = r$, the identity

$$0 = \overline{\psi}(\mathbf{z}) = \overline{\psi}(\mathbf{x} * \mathbf{y}^{d-1}) = \psi(\mathbf{x}) \cdot (\psi(\mathbf{y}))^{d-1}$$

holds by definition of (d, r)-multiplicativity. By Lemma 12.5, it holds that

$$(\psi(\mathbf{y}))^{d-1} \in A \setminus Z(A).$$

Hence, $\overline{\psi}(\mathbf{x}) = 0$ as claimed $\square$

Remark 12.3 If A is a product of fields, for instance, then the first condition of the lemma is satisfied. If ψ is surjective (so that 1 has a ψ-inverse), then the second condition of the lemma is satisfied.

Definition 12.7 *Suppose (C, ψ) has (d, r)-multiplicativity. Then (C, ψ) is* unital *if $\mathbf{1} \in C$ and $\psi(\mathbf{1}) = 1$.*

Remark 12.4 Suppose (C, ψ) has (d, r)-multiplicativity and suppose it is unital. Then:

1. $C \subset C^{*d}$.
2. The restriction of $\overline{\psi}$ to C coincides with ψ.

In other words, we may say that $\overline{\psi}$ *extends* the map ψ to C^{*d}. By convention, this extension is denoted by ψ as well. Additionally, the following strengthening of the multiplicative relation holds:

$$\psi(\mathbf{x}_1 * \cdots * \mathbf{x}_{d'}) = \prod_{j=1}^{d'} \psi(\mathbf{x}_j),$$

for all integers d' with $1 \leq d' \leq d$ and for all $\mathbf{x}_1, \ldots, \mathbf{x}_{d'} \in C$.

The first property above follows on account of Remark 12.1. The second property follows from the fact that for all $\mathbf{x} \in C$ it holds that

$$\overline{\psi}(\mathbf{x}) = \overline{\psi}(\mathbf{1}^{d-1} * \mathbf{x}) = \left(\prod_{j=1}^{d-1} \psi(\mathbf{1})\right) \cdot \psi(\mathbf{x}) = \psi(\mathbf{x}).$$

The strengthening of the multiplicative relation stated above follows from (d, r)-reconstruction, as for all integers d' with $1 \leq d' \leq d$ and for all $\mathbf{x}_1, \ldots, \mathbf{x}_{d'} \in C$, it holds that

$$\psi(\mathbf{x}_1 * \cdots * \mathbf{x}_{d'} * \mathbf{1}^{d-d'}) = \left(\prod_{j=1}^{d'} \psi(\mathbf{x}_j)\right) \cdot \psi(\mathbf{1})^{d-d'} = \prod_{j=1}^{d'} \psi(\mathbf{x}_j).$$

12.2.3 Disconnection

Definition 12.8 (t-disconnection) *Let A be a K-algebra. Let n, t be integers with $0 \leq t < n$. Let $C \subset K^n$ be a K-vector subspace and let*

$$\psi : C \longrightarrow A$$

be a K-vector space morphism. If $t = 0$, then (C, ψ) is 0-disconnected *by default. Suppose $t > 0$. Then (C, ψ) is* t-disconnected *if, for each $B \subset \{1, \ldots, n\}$ with $|B| = t$, the projection map*

$$\pi_{\psi,B} : C \longrightarrow \psi(C) \times \pi_B(C)$$

$$\mathbf{x} \mapsto (\psi(\mathbf{x}), \pi_B(\mathbf{x}))$$

is surjective. If, in addition, $\pi_B(C) = K^t$, there is t-disconnection with uniformity.

In the case where K is a finite field (so that C and $\psi(C)$ are finite) there is the following interpretation. Suppose $t > 0$ and let $B \subset \{1, \ldots, n\}$ with $|B| = t$. By Lemma 11.59, the map $\pi_{\psi,B}$ is regular and, if $\mathbf{x}$ is uniformly random on C, then $\pi_{\psi,B}(\mathbf{x})$ is uniformly random on $\psi(C) \times \pi_B(C)$. By elementary probability, this is equivalent to saying that $\psi(\mathbf{x})$ is uniformly random on $\psi(C)$, the vector $\pi_B(\mathbf{x})$ is uniformly random on $\pi_B(C)$, and $\psi(\mathbf{x})$ and $\pi_B(\mathbf{x})$ are *independently distributed*. In particular, if $\psi \not\equiv 0$, then $\psi(\mathbf{x}) \in A$ can be guessed from $\pi_B(\mathbf{x}) \in K^t$ with probability ϵ, where $\epsilon = 1/|\psi(C)| \leq 1/|K|$, but no better. Note that $\epsilon = 1/|A|$ if

ψ is surjective. Uniformity means that, in addition, $\pi_B(\mathbf{x})$ is uniformly random on K^t.

It is sometimes convenient to work with the following equivalent formulation of disconnection.

Lemma 12.9 *Let A be a K-algebra. Let n, t be integers with $1 \leq t < n$. Let $C \subset K^n$ be a K-vector subspace and let $\psi : C \longrightarrow A$ be a K-vector space morphism. Then (C, ψ) is t-disconnected if and only if for each $B \subset \{1, \ldots, n\}$ with $|B| = t$ and for each $a \in \psi(C)$, there is $\mathbf{x} \in C$ with $\psi(\mathbf{x}) = a$ and $\mathbf{x}_B = \mathbf{0} \in K^t$.*

Proof The forward direction follows from the definition of disconnection. In the other direction, we argue as follows. Let $B \subset \{1, \ldots, n\}$ with $|B| = t$. Let $a' \in \psi(C)$ and let $\mathbf{x}' \in C$. Then $a' - \psi(\mathbf{x}') \in \psi(C)$. Let $\mathbf{x}'' \in C$ be such that $\psi(\mathbf{x}'') = a' - \psi(\mathbf{x}')$ and $\mathbf{x}''_B = \mathbf{0} \in K^t$. Define

$$\mathbf{z} = \mathbf{x}' + \mathbf{x}'' \in C$$

Then

$$\psi(\mathbf{z}) = \psi(\mathbf{x}' + \mathbf{x}'') = \psi(\mathbf{x}') + \psi(\mathbf{x}'') = \psi(\mathbf{x}') + (a' - \psi(\mathbf{x}')) = a'$$

and

$$\mathbf{z}_B = \mathbf{x}'_B.$$

This shows that the image of $\pi_{\psi,B}$ is all of $\psi(C) \times \pi_B(C)$. □

12.2.4 Codices

We are now ready to define codices.

Definition 12.10 (Codex) *[63, 48] Let A be a K-algebra with $\dim_K A < \infty$. Let n, t, d, r be integers with $d \geq 1$ and $0 \leq t < r \leq n$. An (n, t, d, r)-*codex *for A over K is a pair (C, ψ) where*

$$C \subset K^n$$

is a K-vector subspace and

$$\psi : C \longrightarrow A$$

is a K-vector space morphism such that:

1. *ψ is surjective.*
2. *(C, ψ) satisfies (d, r)-multiplicativity.*
3. *(C, ψ) satisfies t-disconnection.*

An (n, t, d, r)-codex for A over K has uniformity *if it satisfies the stronger property of t-disconnection with uniformity. It is* unital *if (C, ψ) is unital.*

Definition 12.11 (Arithmetic Secret Sharing) *If K is a finite field, $d \geq 2$ and $t \geq 1$, then the codex in Definition 12.10 is called an* (n, t, d, r)-arithmetic secret sharing scheme *with secret-space A and share-space K.*

These general definitions of codices and arithmetic secret sharing schemes grew out of a series of results [68, 54, 56, 43, 46], starting with the results on linear secret sharing with (strong) multiplication in [68]. The latter were in turn inspired by the way multiplicative properties of Shamir's scheme have been exploited in secure multiparty computation in [16, 52, 96].

As opposed to the notion of "plain secret sharing", which is very suggestive as to how it may actually be used in cryptographic protocols, the notion of arithmetic secret sharing is less intuitive. For instance, the way the properties of these schemes are exploited in secure computation can hardly be guessed straight from their definition. Please refer to Section 12.5 or to [46] for a high-level explanation of two of the main applications of arithmetic secret sharing to secure computation. See also the references in Section 12.8.

Definition 12.12 (Arithmetic Embedding) *In Definition 12.10, if ψ is a K-vector space isomorphism and if $d \geq 2$ then the codex is an* (n, d)-arithmetic embedding *(of A over K).*

Note that $t > 0$ is not possible in case of an arithmetic embedding. If $d = 2$, then the smallest n such that an $(n, 2)$-arithmetic embedding of A over K exists is the *bilinear multiplication complexity* of A over K, a classical notion from algebraic complexity theory (see e.g. [34]). Especially the case where K is a finite field $\mathbb{F}_q$ and A is an extension field $\mathbb{F}_{q^k}$ (for some integer $k > 1$) has been extensively studied during the last four decades.

Note that in the case where K is a finite field and A is a finite-degree extension field of K, the codex notion generalizes this classical notion of bilinear multiplication algorithms in finite fields in two ways if both $t > 0$ and $r < n$. Also note that arithmetic secret sharing schemes with secret-space $\mathbb{F}_q^k$ and share-space $\mathbb{F}_q$ have particularly important cryptographic applications (see the references in [46] and see Section 12.5), whereas bilinear complexity is trivial here. From a cryptographic point of view, our notion encompasses all known relevant variations on arithmetic secret sharing.

Finally, for favorable settings of the parameters, codices support "efficient decoding" of the A-element even if a presentation $\mathbf{x} \in C$ comes with some errors, by a linearization argument that makes generic use of the properties of the codex. An explanation is given in Section 12.5.4.

12.3 Equivalent Definitions

12.3.1 A Definition in Terms of Generalized Codes

We give an equivalent definition of codices in terms of generalized codes in Theorem 12.14. This definition is particularly useful when analyzing certain explicit constructions.

Let A be a K-algebra and let n be a positive integer. Consider the $(n+1)$-fold direct product $A \times K^n$ and index it with the set $\mathcal{I} = \{0, 1 \ldots, n\}$ [4]. Define $\mathcal{I}^* = \{1, \ldots, n\}$. For $B \subset \mathcal{I}^*$ nonempty, define the following. The projection functions $\pi_0 : A \times K^n \longrightarrow A$ and $\pi_B : A \times K^n \longrightarrow K^{|B|}$ are defined in the obvious way. The notation $\pi_{0,B}$ is a shorthand for the projection $\pi_{\{0\}\cup B}$.

If $\widetilde{C} \subset A \times K^n$ is a K-vector subspace, then $C \subset K^n$ denotes the *K-part of* $\widetilde{C}$, i.e., the K-vector subspace consisting of all $\mathbf{x} \in K^n$ such that $(a, \mathbf{x}) \in \widetilde{C}$ for some $a \in A$. Equivalently, $C = \pi_{\mathcal{I}^*}(\widetilde{C})$.

Conversely, if $C \subset K^n$ is a K-vector subspace and $\psi : C \longrightarrow A$ is a K-vector space morphism, then $\widetilde{C} \subset A \times K^n$ is the K-vector subspace consisting of all $(\psi(\mathbf{x}), \mathbf{x}) \in A \times K^n$ with $\mathbf{x} \in C$, *the A-extension of (C, ψ).*

Lemma 12.13 *Let A be a K-algebra. Let n, d, r be integers with $d \geq 1$ and $1 \leq r \leq n$. Then:*

1. *Let $\widetilde{C} \subset A \times K^n$ be a K-vector subspace. Assume that the only nilpotent in A is 0 or that there exists $(u, \mathbf{y}) \in \widetilde{C}$ such that $u \in A \setminus Z(A)$.* [5] *Suppose there is a K-vector subspace $\widetilde{D} \subset A \times K^n$ such that*
$$\widetilde{C}^{*d} \subset \widetilde{D}$$
and such that for all $\widetilde{\mathbf{z}} \in \widetilde{D}$ it holds that
$$\widetilde{z}_0 = 0 \text{ if } \widetilde{\mathbf{z}}_B = \mathbf{0} \text{ for some } B \subset \mathcal{I}^* \text{ with } |B| = r.$$
Then (C, ψ) has (d, r)-multiplicativity where $C \subset K^n$ is the K-part of $\widetilde{C}$ and where the K-vector space morphism
$$\psi : C \longrightarrow A$$
$$\mathbf{x} \mapsto a$$
is such that $a \in A$ is the unique element with $(a, \mathbf{x}) \in \widetilde{C}$.
2. *Conversely, let $C \subset K^n$ be a K-vector subspace and let $\psi : C \longrightarrow A$ be a K-vector space morphism. Suppose (C, ψ) has (d, r)-multiplicativity. Let $\overline{\psi}$ be the associated map from Definition 12.3. Then its A-extension $\widetilde{C} \subset A \times K^n$ satisfies*
$$\widetilde{C}^{*d} = \{(\overline{\psi}(\mathbf{z}), \mathbf{z}) \mid \mathbf{z} \in C^{*d}\} \subset A \times K^n.$$

[4] To avoid confusion, we mean that index 0 refers to the A-coordinate, whereas indices $1, \ldots, n$ refer to the n K-coordinates, as usual.

[5] See Remark 12.3; this condition is naturally satisfied in the cases of our interest here.

Remark 12.5 The reader may as well set $\widetilde{D} = \widetilde{C}^{*d}$ in this lemma and the subsequent Theorem 12.14. However, the actual statement of the lemma reflects the fact that in concrete situations, one typically proves bounds on the parameters of $\widetilde{C}^{*d}$ by finding a (larger) code $\widetilde{D}$ whose relevant parameters can be more easily analyzed. Therefore, it is useful to have the lemma already taking care of this situation.

Proof (of Lemma 12.13)
Let $\mathbf{z} \in K^n$. If there is $a \in A$ such that $(a, \mathbf{z}) \in \widetilde{C}^{*d}$, then this value a is unique. Namely, suppose $(a', \mathbf{z}) \in \widetilde{C}^{*d}$ with $a' \in A$. Then $(a - a', \mathbf{0}) \in \widetilde{C}^{*d}$. By the conditions on $\widetilde{D}$, taking $B = \{1, \ldots, n\}$ and noting that $r \leq n$, it follows that $a = a'$. Therefore, the K-vector space morphism

$$\overline{\psi} : C^{*d} \longrightarrow A$$

$$\mathbf{z} \mapsto a$$

such that $(a, \mathbf{z}) \in \widetilde{C}^{*d}$ is well-defined.

Furthermore, let $\mathbf{x} \in K^n$. If there is $a \in A$ such that $(a, \mathbf{x}) \in \widetilde{C}$, then this value a is unique as well. We may assume $d > 1$. Suppose that $\widetilde{\mathbf{x}} := (b, \mathbf{0}) \in \widetilde{C}$ for some $b \in A$. We show that $b = 0$. Since

$$\widetilde{\mathbf{x}}^d = (b^d, \mathbf{0}) \in \widetilde{C}^{*d},$$

it follows from the conditions on $\widetilde{D}$ that

$$b^d = 0.$$

Hence, b is a nilpotent of A. Therefore, if the only nilpotent in A is 0, it follows that $b = 0$.

If A does have nontrivial nilpotents, let $(u, \mathbf{y})$ be as in the statement of the lemma. Then

$$\widetilde{\mathbf{x}} * (u^{d-1}, \mathbf{y}^{d-1}) = (bu^{d-1}, \mathbf{0}) \in \widetilde{C}^{*d}.$$

By the conditions on $\widetilde{D}$, it follows that

$$bu^{d-1} = 0.$$

By Lemma 12.5, it follows that

$$u^{d-1} \in A \setminus Z(A).$$

Hence, $b = 0$. In conclusion, the K-vector space morphism

$$\psi : C \longrightarrow A$$

$$\mathbf{x} \mapsto a$$

such that $(a, \mathbf{x}) \in \widetilde{C}$ is well-defined.

Next, note that for all $\mathbf{x}_1, \ldots, \mathbf{x}_d \in C$ it holds that

$$(\psi(\mathbf{x}_1),\ \mathbf{x}_1) * \cdots * (\psi(\mathbf{x}_d),\ \mathbf{x}_d) =$$

$$(\psi(\mathbf{x}_1)\cdots\psi(\mathbf{x}_d),\ \mathbf{x}_1 * \cdots * \mathbf{x}_d) = (\overline{\psi}(\mathbf{x}_1 * \cdots * \mathbf{x}_d),\ \mathbf{x}_1 * \cdots \mathbf{x}_d).$$

The first identity holds by the definition of the $*$-product. The second identity holds by a combination of the definition of $\overline{\psi}$ with the fact that the element from the penultimate expression is contained in $\widetilde{C}^{*d}$. From the second identity we see that the multiplicative relation holds for (C, ψ). Finally, $\overline{\psi}$ is r-wise determined by the condition on $\widetilde{D}$. This proves the first claim.

The proof of the second claim is straightforward from the definition of multiplicativity. $\square$

Combining Lemmas 12.9 and 12.13, we get the following theorem (see also Remark 12.5).

Theorem 12.14 (Codex as Generalized Code) *Let A be a K-algebra with $\dim_K A < \infty$. Let n, t, d, r be integers with $d \geq 1$ and $0 \leq t < r \leq n$.*

Let

$$\widetilde{C} \subset A \times K^n$$

be a K-vector subspace. Suppose that the following holds.

- *$\pi_0(\widetilde{C}) = A$.*
- *If $t > 0$ then for each $B \subset \mathcal{I}^*$ with $|B| = t$ and for each $a \in A$, there is $\widetilde{\mathbf{x}} \in \widetilde{C}$ with $\widetilde{\mathbf{x}}_B = \mathbf{0}$ and $\widetilde{x}_0 = a$.*

Furthermore, suppose there is a K-vector subspace

$$\widetilde{D} \subset A \times K^n$$

such that

- *$\widetilde{C}^{*d} \subset \widetilde{D}$.*
- *For all $\widetilde{\mathbf{z}} \in \widetilde{D}$ it holds that $\widetilde{z}_0 = 0$ if $\widetilde{\mathbf{z}}_B = \mathbf{0}$ for some $B \subset \mathcal{I}^*$ with $|B| = r$.*

Then (C, ψ) is an (n, t, d, r)-codex for A over K, where $C := \pi_{\mathcal{I}^*}(\widetilde{C}) \subset K^n$ and where $\psi : C \longrightarrow A$ is the K-vector space morphism such that $\widetilde{C}$ consists of all $(a, \mathbf{x}) \in A \times K^n$ with $\mathbf{x} \in C$ and $a = \psi(\mathbf{x})$.

There is also a natural converse, but we do not state that here.

12.3.2 A Definition in Terms of Multi-Linear Algebra

It is also natural to develop the codex definition in the language of *multi-linear algebra.* Although we shall not use it further here, we briefly show an example [6] of such a definition in the case of $d = 2$. For concreteness, we apply it to an important class of arithmetic secret sharing schemes, namely $(n, t, 2, n-t)$-codices for $\mathbb{F}_q^k$

[6] This example is due to [44], where it is shown that the codex definition is "minimal" in a certain well-defined sense.

over $\mathbb{F}_q$ with $t \geq 1$. [7] We obtain an equivalent definition in terms of *linear algebra in a space of symmetric tensors.*

Let V be a finite-dimensional K-vector space and consider the tensor product $V \otimes_K V$. Let

$$S^2(V) \subset V \otimes_K V$$

denote the K-vector subspace of *symmetric tensors*, i.e., the K-span of all tensors $x \otimes x$ with $x \in V$.

Furthermore, let $\mathrm{Bil}(V)$ denote the space of K-bilinear forms $\Phi : V \times V \longrightarrow K$. Let $\mathrm{Sym}(V)$ denote the subspace of $\mathrm{Bil}(V)$ consisting of the *symmetric* forms Φ', i.e., $\Phi'(x, y) = \Phi'(y, x)$ for all $x, y \in V$. Let $\mathrm{Alt}(V)$ denote the subspace of $\mathrm{Bil}(V)$ consisting of *alternating* forms Φ'', i.e., $\Phi''(x, x) = 0$ for all $x \in V$.

By universality of the tensor product, there is a one-to-one correspondence between the K-linear forms $\phi : S^2(V) \longrightarrow K$ and the bilinear forms modulo the alternating forms. More precisely:

$$\mathrm{Bil}(V)/\mathrm{Alt}(V) \simeq (S^2(V))^*,$$

where an isomorphism is given by the assignment

$$\Phi + \mathrm{Alt}(V) \mapsto \phi,$$

where ϕ is the unique K-linear form on $S^2(V)$ such that

$$\phi(x \otimes x) = \Phi(x, x)$$

for all $x \in V$.

Now suppose for simplicity that the characteristic of K is different from 2. Then we have the following. Each class $\Phi + \mathrm{Alt}(V)$ contains a unique symmetric form Φ'. In particular, the isomorphism above induces a one-to-one correspondence between the elements of $(S^2(V))^*$ and the symmetric bilinear forms on V. We define the *rank* of $\phi \in (S^2(V))^*$ as the rank of the unique symmetric form Φ' by which it can thus be represented. In other words, consider the K-vector space morphism

$$V \longrightarrow V^*$$

$$x \mapsto \Phi'(x, \cdot).$$

Then the rank of Φ' is the codimension of the kernel of this map, i.e., the dimension of V minus the dimension of the kernel. In particular, ϕ is of rank 1 if there is some $\Psi_0 \in V^*$ and $\lambda \in K$ with $\lambda \neq 0$ such that

$$\Phi'(x, x) = \lambda \cdot (\Psi_0(x))^2$$

for all $x \in V$.

[7] See also the definition of *tensor-rank* of multiplication in finite fields, e.g. in [34], which has a longer history.

One way to verify these claims is by using matrices. Fixing a basis for V, there is a one-to-one correspondence between the bilinear forms on V and the $\ell \times \ell$ matrices with entries in K, where ℓ is the K-dimension of V. For each bilinear form Φ there is a unique matrix M such that $\Phi(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T M \mathbf{y}$, and vice-versa. The symmetric forms correspond to the symmetric matrices. The alternating forms correspond to the anti-symmetric matrices with main diagonal consisting of all zeros (a fact that has an easy proof).

If M is an $\ell \times \ell$ matrix with entries in K corresponding to a bilinear form Φ, then $M' := \frac{1}{2}(M + M^T)$ is a symmetric matrix and $\mathbf{x}^T M \mathbf{x} = \mathbf{x}^T M' \mathbf{x}$ for all $\mathbf{x} \in V$. Hence the symmetric bilinear form Φ' corresponding to M' is in the same class as Φ. Note that M' is well-defined since the characteristic of K is different from 2. For the same reason, the only symmetric form in the set $\Phi + \mathrm{Alt}(V)$ is indeed Φ': if there are two, then their difference is both symmetric and alternating, which is only possible if this difference is the zero matrix. The rank of Φ' is just the rank of M' as a matrix. A symmetric matrix M' of rank 1 clearly corresponds to a symmetric bilinear form that is the square of some linear form on V, up to some nonzero scalar.

Lemma 12.15 (Tensor-Product Characterization) *Suppose that the characteristic p of $\mathbb{F}_q$ satisfies $p > 2$. Let k, n, t be integers with $k \geq 1$ and $1 \leq t < n - t$. Index $\mathbb{F}_q^k \times \mathbb{F}_q^n$ with the set $\mathcal{I} = \{-k+1, \ldots, 0, 1, \ldots, n\}$. Define $\mathcal{Z} = \{-k+1, \ldots, 0\}$ and $\mathcal{I}^* = \{1, \ldots, n\}$.*

There exists an $(n, t, 2, n-t)$-arithmetic secret sharing scheme for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ if and only if there exists a finite-dimensional $\mathbb{F}_q$-vector space V and

$$\{\mathbf{z}_\ell\}_{\ell \in \mathcal{I}} \subset V$$

such that, for all $B \subset \mathcal{I}^*$, the following holds.

1. *The set of elements $\{\mathbf{z}_j\}_{j \in \mathcal{Z}}$ is linearly independent over $\mathbb{F}_q$.*
2. *If $|B| = t$, then for each $j \in \mathcal{Z}$ there is $\phi \in (S^2(V))^*$ of rank 1 such that $\phi(\mathbf{z}_i \otimes \mathbf{z}_i) = 0$ for all $i \in B$ and $\phi(\mathbf{z}_j \otimes \mathbf{z}_j) = 1$, whereas $\phi(\mathbf{z}_{j'} \otimes \mathbf{z}_{j'}) = 0$ for all $j' \in \mathcal{Z} \setminus \{j\}$ if $k > 1$.*
3. *If $|B| = n - t$, then there is no $\phi \in (S^2(V))^*$ such that $\phi(\mathbf{z}_i \otimes \mathbf{z}_i) = 0$ for all $i \in B$ and such that $\phi(\mathbf{z}_j \otimes \mathbf{z}_j) = 1$ for some $j \in \mathcal{Z}$.*

Proof We only show the "if" part. We verify that the conditions of Theorem 12.14 are satisfied. Consider the $\mathbb{F}_q$-linear code

$$C := \{(\phi(\mathbf{z}_{-k+1}), \ldots, \phi(\mathbf{z}_0), \phi(\mathbf{z}_1), \ldots, \phi(\mathbf{z}_n)) \mid \phi \in V^*\} \subset \mathbb{F}_q^k \times \mathbb{F}_q^n.$$

Let $\mathcal{I}, \mathcal{I}^*, \mathcal{Z}$ be as defined above.

By the first condition it holds that $\pi_{\mathcal{Z}}(C) = \mathbb{F}_q^k$. For each $j \in \mathcal{Z}$ and for each $B \subset \mathcal{I}^*$ with $|B| = t$, the postulated form ϕ in the second condition corresponds to the square of a linear form $\Phi_0 \in V^*$, up to some non-zero scalar. Hence, $\Phi_0(\mathbf{z}_i) = 0$ for all $i \in B$ and $\Phi_0(\mathbf{z}_j) \neq 0$.

By the third condition, Lemma 11.75 implies that the $\mathbb{F}_q$-span of any $n - t$

tensors $\mathbf{z}_i \otimes \mathbf{z}_i$ with $i \in \{1,\ldots,n\}$ contains the $\mathbb{F}_q$-span of the $\mathbf{z}_j \otimes \mathbf{z}_j$ with $i \in \{-k+1,\ldots,0\}$. Therefore, for each $j \in \{-k+1,\ldots,0\}$ and for each $B \subset \{1,\ldots,n\}$ with $|B| = n-t$ there are $(\lambda_i)_{i\in B} \in \mathbb{F}_q^{n-t}$ such that

$$\mathbf{z}_j \otimes \mathbf{z}_j = \sum_{i\in B} \lambda_i \cdot (\mathbf{z}_i \otimes \mathbf{z}_i).$$

It follows at once that

$$\chi(\mathbf{z}_j)\chi'(\mathbf{z}_j) = \sum_{i\in B} \lambda_i \cdot \chi(\mathbf{z}_i)\chi'(\mathbf{z}_i)$$

for all $\chi, \chi' \in V^*$. [8]

This implies that for all $\mathbf{z} \in C^{*2}$ and for all $B \subset \mathcal{I}^*$ with $|B| = n-t$ it holds that if $\mathbf{z}_B = \mathbf{0}$ then $\mathbf{z}_{\mathcal{Z}} = \mathbf{0}$. □

12.4 Basic Constructions

12.4.1 Polynomial Codices

THEOREM 12.16 (POLYNOMIAL CODICES) *Let A be a finite-degree extension field of K. Suppose $A = K(\alpha_0)$ for some $\alpha_0 \in A$. Define $k := \dim_K A$. Let n, t, d be integers such that*

1. $n \geq 1$, $d \geq 1$, *and* $t \geq 0$.
2. $0 < d(t+k-1) < n$.
3. $n+1 \leq |K|$ *if* $k = 1$ *and* $n \leq |K|$ *if* $k > 1$.

Then there exists an $(n, t, d, d(t+k-1)+1)$-codex for A over K. Moreover, it is unital and it has uniformity.

PROOF Let $\alpha_1,\ldots,\alpha_n \in K$ be such that $\alpha_0, \alpha_1, \ldots, \alpha_n$ are pairwise distinct. For each integer m consider the *polynomial evaluation code*

$$E(m) = \{(f(\alpha_0), f(\alpha_1), \ldots, f(\alpha_n)) \mid f(X) \in K[X]_{\leq m}\} \subset A \times K^n,$$

where $K[X]_{\leq m}$ denotes the K-vector space of polynomials of degree at most m. Using Theorem 11.84 we shall argue that the code

$$\widetilde{C} := E(t+k-1) \subset A \times K^n$$

satisfies the conditions of Theorem 12.14. Define $\mathcal{I} = \{0, 1, \ldots, n\}$ as the index set of $A \times K^n$ and define $\mathcal{I}^* = \{1, \ldots, n\}$. First, since $K(\alpha_0)$ and $K[X]_{\leq k-1}$ are isomorphic as K-vector spaces, it follows at once that $\pi_0(\widetilde{C}) = K(\alpha_0) = A$. Second (assuming $t > 0$), for each $B \subset \mathcal{I}^*$ with $|B| = t$ it follows similarly that $\pi_{0,B}(\widetilde{C}) = A \times K^{|B|}$ since $K(\alpha_0) \bigoplus K^t$ and $K[X]_{\leq t+k-1}$ are isomorphic as K-vector spaces. Third, by definition,

$$\widetilde{C}^{*d} \subset E(d(t+k-1)) \subset A \times K^n.$$

[8] This follows by universality of the tensor product, since the product $\chi(\mathbf{v})\chi'(\mathbf{v}')$ with $(\mathbf{v}, \mathbf{v}') \in V \times V$ defines a bilinear form on $V \times V$.

Finally, if $\widetilde{\mathbf{z}} = (g(\alpha_j))_{j\in\mathcal{I}} \in E(d(t+k-1))$ with $g(X) \in K[X]_{d(t+k-1)}$ satisfies $\widetilde{\mathbf{z}}_B = \mathbf{0}$ for some set $B \subset \mathcal{I}^*$ with $|B| = d(t+k-1)+1$, then g is identically 0 and hence, $g(\alpha_0) = 0$. $\square$

Remark 12.6 (Using the Point at Infinity) The condition $n+1 \leq |K|$ if $k=1$ and $n \leq |K|$ if $k>1$ can be weakened to $n \leq |K|$ if $k=1$ and $n \leq |K|+1$ if $k>1$. This is by introducing one extra evaluation in the definition of evaluation code, namely, the evaluation $f(\infty)$ defined as the coefficient of X^{t+k-1} in $f(X)$. See Section 11.7 for details, which are easily adapted to the present case.

We also have the following important variation.

Theorem 12.17 (Polynomial Codices (Variation)) *Let k be a positive integer and let $A = K^k$. Let n, t, d be integers such that*

1. *$n \geq 1$, $d \geq 1$, and $t \geq 0$.*
2. *$0 < d(t+k-1) < n$.*
3. *$n+k \leq |K|$.*

Then there exists an $(n, t, d, d(t+k-1)+1)$-codex for A over K. Moreover, it is unital and it has uniformity.

The proof of this theorem is very similar to that of Theorem 12.16. Remark 12.6 applies here as well.

Corollary 12.18 *Let $p > 0$ be a prime number and let v be a positive integer. Define $q = p^v$. Let $\mathbb{F}_q$ be a finite field of cardinality q. Let n, d, k be integers with $n \geq 1$, $d \geq 1$, $t \geq 0$ and $d(t+k-1)+1 \leq n$. Then:*

- *There is an $(n, t, d, d(t+k-1)+1)$-codex for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ if $n+k \leq q$.*
- *There is an $(n, t, d, d(t+k-1)+1)$-codex for $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$ if $n \leq q$ and $k > 1$.*

In each case the codex is unital and there is uniformity if $t \geq 1$.

12.4.2 Codices from Error Correcting Codes

There are sufficient conditions for the existence of (n, t, d, r)-codices for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ that can be conveniently stated in terms of the theory of error correcting codes.

We combine Theorem 11.80 with Theorem 12.14:

Theorem 12.19 *Let n, t, d, r, k be integers such that $0 \leq t < r \leq n$ and $k \geq 1$. Suppose $C \subset \mathbb{F}_q^{n+k}$ is an $\mathbb{F}_q$-linear code such that*

1. *$d_{min}(C^{\perp}) \geq t+k+1$.*
2. *$d_{min}(C^{*d}) \geq n-r+k+1$.*

Then there exists an (n, t, d, r)-codex for $\mathbb{F}_q^k$ over $\mathbb{F}_q$.

12.4.3 Connection with Known Schemes

We now have the following examples. Let n, t, d, k be integers such that $0 \leq t < n$ and $d, k \geq 1$.

1. Suppose $q \geq 2k-1$. For any $k > 1$, there is a $(2k-1, 2)$-arithmetic embedding for $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$. This corresponds to a *bilinear multiplication algorithm* for $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$. This is a classical notion in algebraic complexity theory, see [34].
2. Suppose $n < q$, $1 \leq t < n$. *Shamir's secret sharing scheme* [163] is an $(n, t, 1, t+1)$-codex for $\mathbb{F}_q$ over $\mathbb{F}_q$.
3. Suppose $n < q$, $1 \leq t < \frac{1}{2}n$ (resp. $1 \leq t < \frac{1}{3}n$). If $t < \frac{1}{2}n$, Shamir's scheme has *multiplication.* If, in fact, $t < \frac{1}{3}n$, then it has *strong multiplication* (see [68]). This corresponds to an $(n, t, 2, n)$-arithmetic secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$, respectively, an $(n, t, 2, n-t)$-arithmetic secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$. These properties were first used in [16, 52] in the context of secure multi-party computation.
4. Suppose $k \geq 1$, $n+k-1 < q$, $1 \leq t < \frac{1}{3}(n-2k+2)$. *Franklin-Yung*'s variation [96] on Shamir's scheme, also known as a "packed secret sharing scheme" (with strong multiplication) corresponds to an $(n, t, 2, n-t)$-arithmetic secret sharing scheme for $\mathbb{F}_q^k$ over $\mathbb{F}_q$.
5. Suppose $k > 1$, $n \leq q$, $1 \leq t < \frac{1}{3}(n-2k+2)$. *A variation on Franklin-Yung*'s scheme [55], where $\mathbb{F}_q^k$ is replaced by $\mathbb{F}_{q^k}$. This corresponds to $(n, t, 2, n-t)$-arithmetic secret sharing scheme for $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$.
6. *Construction from Self-Dual Codes* [68, 56]. Another type of elementary example is as follows [56]. Instantiating Theorem 12.19 with self-dual codes of length $n+1$ and minimum distance $d' > 2$ gives rise to an $(n, d'-2, 2, n)$-arithmetic secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$.

   Various other constructions based on general linear codes (e.g., high information rate ramp schemes with/without multiplicative properties) can also be found in [56]. These results were inspired by the monotone span program based construction of linear secret sharing schemes with multiplication from [68]. In [91], construction from Reed-Muller codes is discussed.

Note that Remark 12.6 about using the point of infinity applies here as well, giving slightly weaker requirements in some cases.

12.5 Applications

We briefly sketch some applications of codices. First it is useful to discuss some basic properties of codices.

12.5.1 Basic Properties

Let A be a K-algebra. Let n, d, r be integers with $d \geq 1$ and $1 \leq r \leq n$. Let $C \subset K^n$ be a K-vector subspace and let $\psi : C \longrightarrow A$ be a K-vector space morphism.

Definition 12.20 (Homogenization of Polynomials) *Let $m \geq 1$ be an integer and let $f \in K[X_1, \ldots, X_m]$. Then $f^\dagger \in K[X_0, X_1, \ldots, X_m]$ is the unique homogeneous polynomial*[9] *with $\deg f^\dagger = \deg f$ (total degree) such that*

$$f^\dagger(1, X_1, \ldots, X_m) = f(X_1, \ldots, X_m)$$

in $K[X_1, \ldots, X_m]$.

Most of applications in which multiplicativity plays a role rely, in part, on the following property.

Lemma 12.21 *Suppose (C, ψ) has (d, r)-multiplicativity. Let $m \geq 1$ be an integer and let $f \in K[X_1, \ldots, X_m]$ with $\deg f \leq d$. Then the following holds.*

1. *If (C, ψ) is unital, define the map*

$$\mathcal{F} : \bigoplus_{\ell=1}^{m} K^n \longrightarrow K^n$$

$$(\mathbf{x}_1, \ldots, \mathbf{x}_m) \mapsto (\ldots, f(x_{1j}, \ldots, x_{mj}), \ldots),$$

where x_{ij} denotes the j-th coordinate of $\mathbf{x}_i$ ($i = 1, \ldots, m$, $j = 1, \ldots, n$).
Then for all $\mathbf{x}_1, \ldots, \mathbf{x}_m \in C$ it holds that

$$\mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_m) \in C^{*d}$$

and

$$\psi(\mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_m)) = f(\psi(\mathbf{x}_1), \ldots, \psi(\mathbf{x}_m)).$$

2. *More generally, assume that $1 \in \operatorname{im} \psi$.*[10] *Let $\mathbf{u} \in C$ be such that $\psi(\mathbf{u}) = 1$. Define the map*

$$\mathcal{F}^\dagger_{\mathbf{u}} : \bigoplus_{\ell=1}^{m} K^n \longrightarrow K^n$$

$$(\mathbf{x}_1, \ldots, \mathbf{x}_m) \mapsto (\ldots, f^\dagger(u_i, x_{1j}, \ldots, x_{mj}), \ldots),$$

where x_{ij} denotes the j-th coordinate of $\mathbf{x}_i$ ($i = 1, \ldots, m$, $j = 1, \ldots, n$). Then for all $\mathbf{x}_1, \ldots, \mathbf{x}_m \in C$ it holds that

$$\mathcal{F}^\dagger_{\mathbf{u}}(\mathbf{x}_1, \ldots, \mathbf{x}_m) \in C^{*d}$$

and

$$\overline{\psi}(\mathcal{F}^\dagger_{\mathbf{u}}(\mathbf{x}_1, \ldots, \mathbf{x}_m)) = f(\psi(\mathbf{x}_1), \ldots, \psi(\mathbf{x}_m)).$$

[9] i.e., the monomials with nonzero coefficient all have the same degree.
[10] Of course, this condition is satisfied if ψ is surjective.

The proof of this lemma follows at once from the definition of (d, r)-multiplicativity. In the unital case, also take Remark 12.4 into account. Note that since A is a K-algebra, the evaluation $f(a_1, \dots, a_m) \in A$ is well-defined for all $a_1, \dots, a_m \in A$.

The next lemma is also useful in some protocol applications where it is desirable to have an explicit representation of $\overline{\psi}$ as a matrix that only picks the coordinates indexed by a set $B \subset \{1, \dots, n\}$ with $|B| = r$.

Lemma 12.22 *(C, ψ) has (d, r)-multiplicativity if and only if, for each set $B \subset \{1, \dots, n\}$ with $|B| = r$, there exists a K-vector space morphism*

$$\rho^B : K^n \longrightarrow A$$

such that the following conditions are satisfied.

1. $\rho^B(\prod_{i=1}^d \mathbf{x}_i) = \prod_{i=1}^d \psi(\mathbf{x}_i)$ *for all* $(\mathbf{x}_1, \dots, \mathbf{x}_d) \in C^d$*, i.e.,* ρ^B *satisfies the same multiplicative relation as* $\overline{\psi}$ *does.*
2. $\ker \pi_B \subset \ker \rho^B$*, i.e.,* ρ^B *"ignores all coordinates* $i \notin B$*".*

Proof The proof is by elementary linear algebra. In the forward direction, select an arbitrary $B \subset \{1, \dots, n\}$ with $|B| = r$. The map $\overline{\psi}$ is defined on C^{*d} and vanishes on $C^{*d} \cap \ker \pi_B$ by definition. Now extend it to the K-vector subspace $C^{*d} + \ker \pi_B \subset K^n$ such that it vanishes on *all* of $\ker \pi_B$. This extension is of course unique. The required map ρ^B can now be taken as an arbitrary extension of the latter to all of K^n.

In the other direction, select an arbitrary $B \subset \{1, \dots, n\}$ with $|B| = r$ and define $\overline{\psi}$ as the restriction of the map ρ^B to C^{*d}. It follows at once that $\overline{\psi}$ satisfies the multiplicative relation. This definition of $\overline{\psi}$ is independent of B, as the action of ρ^B on $m_d(C)$ (and hence, that on all of C^{*d}) only depends on ψ. Therefore, if $\mathbf{z} \in C^{*d}$ satisfies $\pi_{B'}(\mathbf{z}) = \mathbf{0}$ for some $B' \subset \{1, \dots, n\}$ with $|B'| = r$, it holds that $\overline{\psi}(\mathbf{z}) = \rho^{B'}(\mathbf{z}) = 0$. $\square$

Remark 12.7 (Recombination Vectors) Write $k := \dim_K A$ and suppose $k < \infty$. Once a K-basis for A is fixed (as well as, say, the standard basis for K^n), each ρ^B may be given by a matrix with k rows and n columns and entries in K. If $A = K$, for instance, each ρ^B is thus representable by a K-linear form $\langle \mathbf{r}^B, \cdot \rangle$ with $r_i^B = 0$ if $i \notin B$.

The definition of "recombination vectors" originates with [68] where multiplicative properties of linear secret sharing schemes were first formalized and the first general results were obtained.

12.5.2 Multiparty Computation Secure Against a Passive Adversary

Consider an $(n, t, 2, n)$-arithmetic secret sharing scheme (C, ψ) for $\mathbb{F}_q^k$ over $\mathbb{F}_q$. Index $\mathbb{F}_q^k \times \mathbb{F}_q^n$ with the set $\mathcal{I} = \{-k+1, \dots, 0, 1, \dots, n\}$. Define $\mathcal{Z} = \{-k+1, \dots, 0\}$ and $\mathcal{I}^* = \{1, \dots, n\}$.

Such a scheme can be used to reduce n-party secure multiplication to secure addition in the case of a *passive adversary*, at the cost of one round of interaction. This is done by taking the protocol we presented in Chapter 3 for Shamir's scheme and generalizing it in the natural way.

On account of Remark 12.7, there is a matrix ρ with k rows and n columns and entries in $\mathbb{F}_q$ such that, for all $\mathbf{x}, \mathbf{x}' \in C$,

$$\rho(x_1x_1', \dots, x_nx_n')^T = \psi(\mathbf{x}) * \psi(\mathbf{x}') \in \mathbb{F}_q^k.$$

The reduction works as follows. As in Chapter 3, we have n players arranged in a complete communication network of perfectly private channels between each pair of players. There is a passive adversary and a bound t on the number of players it can corrupt. Recall that a passive adversary knows all about each of the at most t players it corrupts but cannot modify their actions. If a player is not corrupted, it is honest and executes its required actions faithfully.

The purpose of the protocol to follow is to establish a secret-sharing of the product of two secrets, given secret-sharings of these secrets. Of course, no information should leak in the process. Moreover, the secret-sharing of the product should be in the same scheme as that of the two secrets, so that it facilitates an ongoing secure computation.

Suppose that the network has access to secret-sharings $\mathbf{x}, \mathbf{x}' \in C$, with respective secrets $\psi(\mathbf{x}), \psi(\mathbf{x}') \in \mathbb{F}_q^k$. It does not matter how these sharings have come into existence: for instance, perhaps they were obtained from some external source or from different players in the network or they may have resulted from a previous secure computation. We assume that, for $i = 1, \dots, n$, player P_i holds share x_i in secret $\psi(\mathbf{x}) \in \mathbb{F}_q^k$ and share x_i' in secret $\psi(\mathbf{x}') \in \mathbb{F}_q^k$.

Now, for $i = 1, \dots, n$, player P_i secret-shares the locally computable secret value

$$(\rho_{i1}x_ix_i', \dots, \rho_{ik}x_ix_i') \in \mathbb{F}_q^k,$$

where the coefficient vector is the i-th row of the matrix ρ, by choosing a uniformly random element $\mathbf{x}^{(i)} \in C$ such that

$$\psi(\mathbf{x}^{(i)}) = (\rho_{i1}x_ix_i', \dots, \rho_{ik}x_ix_i')^T,$$

and, for $j = 1, \dots, n$, sending the j-th share $x_j^{(i)}$ to player P_j over the private channel connecting the two.

Next, for $j = 1, \dots, n$, player P_j sums the n received shares. This gives a secret-sharing of $\psi(\mathbf{x}) * \psi(\mathbf{x}')$ according to C (see e.g. [55]). This generalizes Shamir-based solutions from [16, 52, 96], see also [68].[11]

[11] The "local share-multiplication plus re-sharing" simplification (in the case of Shamir's scheme) has been attributed to Michael Rabin.

12.5.3 Zero-Knowledge Verification of Secret Multiplications

Consider an $(n, t, 2, n - t)$-arithmetic secret sharing scheme (C, ψ) for $\mathbb{F}_q^k$ over $\mathbb{F}_q$. Such a scheme can be used for "zero-knowledge verification of secret multiplications." In a nutshell, the main idea is as follows. Suppose a prover and a verifier have access to an $\mathbb{F}_q$-linear cryptographic commitment scheme, allowing to commit to a value in $\mathbb{F}_q$. Before the commitment is opened, the verifier gains no knowledge about the $\mathbb{F}_q$-value committed to (hiding property), while the prover can open the commitment later only to reveal the original value (binding property). Linearity means that $\mathbb{F}_q$-linear combinations of commitments can be taken non-interactively. [12]

Suppose a prover gives (coordinate-wise) commitments to secrets $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0 \in \mathbb{F}_q^k$, and claims that $\mathbf{x}_0 * \mathbf{y}_0 = \mathbf{z}_0$. The purpose of the protocol below is for the prover to convince a sceptical verifier that this is indeed true (*correctness*), without revealing the secrets in the process (*privacy*).

To prove the claim, the prover gives (coordinate-wise) commitments to random $\mathbf{x}, \mathbf{y} \in C$ such that $\psi(\mathbf{x}) = \mathbf{x}_0$ and $\psi(\mathbf{y}) = \mathbf{y}_0$, and a (coordinate-wise) commitment to some $\mathbf{z} \in C^{*2}$ such that $\overline{\psi}(\mathbf{z}) = \mathbf{z}_0$. If the commitment scheme is $\mathbb{F}_q$-linear, it is easy to design a mechanism so that, regardless of the prover's honesty, the correctness constraints are satisfied that, indeed, $\mathbf{x}, \mathbf{y} \in C$ and $\mathbf{z} \in C^{*2}$, and indeed $\psi(\mathbf{x}) = \mathbf{x}_0$, $\psi(\mathbf{y}) = \mathbf{y}_0$, and $\overline{\psi}(\mathbf{z}) = \mathbf{z}_0$.

Moreover, the mechanism allows the prover to select $\mathbf{z} = \mathbf{x} * \mathbf{y}$ and, in fact, the *honest* prover does so. Note that gives

$$\mathbf{z}_0 = \overline{\psi}(\mathbf{z}) = \overline{\psi}(\mathbf{x} * \mathbf{y}) = \mathbf{x}_0 * \mathbf{y}_0.$$

Meanwhile, the mechanism protects privacy of the honest prover's secrets. However, there is no guarantee about the behavior of a *cheating* prover in this respect. See for instance [68] to fill in the details of this mechanism.

So, at this point, the verifier has no knowledge about the values that have been committed to, except that they satisfy the constraints above.

Now, if $\mathbf{z} = \mathbf{x} * \mathbf{y}$ (as in the case of an honest prover), then indeed $\mathbf{x}_0 * \mathbf{y}_0 = \mathbf{z}_0$. In this case, inspection by the verifier (after opening by the prover) of any t "share-triples" (x_i, y_i, z_i) gives no information on the "secret-triple" $(\mathbf{x}_0, \mathbf{y}_0, \mathbf{x}_0 * \mathbf{y}_0)$, by t-privacy. Moreover, $z_i = x_i y_i$ for each of those t share-triples.

On the other hand, suppose that there is a cheating prover and that $\mathbf{z}_0 \neq \mathbf{x}_0 * \mathbf{y}_0$. Then there are at most $n - t - 1$ share-triples (x_i, y_i, z_i) such that $z_i = x_i y_i$, and hence there are at least $t + 1$ share-triples for which there is an inconsistency. This follows from $(2, n - t)$-multiplicativity.

If t is not too small compared to n and if the verifier choose t positions i at

[12] i.e., any secret value taken in $\mathbb{F}_q$ can be committed to and just given two commitments to respective secrets s, s' and given a scalar $\lambda \in \mathbb{F}_q$ as inputs, a commitment to $s + \lambda s' \in \mathbb{F}_q$ can be efficiently computed. Moreover, given opening information for the input commitments, opening information for the output commitment can be efficiently computed.

random, it is with high probability that this cheating prover is exposed. By the observations above, there are no inconsistencies in the case an honest prover and moreover, the prover's privacy is guaranteed. This procedure above is essentially from [68], building on ideas from [16, 52].

These facts together give a handle for probabilistical private checking that $\mathbf{z}_0 = \mathbf{x}_0 * \mathbf{y}_0$ in several different application scenarios, most notably perfect information-theoretically secure general multi-party computation, in the case of a *malicious adversary.* In [69] a quite efficient method is given for checking in zero-knowledge that a private vector of committed values satisfies a given arithmetic circuit. This involves arithmetic secret sharing schemes with $d > 2$.

12.5.4 Error Correction in Codices

The important subclass of arithmetic secret sharing schemes that satisfy $r = n - t$ offers efficient recovery of the secret even if a given full share vector contains some (malicious) errors. The method given below can be understood as a variation on a decoding method based on *error correcting pairs* [151, 123, 89], which in itself is a generalization of the *Welch-Berlekamp algorithm* (Section 11.8.2).

THEOREM 12.23 *[70, 46] Let A be an $\mathbb{F}_q$-algebra with $\dim_k A < \infty$. Suppose (C, ψ) is an $(n, t, d, n-t)$-arithmetic secret sharing scheme for A over $\mathbb{F}_q$. On input $(i, \tilde{\mathbf{x}})$, where i is an integer with $1 \le i \le d-1$ and where*

$$\tilde{\mathbf{x}} := \mathbf{x} + \mathbf{e} \ \in \mathbb{F}_q^n,$$

with $\mathbf{x} \in C^{*(d-i)}$ and with $\mathbf{e} \in \mathbb{F}_q^n$ such that $w_{\mathrm{H}}(\mathbf{e}) \le t$, the value

$$a := \psi(\mathbf{x}) \ \in A,$$

is recovered deterministically by linear algebra.

PROOF Recall that $d \ge 2, t \ge 1$ by definition of an arithmetic secret sharing scheme. For convenience of exposition, suppose it is unital. Consequently, the map $\overline{\psi}$ is an extension of the map ψ since, in this case, $C \subset C^{*(d-1)} \subset C^{*d}$. See Section 12.5.1. Moreover, it is therefore sufficient to show the case $i = 1$.

Consider the system of equations

$$\widetilde{\mathbf{x}} * \mathbf{z}' = \mathbf{z} \ \text{ and } \ \psi(\mathbf{z}') = 1$$

in the pair of unknowns

$$(\mathbf{z}', \mathbf{z}) \in C \times C^{*d}.$$

Note that this is in fact a *linear* system of equations (taking into account that, of course, membership of a subspace can be captured by a linear system of equations). If a generator matrix of C is given, an $\mathbb{F}_q$-basis for A is selected, and ψ is given by a matrix, then this system can be made explicit.

We prove that, first, this system has *some* solution $(\mathbf{z}', \mathbf{z}) \in C \times C^{*d}$, and that, second, *any* solution $(\mathbf{y}', \mathbf{y}) \in C \times C^{*d}$ satisfies

$$\psi(\mathbf{y}) = a.$$

First, define $B \subset \{1, \ldots, n\}$ as the set of all j with $e_j \neq 0$. Let $\overline{B}$ denote its complement.

If $B \neq \emptyset$, let $\mathbf{z}' \in C$ be such that $\psi(\mathbf{z}') = 1$ and $\mathbf{z}'_B = \mathbf{0}$. Else, let $\mathbf{z}' \in C$ be such that $\psi(\mathbf{z}') = 1$. Since $|B| \leq t$, this is possible from the assumptions in either case.

Then

$$(\mathbf{z}', \mathbf{z}) \in C \times C^{*d} \text{ with } \mathbf{z} := \mathbf{x} * \mathbf{z}'$$

is a solution since $\psi(\mathbf{z}') = 1$ and

$$\widetilde{\mathbf{x}} * \mathbf{z}' = \mathbf{x} * \mathbf{z}' = \mathbf{z}.$$

Second, let $(\mathbf{y}', \mathbf{y}) \in C \times C^{*d}$ be *any* solution. Since

$$(\widetilde{\mathbf{x}} * \mathbf{y}')_{\overline{B}} = (\mathbf{x} * \mathbf{y}')_{\overline{B}} = \mathbf{y}_{\overline{B}},$$

$$\mathbf{x} * \mathbf{y}' \in C^{*d}$$

and $|\overline{B}| \geq n - t$, it follows that

$$\psi(\mathbf{y}) = \psi(\mathbf{x} * \mathbf{y}') = \psi(\mathbf{x})\psi(\mathbf{y}') = a \cdot 1 = a,$$

as desired. □

Theorem 12.23 differs from the result in [151, 123] in that, first, it is not the classical error correction scenario for linear codes that is considered here, but rather recovery of just the algebra-element presented by a corrupted codex element. Second, the method from [151, 123] is given in terms of minimum distance, whereas the method here relies in essence on a measure closely related to the weights from Theorem 11.69. Third, recovery in higher powers is considered here, not just in the "base code." Fourth, the respective requirements do not appear to be easily comparable.

That said, the spirit of the proof of Theorem 12.23 bears similarity to that of [151, 123]. In [88], the method from [70] for the special case of strongly multiplicative linear secret sharing schemes is explained from the perspective of [151, 123]. For relaxed conditions under which the theorem holds, please refer to [46].

12.5.5 Bilinear Multiplication Algorithms

We conclude with a classical application from algebraic complexity theory. If (C, ψ) is an $(n, 2)$-arithmetic embedding, say for $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$, then it may be interpreted as a specialized "vectorial representation" of the extension field $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$. Extension field addition (resp., scalar multiplication) amounts to vector addition (resp., multiplication of a vector by a scalar). Extension field multiplication requires just the coordinate-wise product of vectors, followed by a linear map. All these operations are over the base field $\mathbb{F}_q$.

Concretely, this interpretation can be given as follows. By definition of an arithmetic embedding, the map

$$\psi : C \longrightarrow \mathbb{F}_{q^k}$$

is an isomorphism of $\mathbb{F}_q$-vector spaces. Therefore, the presentation of the elements of $\mathbb{F}_{q^k}$ by the elements from C is one-to-one. By Remark 12.7 there is an $\mathbb{F}_q$-vector space morphism

$$\rho : \mathbb{F}_q^n \longrightarrow \mathbb{F}_{q^k}$$

such that, for all $\mathbf{x}, \mathbf{y} \in C$,

$$\psi^{-1} \circ \rho(\mathbf{x} * \mathbf{y}) \in C$$

presents

$$\psi(\mathbf{x}) \cdot \psi(\mathbf{y}) \in \mathbb{F}_{q^k}.$$

Note that all operations (addition and scalar multiplication on the one hand and multiplication on the other hand) are over the base field $\mathbb{F}_q$ and that multiplications only arise in the product $\mathbf{x} * \mathbf{y}$, which gives n of them in total.

In a model where the resources needed for addition as well as scalar multiplication (each over the base field) can be neglected, it makes sense to consider multiplication algorithms minimizing just the number of multiplications over the base field. For an in-depth treatment and history, please consult [34].

In Theorem 12.16 we have given a method for constructing arithmetic embeddings for finite field extensions. In general, there is the following classical generic construction of arithmetic embeddings, once again stated and analyzed in our framework.

Theorem 12.24 *Let A be a K-algebra with $1 \leq dim_K A < \infty$. Define $k :=$ $\dim_K A$. There exists a $(\frac{1}{2}k(k+1), 2)$-arithmetic embedding of A over K.*

Proof The statement is trivial for $k = 1$. So assume $k > 1$. Choose a K-basis $e_1, \ldots, e_k$ of A as a K-vector space. For $a \in A$, let $\mathbf{a} \in K^k$ denote the coordinate vector of a with respect to this basis, i.e.,

$$a = a_1 e_1 + \cdots + a_k e_k.$$

Consider the K-vector space morphism

$$\phi : A \longrightarrow K^{\frac{1}{2}k(k+1)}$$

$$a \mapsto (a_1, \ldots, a_k, \ldots, a_i + a_j, \ldots)$$

where $1 \leq i < j \leq k$. Note that this map is injective. Define

$$C = \phi(A).$$

Furthermore, define

$$\psi : C \longrightarrow A$$

as the unique K-vector space morphism such that

$$\psi \circ \phi(a) = a$$

for all $a \in A$. This is possible since ϕ is injective. Note that ψ essentially "picks" the first k coordinates $x_1, \ldots, x_k$ of an argument $\mathbf{x} \in C$ and sets

$$\psi(\mathbf{x}) = x_1 e_1 + \cdots + x_k e_k.$$

For all $a_1, \ldots, a_k, b_1, \ldots, b_k \in K$, it holds that

$$(a_1 e_1 + \cdots + a_k e_k) \cdot (b_1 e_1 + \cdots + b_k e_k) =$$

$$\sum_{1 \le i \le k} a_i b_i e_i^2 + \sum_{1 \le i < j \le k} (a_i b_j + a_j b_i) e_i e_j =$$

$$\sum_{1 \le i \le k} a_i b_i e_i^2 + \sum_{1 \le i < j \le k} ((a_i + a_j)(b_i + b_j) - a_i b_i - a_j b_j) e_i e_j.$$

One now quickly sees that there is a K-vector space morphism $\overline{\psi} : C^{*2} \longrightarrow A$ as required in Definition 12.12. In fact, Lemma 12.22 in combination with the subsequent Remark 12.7 renders the conclusion immediate by inspection of the latter identity. $\square$

For advanced results, consult the references given in Section 12.8.

12.6 Limitations on Codices

We show some fundamental limitations on realizable codex parameters. In particular, we give some initial remarks about asymptotic properties.

12.6.1 Remarks on the Powering Operation

It is worth noting that (n, t, d, r)-codices involve subspaces "whose d-th power does not fill the whole space":

LEMMA 12.25 *Let A be a K-algebra. Let n, d, r be integers with $d \ge 1$ and $1 \le r \le n$. Let $C \subset K^n$ be a K-vector subspace and let $\psi : C \longrightarrow A$ be a K-vector space morphism. If (C, ψ) has (d, r)-multiplicativity, then*

$$\widetilde{C}^{*d} \subsetneq A \times K^n.$$

Suppose that the only nilpotent in A is 0 or that $\operatorname{im} \psi \not\subset Z(A)$. [13] *If $r < n$ and $\psi \not\equiv 0$, then the stronger property holds that*

$$C^{*d} \subsetneq K^n.$$

[13] See Remark 12.3.

Proof The first part of the remark is trivial: since $\overline{\psi}(\mathbf{0}) = 0$, there is in particular no element $(a, \mathbf{0}) \in \widetilde{C}^{*d}$ with $a \neq 0$. As to the second part, write $\mathbf{e}_1, \ldots, \mathbf{e}_n \in K^n$ for the vectors of the standard basis. Towards a contradiction, suppose that $C^{*d} = K^n$. It follows that $\mathbf{e}_1, \ldots, \mathbf{e}_n \in C^{*d}$ and that

$$\overline{\psi}(\mathbf{e}_1) = \ldots = \overline{\psi}(\mathbf{e}_n) = 0,$$

since the Hamming-weight w of each of these $\mathbf{e}_i$'s satisfies $1 = w \leq n - r$. Therefore,

$$\overline{\psi} \equiv 0.$$

We show that this leads to the contradiction that

$$\psi \equiv 0.$$

Let $\mathbf{x} \in C$ with $\mathbf{x} \neq \mathbf{0}$. Then

$$0 = \overline{\psi}(\mathbf{x}^d) = (\psi(\mathbf{x}))^d.$$

If $d = 1$, then $\psi(\mathbf{x}) = 0$. So assume $d > 1$. If the only nilpotent in A is 0, it follows that $\psi(\mathbf{x}) = \mathbf{0}$. Otherwise, let $\mathbf{y} \in C$ be such that

$$\psi(\mathbf{y}) \in A \setminus Z(A).$$

Then

$$0 = \overline{\psi}(\mathbf{x} * \mathbf{y}^{d-1}) = \psi(\mathbf{x}) \cdot (\psi(\mathbf{y}))^{d-1}.$$

By Lemma 12.5, it follows that $(\psi(\mathbf{y}))^{d-1} \in A \setminus Z(A)$. Hence, $\psi(\mathbf{x}) = \mathbf{0}$. The claim follows. □

If $r = n$, then it does *not* necessarily hold that $C^{*d} \subsetneq K^n$. For instance, if $K = \mathbb{F}_2$ and $A = \mathbb{F}_4$ then the construction from Theorem 12.16 (assuming that the subsequent instruction from Remark 12.6 is implemented), gives a $(3, 2)$-arithmetic embedding (C, ψ) of $\mathbb{F}_4$ over $\mathbb{F}_2$ if we set $k = 2, n = 3, d = 2$ and $t = 0$. However, one checks by hand that $C^{*2} = \mathbb{F}_2^3$.

The powering operation is *not benign*, in a sense. For instance, $C^{*2} = K^n$ for a generic subspace $C \subset K^n$ of dimension $\gg \sqrt{n}$ when K is a fixed finite field. We give here just one elementary reason pointing in that direction. The general result [45] hinted at here involves some nontrivial facts from the theory of quadratic forms over finite fields.

Lemma 12.26 *Suppose $C \subset K^n$ ($n > 2$) is a K-vector subspace with $C \neq \{\mathbf{0}\}, K^n$. Suppose t is an integer such that $d_{\min}(C^{\perp}) > t > 1$. Then $C^{*\lceil \frac{n-1}{t-1} \rceil} = K^n$.*

Proof The conditions imply that $\pi_B(C) = K^t$ for all $B \subset \{1, \ldots, n\}$ with $|B| = t$. Now construct the n standard basis-vectors $\mathbf{e}_i$ of K^n one-by-one, as follows. Without loss of generality, consider just $\mathbf{e}_1 = (1, 0, \ldots, 0) \in K^n$. Select a vector in C such that its "left-most" coordinate equals 1, followed by a window of $t - 1 > 0$ consecutive 0's. Next, do as before, except that the window of 0's starts right after where the previous ended. Repeat this until the "end of the vector has been reached" (where, in the very last step, the window may possibly be of smaller size than $t-1$, of course). This way, $\lceil \frac{n-1}{t-1} \rceil$ vectors in C are obtained whose coordinate-wise product equals $\mathbf{e}_1$. □

From the lemma one sees that if $t = \Omega(n)$, then some constant power of C fills up all of K^n.

We now show an upper bound on the parameter d in the case where $A = \mathbb{F}_{q^k}$. [14]

Theorem 12.27 *[47] Suppose (C, ψ) is an (n, t, d, r)-codex for $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$ with $k \geq 2$. Then*

$$d \leq q.$$

Proof Towards a contradiction, suppose $d \geq q+1$. Since the Frobenius map, i.e., the q-th power map, acts on $\mathbb{F}_q^k$ as the identity, it follows that

$$\mathbf{x}^q * \mathbf{y}^{d-q} = \mathbf{x} * \mathbf{y}^{d-1}$$

for all $\mathbf{x}, \mathbf{y} \in C$. Define

$$a := \psi(\mathbf{x}) \ , \ b := \psi(\mathbf{y}).$$

Then this implies

$$a^q b^{d-q} = \overline{\psi}(\mathbf{x}^q * \mathbf{y}^{d-q}) = \overline{\psi}(\mathbf{x} * \mathbf{y}^{d-1}) = \psi(\mathbf{x}) \cdot \psi(\mathbf{y})^{d-1} = ab^{d-1}.$$

If $a \neq 0$ and $b \neq 0$, this implies that

$$a^{q-1} = b^{q-1}.$$

However, since $k \geq 2$ there are $a_0, b_0 \in \mathbb{F}_{q^k} \setminus \{0\}$ with

$$a_0^{q-1} \neq b_0^{q-1}.$$

Since ψ is surjective, there are $\mathbf{x}, \mathbf{y} \in C$ such that

$$\psi(\mathbf{x}) = a_0 \ , \ \psi(\mathbf{y}) = b_0,$$

and a contradiction arises. □

Remark 12.8 (A Generalization) Results from [33, 157] give the stronger result that $d \leq q$ for any finite-dimensional $\mathbb{F}_q$-algebra A such that $A \not\simeq \mathbb{F}_q^k$ where $k = \dim_K A$.

We verify this by combining the proof strategy above with the algebra classification from Theorem 10.11. Let A be a finite-dimensional $\mathbb{F}_q$-algebra. Suppose there is an (n, t, d, r)-codex for A over $\mathbb{F}_q$. Conducting the argument above with $\mathbb{F}_{q^k}$ replaced by A, it follows that $d \leq q$ if the property holds that the Frobenius map does not act as the identity on A. [15] Suppose A *does* have the property that the Frobenius map acts as the identity on A. Then A also has the property of

[14] See [34] for lower bounds on bilinear complexity.
[15] To see this, select $\mathbf{y} \in C$ such that $\psi(\mathbf{y}) = 1$ and define $b = 1$ at the start of the proof.

having 0 as its only nilpotent. [16] Thus, Theorem 10.11 applies and A is a finite product of finite extensions of $\mathbb{F}_q$. Since the Frobenius map acts as the identity on A, each of these extensions has degree 1. This concludes the proof.

On a historical note, mathematical considerations on products of codes or powers of codes in the sense of the $*$-product have been present e.g. in work from the 1980s on *a proof of the Roos bound for error correcting codes* [177], on *secure multi-party computation* [16, 52] (essentially, the square of certain Reed-Solomon codes or Shamir's secret sharing scheme), *bilinear complexity of extension field multiplication* (e.g., [57]), and in work from the 1990s on *error correction algorithms* [151, 123] (so-called error-correcting pairs), once again *secure multi-party computation* [68] ((strongly) multiplicative properties of general linear secret sharing schemes). Some years later, motivated in part by *asymptotics of secure multi-party computation*, (see the overview in Section 12.8), further systematic study of the notion has been conducted, starting with [54] and followed by [56, 43, 63, 46, 157, 155, 48, 140, 156, 49]. Novel applications have been proposed in [119, 117, 109, 116, 76, 115, 69, 60, 83, 61]. Most of these deal with various aspects of secure two-party (or multi-party) cryptography, while some others are concerned with cryptanalysis of public-key cryptosystems based on error-correcting codes or geometry of numbers.

It is interesting to observe that, so far, the most striking applications of codices in secure multi-party computation or in two-party cryptography not only make requirements on powers of the code, but also on its dual.

12.6.2 Bound on Arithmetic Embeddings

We discuss a classical bound for arithmetic embeddings (see e.g. [34]), stated and analyzed in our framework.

THEOREM 12.28 *Let A be an extension field of K with $k := \dim_K A < \infty$. Let n be a positive integer. Suppose (C, ψ) is an $(n, 2)$-arithmetic embedding of A over K. Then $n \geq 2k - 1$.*

PROOF We claim that the minimum distance of C is at least k. Hence, by the Singleton Bound, $k \leq n - k + 1$ and the desired result follows. We now prove the claim. Towards a contradiction, let $\mathbf{x} \in C$ with $\mathbf{x} \neq \mathbf{0}$ and $w_{\mathrm{H}}(\mathbf{x}) \leq k - 1$. Consider the K-vector space $\mathbf{x} * C \subset K^n$. Since $w_{\mathrm{H}}(\mathbf{x}) \leq k-1$ it follows that this space has dimension at most $k - 1$. However, a contradiction arises as it follows from $\mathbf{x} \neq \mathbf{0}$ that the K-vector space morphism $\overline{\psi}$ restricted to $\mathbf{x}*C$ is a surjection onto A, which has dimension k. Indeed, $\overline{\psi}(\mathbf{x} * \mathbf{y}) = \psi(\mathbf{x}) \cdot \psi(\mathbf{y})$ for all $\mathbf{y} \in C$ and $\psi(\mathbf{x}) \neq 0$ since ψ is an isomorphism of K-vector spaces. This proves the claim. □

Generally, if there is an (n, d)-arithmetic embedding of the extension field A over K with $d \geq 2$, then $n \geq dk - (d - 1)$.

[16] Namely, $x^q = x$ and $x^m = 0$ for some $x \in A$ and for some positive integer m implies $x = 0$. This is clear if $m \geq q$. If $m > q$ it is reduced inductively until $m < q$.

12.6.3 General Bounds on Codices

LEMMA 12.29 *Let A be a K-algebra with $\dim_K A < \infty$. If (C, ψ) is an (n, t, d, r)-codex for A over K, then $(C^{*d}, \overline{\psi})$ is an $(n, dt, 1, r)$-codex for A over K. In particular,*

$$dt < r.$$

PROOF We first show that the hypothesis on (C, ψ) implies that $dt < r$. If $t = 0$ this holds by default. So assume $t > 0$. Towards a contradiction, suppose $dt \geq r$. Index K^n with the set $\mathcal{I} = \{1, \ldots, n\}$. Choose sets $B_1, \ldots, B_d \subset \mathcal{I}$ and elements $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(d)} \in C$ such that

- For $i = 1, \ldots, d$, it holds that $|B_i| = t$.
- $| \cup_{i=1}^d B_i| \geq r$.
- For $i = 1, \ldots, d$, it holds that $\mathbf{y}_{B_i}^{(i)} = \mathbf{0}$ and $\psi(\mathbf{y}^{(i)}) = 1$. Note that this makes sense on account of Lemma 12.9.

Then

$$\overline{\psi}\left(\prod_{i=1}^d \mathbf{y}^{(i)}\right) = \prod_{i=1}^d \psi(\mathbf{y}^{(i)}) = 1.$$

However,

$$w_{\mathrm{H}}\left(\prod_{i=1}^d \mathbf{y}^{(i)}\right) \leq n - r \quad \text{and} \quad \prod_{i=1}^d \mathbf{y}^{(i)} \in C^{*d}.$$

Hence, by (d, r)-multiplicativity,

$$\overline{\psi}\left(\prod_{i=1}^d \mathbf{y}^{(i)}\right) = 0,$$

a contradiction. Therefore, $dt < r$ as desired.

We now show the claim about $(C^{*d}, \overline{\psi})$. It is easy to see that the map

$$\overline{\psi} : C^{*d} \longrightarrow A$$

is surjective. If $d = 1$, then $C^{*d} = C$ and $\overline{\psi}$ is the same as ψ. Hence, the claim follows from the codex definition in this case. Now assume $d > 1$. For given $a \in A$, choose $\mathbf{v} \in C$ such that $\psi(\mathbf{v}) = a$. Furthermore, choose $\mathbf{u} \in C$ such that $\psi(\mathbf{u}) = 1$. Now define

$$\mathbf{z} = \mathbf{u}^{d-1} * \mathbf{v}$$

and note that

$$\overline{\psi}(\mathbf{z}) = \overline{\psi}(\mathbf{u}^{d-1} * \mathbf{v}) = (\psi(\mathbf{u}))^{d-1} * \psi(\mathbf{v}) = a$$

by the codex definition. Therefore, the pair $(C^{*d}, \overline{\psi})$ constitutes an $(n, t', 1, r)$-codex for A over K for some integer $t' \geq 0$.

We now show that we can take $t' = dt$. If $t = 0$, then this is trivial. So assume that $t > 0$. Let $B \subset \mathcal{I}$ be such that $|B| = dt$. Since $dt < r \leq n$ this makes sense.

Select some disjoint partition of B into subsets $B_1, \ldots, B_d$, each of cardinality t. Then select $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(d)} \in C$ such that $\psi(\mathbf{y}^{(i)}) = 1$ and $\pi_{B_i}(\mathbf{y}^{(i)}) = \mathbf{0}$ for $i = 1, \ldots, d$. This makes sense on account of Lemma 12.9. Finally, define

$$\mathbf{y} = \mathbf{y}^{(1)} * \cdots * \mathbf{y}^{(d)}$$

and note that

$$\overline{\psi}(\mathbf{y}) = \overline{\psi}(\mathbf{y}^{(1)} * \cdots * \mathbf{y}^{(d)}) = \psi(\mathbf{y}^{(1)}) * \cdots * \psi(\mathbf{y}^{(d)}) = 1$$

and

$$\pi_{B_i}(\mathbf{y}) = \mathbf{0}.$$

The claim follows from Lemma 12.9. □

A small variation of the argument above implies the following:

LEMMA 12.30 *Let A be a K-algebra with $\dim_K A < \infty$. If (C, ψ) is an (n, t, d, r)-codex for A over K then it is an $(n, t, 1, r - (d-1)t)$-codex for A over K.*

PROOF If $t = 0$, then the claim follows from Lemma 12.6. So assume $t > 0$. Index K^n with the set $\mathcal{I} = \{1, \ldots, n\}$. Let $\mathbf{x} \in C$ and $B \subset \mathcal{I}$ such that

$$\mathbf{x}_B = \mathbf{0} \ \text{ and } \ |B| = r - (d-1)t.$$

By Lemma 12.29, it holds that $r - (d-1)t > 0$. Therefore, the selection above makes sense. We have to show that $\psi(\mathbf{x}) = 0$.

Choose sets $B_1, \ldots, B_{d-1} \subset \mathcal{I}$ and elements $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(d-1)} \in C$ such that

- For $i = 1, \ldots, d-1$, it holds that $|B_i| = t$.
- $B, B_1, \ldots, B_{d-1}$ are pairwise disjoint (so their union has cardinality $r \leq n$).
- For $i = 1, \ldots, d-1$, it holds that $\mathbf{y}^{(i)}_{B_i} = \mathbf{0}$ and $\psi(\mathbf{y}^{(i)}) = 1$.

Then

$$\mathbf{x} * \prod_{i=1}^{d-1} \mathbf{y}^{(i)} \in C^{*d} \ \text{ and } \ w_{\mathrm{H}}\left(\mathbf{x} * \prod_{i=1}^{d-1} \mathbf{y}^{(i)}\right) \leq n - r.$$

Hence, by (d, r)-multiplicativity,

$$0 = \overline{\psi}\left(\mathbf{x} * \prod_{i=1}^{d-1} \mathbf{y}^{(i)}\right) = \psi(\mathbf{x}) \cdot \prod_{i=1}^{d-1} \psi(\mathbf{y}^{(i)}) = \psi(\mathbf{x}).$$

Thus, ψ is $(r - (d-1)t)$-determined. □

Combining Lemmas 12.29 and 12.30, whose proof ideas are inspired by results from [68], with the gap bound from Corollary 11.105 from [48] or its refinements, implies nontrivial bounds on codices with $t > 0$.

12.6.4 An Asymptotical Bound

The bound $dt < r$ from Lemma 12.29 can be strengthened substantially by application of Theorem 11.104 (which originates from [48]) as follows.

THEOREM 12.31 *[47] Let A be an $\mathbb{F}_q$-algebra with $\dim_{\mathbb{F}_q} A < \infty$. If there is an (n, t, d, r)-codex for A over $\mathbb{F}_q$ with $d > 1$ and $t > 0$ then*

$$r - dt > \frac{n+2}{2q-1}.$$

In particular, if $r = n - t$, then

$$(d+1)t < n - \frac{n+2}{2q-1}.$$

If, additionally, $d = 2$, then

$$3t < n - \frac{n+2}{2q-1}.$$

For example, suppose $A = \mathbb{F}_q$ and $t > 0$ such that $n := 3t + 1 < q$. Then Shamir's scheme gives an $(n, t, 2, n - t)$-arithmetic secret sharing scheme for $\mathbb{F}_q$ over $\mathbb{F}_q$ (use Theorem 12.16). Shamir's scheme with these parameters plays a central role in secure multiparty computation [16, 68]. It follows from the inequality for $r = n - t$ and $d = 2$ above that the ratio $\frac{3t}{n-1} = 1$ is *not* attainable if q is *small* compared to n, i.e., if

$$q < \frac{n+3}{2}.$$

In fact, this inequality has a stronger implication. For fixed q and for n tending to infinity, we see that, asymptotically, this ratio is upper bounded by a constant *strictly smaller than 1.* Thus, it is not even possible to get this ratio arbitrarily close to 1.

DEFINITION 12.32 *For each $n \geq 1$, let $T(n, q)$ denote the largest integer t such that there exists an $(n, t, 2, n - t)$-codex for $\mathbb{F}_q$ over $\mathbb{F}_q$. Then*

$$\widehat{\tau}(q) := \limsup_{n \to \infty} \frac{3 \cdot T(n, q)}{n - 1}.$$

Note that this is well-defined, since $t = 0$ is allowed here and for each $n \geq 1$, there exists a trivial $(n, 0, 2, n)$-codex for $\mathbb{F}_q$ over $\mathbb{F}_q$. [17]

THEOREM 12.33 *[48] For each finite field $\mathbb{F}_q$ it holds that*

$$\widehat{\tau}(q) < 1 - \frac{1}{2q-1} < 1$$

[17] Namely, the code C is the n-fold product of $\mathbb{F}_q$ and the map ψ just projects on its first component.

See [48] also for improvements of this bound.

At this point, it is not clear *at all* whether it holds that $\widehat{\tau}(q) > 0$ for some finite field $\mathbb{F}_q$: in the polynomial codex construction n is bounded as a function of q. But for sure, *if* it does (and we shall see shortly *that* it does), then the rate $\frac{t}{n-1}$ is asymptotically strictly smaller than $1/3$. [18]

To show that positive rates can indeed be achieved is more involved. Before we can do so in Section 12.8, we need to discuss some results on towers of algebraic function fields.

12.7 Towers of Algebraic Function Fields

We give an overview of the basic theory of algebraic function fields and we discuss fundamental asymptotical results on towers of algebraic function fields over finite fields. Our exposition follows mostly Stichtenoth [169], but at some places we follow Neukirch [143] or Rosen [159].

12.7.1 Definition of an Algebraic Function Field

Let K be a field. An *algebraic function field* F/K (in one variable) is an extension field F of K such that F is a finite-degree algebraic extension of $K(x)$, for some $x \in F$ that is transcendental over K. Equivalently, F is finitely generated over K as a field, with transcendence degree 1. Note that the field $K(x)$ is isomorphic to the field of rational functions $K(X)$, i.e., the field of fractions of the polynomial ring $K[X]$. In particular, $|F| = \infty$. Such a transcendental element x can always be chosen such that F is a finite-degree, separable field extension of $K(x)$ if K is perfect. [19] If F/K is an algebraic function field, we always assume that K is the *full field of constants*, i.e., if $z \in F$ is algebraic over K, then $z \in K$.

12.7.2 Valuation Rings

Let F/K be an algebraic function field. Let A be a *valuation ring* of F, i.e., a subring of F such that

- $K \subsetneq A \subsetneq F$.
- For all $f \in F$ with $f \neq 0$, it holds that $f \in A$ or $f^{-1} \in A$.

There are infinitely many valuation rings in F.

The ring A is a PID with a single prime ideal P, its corresponding *place*. Let $t \in A$ be such that $P = tA$, a *uniformizer*. It holds that each nonzero $f \in A$ can be uniquely expressed as $f = ut^m$, where $u \in A$ is a unit and m is a nonnegative

[18] For stronger bounds as well as generalizations, e.g. , for arithmetic secret sharing where the secret space is $\mathbb{F}_q^k$ with k "much larger" than 1, please refer to [47]. The case $k > 1$ additionally exploit classical bounds from the theory of error correcting codes (such as the Griesmer bound).

[19] See [169], Ch. 3, Proposition 3.10.2.

integer. Note that the element t is prime and that the set of uniformizers consists of all $u't$ with $u' \in A$ a unit. The units $U(A)$ of A coincide with the set $A \setminus P$. It is verified at once that P is maximal. Also note that F is the field of fractions of A.

The ring A is determined by P since $f \in A$ if and only if $f^{-1} \notin P$. As a consequence, there is a one-to-one correspondence between the valuation rings of F and the places of F. From now on, we denote the set of places of F by $\mathbb{P}(F)$ and the valuation ring corresponding to $P \in \mathbb{P}(F)$ by $\mathcal{O}_P$.

The valuation ring $\mathcal{O}_P$ gives rise to a *discrete valuation*

$$\vartheta_P : F \longrightarrow \mathbb{Z} \cup \{\infty\}$$

on F. Let $t \in \mathcal{O}_P$ be a uniformizer. For $f \in F$, define $\vartheta_P(f)$ as follows. First, $\vartheta_P(0) = \infty$ by default. Second, if $f \in \mathcal{O}_P$ with $f \neq 0$, define $\vartheta_P(f) = m$, where m is the unique nonnegative integer such that $f = ut^m$ for some unit $u \in \mathcal{O}_P$. Note that $\vartheta_P(f)$ does not depend on the choice of uniformizer. Third, if $f \notin \mathcal{O}_P$, then $f^{-1} \in \mathcal{O}_P$ and define $\vartheta_P(f) = -\vartheta_P(f^{-1})$.

This discrete valuation ϑ_P satisfies the following basic properties. For all $f, g \in F$:

- $\vartheta_P(fg) = \vartheta_P(f) + \vartheta_P(g)$ (*multiplicativity*).
- $\vartheta_P(f+g) \geq \min\{\vartheta_P(f), \vartheta_P(g)\}$ (*triangle inequality*).
- $\vartheta_P(f+g) = \min\{\vartheta_P(f), \vartheta_P(g)\}$ if $\vartheta_P(f) \neq \vartheta_P(g)$ (*strict triangle inequality*).

Note that

$$\mathcal{O}_P = \{f \in F : \vartheta_P(f) \geq 0\},$$

$$P = \{f \in F : \vartheta_P(f) > 0\},$$

$$U(\mathcal{O}_P) = \{f \in F : \vartheta_P(f) = 0\},$$

$$F \setminus \mathcal{O}_P = \{f \in F : \vartheta_P(f) < 0\}.$$

Since $K \subset \mathcal{O}_P$, $K \cap P = \{0\}$, and P is maximal, it follows that $\mathcal{O}_P/P$ is an extension field of K, the *residue class field* of P. The degree of this extension is finite. The *degree of the place* P is defined as

$$\deg P := [\mathcal{O}_P/P : K].$$

Let $f \in F$. If $f \in \mathcal{O}_P$, then

$$f(P) := (f \bmod P) \in \mathcal{O}_P/P,$$

the *evaluation* of f at P. Note that $f(P) = 0$, i.e., f has a *zero* at P, if and only if $f \in P$. If $f \notin \mathcal{O}_P$, then f has a *pole* at P and we write $f(P) = \infty$.

The *rational places* are the places of degree 1. So if P is a rational place and if $f \in \mathcal{O}_P$, then $f(P) \in K$.

If $f \in K$, then $f(P) = f$. Hence, the elements of K are *constants*. There are no other constants, as is implied by the following property.

Let $f \in F$ and $f \notin K$. Then:

- There exists a place $P' \in \mathbb{P}(F)$ such that f has a zero at P', i.e., $f(P') = 0$.
- There exists a place $P'' \in \mathbb{P}(F)$ such that f has a pole at P'', i.e., $f(P'') = \infty$.
- At all places $Q \in \mathbb{P}(F)$, except for a finite number of them, f has neither a pole nor a zero, i.e., $f(Q) \neq 0, \infty$.

More precisely, *counting multiplicities*, f has *as many zeros as poles*:

$$\sum_{P \in \mathbb{P}(F)} \vartheta_P(f) \cdot \deg P = 0.$$

Note that this identity also holds if $f \in K$ with $f \neq 0$ as all valuations are equal to zero for such functions f.

12.7.3 Divisors and the Divisor Class Group

Define $\mathrm{Div}(F)$ as the free abelian group generated by the set $\mathbb{P}(F)$. Its elements are called *divisors*. If $D \in \mathrm{Div}(F)$, then it is represented by a formal sum

$$D = \sum_{P \in \mathbb{P}(F)} \lambda_P \cdot P,$$

such that $\lambda_P \in \mathbb{Z}$ for all $P \in \mathbb{P}(F)$ and $\lambda_P = 0$ except for a finite number of $P \in \mathbb{P}(F)$. The divisor with $\lambda_P = 0$ for all $P \in \mathbb{P}(F)$, i.e., the neutral element of the group $\mathrm{Div}(F)$, is denoted by 0. The *degree* of $D \in \mathrm{Div}(F)$ is defined as

$$\deg D = \sum_{P \in \mathbb{P}(F)} \lambda_P \cdot \deg P \ \in \mathbb{Z}.$$

Note that the map

$$\mathrm{Div}(F) \longrightarrow \mathbb{Z}$$

$$D \mapsto \deg D$$

is a group morphism.

Let $f \in F$ with $f \neq 0$. Then the *divisor of* f is defined as

$$(f) = \sum_{P \in \mathbb{P}(F)} \vartheta_P(f) \cdot P \quad \in \mathrm{Div}(F).$$

Since $\vartheta_P(f) = 0$ for almost all $P \in \mathbb{P}(F)$, this is indeed a divisor. Moreover, it holds that

$$\deg(f) = 0.$$

More precisely, let Z_0 (resp., Z_∞) denote the set of zeros (resp., poles) of f. Define the *zero divisor* (resp., *pole divisor*) of f as

$$(f)_0 = \sum_{P \in Z_0} \vartheta_P(f) \cdot P \quad , \quad (f)_\infty = \sum_{P \in Z_\infty} (-\vartheta_P(f)) \cdot P.$$

Then

$$\deg(f)_0 = \deg(f)_\infty = [F : K(f)] < \infty.$$

The subgroup of $\mathrm{Div}(F)$ consisting of all divisors with degree 0 is denoted $\mathrm{Div}_0(F)$. The set of all divisors (f), the *principal divisors*, is a subgroup of $\mathrm{Div}_0(F)$ and it is denoted by $\mathrm{Prin}(F)$. The (degree-zero) *Picard group* [20] of F is the quotient group

$$\mathrm{Cl}(F) := \mathrm{Div}_0(F)/\,\mathrm{Prin}(F).$$

Its order is called the *class number* of F.

A divisor $D = \sum_{P \in \mathbb{P}(F)} \lambda_P \cdot P \in \mathrm{Div}(F)$ is *effective* if $\lambda_P \geq 0$ for all $P \in \mathbb{P}(F)$. If $D' \in \mathrm{Div}(F)$, then we write $D \geq D'$ if $D - D' \in \mathrm{Div}(F)$ is effective. In this notation, $D \in \mathrm{Div}(F)$ is effective if $D \geq 0$, where $0 \in \mathrm{Div}(F)$. The *support* of D is the finite set $\mathrm{supp}(D)$ consisting of all $P \in \mathbb{P}(F)$ such that $\lambda_P \neq 0$.

12.7.4 The Weak Approximation Theorem

The *Weak Approximation Theorem* states the following.

THEOREM 12.34 *Let F/K be an algebraic function field. Let $P_1, \ldots, P_m \in \mathbb{P}(F)$ be distinct places. Let $f_1, \ldots, f_m \in F$ and $r_1, \ldots, r_m \in \mathbb{Z}$. Then there exists $f \in F$ such that*

$$\vartheta_{P_1}(f - f_1) = r_1, \ldots, \vartheta_{P_m}(f - f_m) = r_m.$$

In a sense, this is a “Chinese Remainder Theorem” for algebraic function fields.

The *Strong Approximation Theorem* states that if S is a *proper* subset $S \subsetneq \mathbb{P}(F)$ such that $P_1, \ldots, P_m \in S$, then we can *additionally* enforce $\vartheta_P(f) \geq 0$ for all $P \in S \setminus \{P_1, \ldots, P_m\}$.

12.7.5 The Riemann-Roch Theorem

Let $D \in \mathrm{Div}(F)$. Define

$$\mathcal{L}(D) = \{f \in F \setminus \{0\} : (f) + D \geq 0\} \cup \{0\},$$

the *Riemann-Roch space* of D. Note that $\mathcal{L}(D)$ is a K-vector space. Define

$$\ell(D) := \dim_K(\mathcal{L}(D)).$$

Elementary properties of Riemann-Roch dimension include the following.

[20] or (degree-zero) *divisor class group*

- $\ell(D) = 0$ if $\deg D < 0$.
- If $D' \geq D$, then $\ell(D') \geq \ell(D)$.
- For all $f \in \mathcal{L}(D)$, it holds that $f(P) \neq \infty$ for each $P \notin \operatorname{supp}(D)$.

The first follows at once. Indeed, if $f \in \mathcal{L}(D)$ with $f \neq 0$, then $0 \leq \deg((f)+D) = \deg(f) + \deg D = \deg D < 0$, a contradiction. The second and third follow from the definitions.

Divisors $D_0, D_1 \in \operatorname{Div}(F)$ are *linearly equivalent*, denoted $D_0 \sim D_1$, if

$$D_1 = (f) + D_0$$

for some $f \in F$ with $f \neq 0$. If $D_0 \sim D_1$, then $\deg D_0 = \deg D_1$ and $\mathcal{L}(D_0) \simeq \mathcal{L}(D_1)$. In particular, $\ell(D_0) = \ell(D_1)$. Each divisor $D \in \operatorname{Div}(F)$ with $\ell(D) > 0$ is linearly equivalent to some effective divisor. Namely, let $f \in \mathcal{L}(D)$ with $f \neq 0$. Then $(f) + D \geq 0$ by definition.

It holds that

$$\ell(D) \leq \deg D + 1$$

if $\deg D \geq 0$. We briefly sketch the proof of this claim. Under the stated condition the claim is true in either of the cases $\ell(D) = 0$ or $D = 0$. Consider the case $\ell(D) > 0$ and $D \geq 0$. It is not hard to show that if $D' \in \operatorname{Div}(F)$ and if $P \in \mathbb{P}(F)$, then $\ell(D' + P) \leq \ell(D') + \deg P$. [21] Applying induction, the claim follows in this case. For general $D \in \operatorname{Div}(F)$ with $\ell(D) > 0$, apply the latter to some effective divisor that is linearly equivalent to D.

Consider the "defect" $\deg D + 1 - \ell(D)$. By the discussion above, this is non-negative if $\deg D \geq 0$. Clearly, it is not positive if $\deg D < 0$.

Definition 12.35 (Genus) *Let F/K be an algebraic function field. The* genus *of F is defined as*

$$g(F) := \max_{D \in \operatorname{Div}(F)} \{\deg D + 1 - \ell(D)\}.$$

So far, it's only clear that $0 \leq g(F) \leq \infty$. *Riemann's Inequality* essentially shows that $g(F) < \infty$. The latter fact, in combination with the definition of the genus, immediately gives a nontrivial lower bound on Riemann-Roch dimension of divisors with degree at least $g(F)$.

Theorem 12.36 (Riemann's Inequality) *Let F/K be an algebraic function field. Then $0 \leq g < \infty$, where g is the genus of F. Consequently, for all $D \in \operatorname{Div}(F)$ it holds that*

$$\ell(D) \geq \deg D + 1 - g.$$

[21] See [169], Ch. 1, Lemma 1.4.8, p. 18.

The classical *Riemann-Roch Theorem* [22] turns Riemann's Inequality into an equality for all $D \in \mathrm{Div}(F)$ by adding the "missing term" into the inequality.

A *canonical divisor* is a divisor $W \in \mathrm{Div}(F)$ such that

- $\deg W = 2g - 2$
- $\ell(W) \geq g$,

which exists. [23]

The Riemann-Roch Theorem states the following.

THEOREM 12.37 (RIEMANN-ROCH) *Let F/K be an algebraic function field. Let $W \in \mathrm{Div}(F)$ be a canonical divisor. For all $D \in \mathrm{Div}(F)$ it holds that*

$$\ell(D) = \deg D + 1 - g + \ell(W - D),$$

where g is the genus of F.

It follows in particular that $\ell(W) = g$ by setting $D = W$.

COROLLARY 12.38 *For all $D \in \mathrm{Div}(F)$ with $\deg D > 2g - 2$, it holds that*

$$\ell(D) = \deg D + 1 - g.$$

The simplest example of an algebraic function field is the rational function field $K(X)$, i.e., the field of fractions of the polynomial ring $K[X]$. Its elements are represented as $a(X)/b(X) \in K(X)$ with $a(X), b(X) \in K[X]$ relatively prime and $b(X)$ nonzero. It is easy to describe the places and valuations. Let $P(X) \in K[X]$ be a polynomial of degree $d \geq 1$ and suppose it is irreducible.

Let $f \in K(X)$. Suppose $f \neq 0$ and write $f = (a(X)/b(X))P(X)^m$ such that $P(X)$ divides neither $a(X)$ nor $b(X)$ as polynomials, with $a(X), b(X) \in K[X]$ relatively prime and $m \in \mathbb{Z}$. Then $f \in \mathcal{O}_P$, the valuation ring corresponding to $P(X)$, if and only if $m \geq 0$. Furthermore, $f \in P$, where P is the place corresponding to $\mathcal{O}_P$, if and only if $m > 0$. Finally, f is a unit in $\mathcal{O}_P$ if $m = 0$. The polynomial $P(X)$ is a uniformizer and the valuation $\vartheta_P(f)$ equals m.

The ring $\mathcal{O}_P$ is the localization of $K[X]$ at the maximal ideal $(P(X))$, i.e., the ring consisting of the elements of $K(X)$ with denominator not divisible by $P(X)$. Moreover, $K[X]/(P(X))$ is an extension field of K of degree d and it is isomorphic to $\mathcal{O}_P/P$ via the assignment $\overline{f(X)} \mapsto f(X)(P)$. If $P(X) = X - z$ ($z \in K$), then P is a rational place and $a(X)/b(X) \in \mathcal{O}_P$ evaluated at P is just $a(z)/b(z)$.

This accounts for all places, except one: the *infinite place* P_∞. Let $f \in K(X)$. Suppose $f \neq 0$ and write $f = a(X)/b(X)$. Then $f \in \mathcal{O}_\infty$ if and only if $\deg b(X) \geq \deg a(X)$. Furthermore, $f \in P_\infty$ if and only if $\deg b(X) > \deg a(X)$. Finally, f is a

[22] See [169], Ch. 1, par. 5, pp. 24–31.
[23] See [169], Sections 1.5 and 1.6.

unit if these degrees are equal. The rational function $1/X$ is a uniformizer of $\mathcal{O}_\infty$. The valuation $\vartheta_{P_\infty}(f)$ equals $\deg b(X) - \deg a(X)$. If $f \in \mathcal{O}_\infty$ then $f(P_\infty) = 0$ if $f \in P_\infty$ and $f(P_\infty) = a_k/b_k$ otherwise, where a_k (resp., b_k) is the leading coefficient of $a(X)$ (resp., $b(X)$). In particular, P_∞ is a rational place.

The genus of the rational function field equals 0. First, it holds that $\ell(m \cdot P_\infty) \geq m+1$, for any integer $m \geq 0$. Indeed, $1, X, \ldots, X^m \in \ell(m \cdot P_\infty)$. Second, by the Riemann-Roch Theorem, this gives $m+1 \leq \ell(m \cdot P_\infty) = \deg(m \cdot P_\infty) + 1 - g = m + 1 - g$, for large enough m. Thus, $g = 0$.

Note that if $K = \mathbb{F}_q$ then there are exactly $q+1$ rational places in the rational function field over $\mathbb{F}_q$.

12.7.6 The Hasse-Weil Theorem

From here on, $K = \mathbb{F}_q$.

Let $N(F)$ be the number of rational places of F. For each nonnegative integer n define

$$A_n = |\{D \in \mathrm{Div}(F) \mid D \geq 0 \text{ and } \deg D = n\}|.$$

It holds that $A_n < \infty$ for each $n \geq 0$. The *Zeta function* of $F/\mathbb{F}_q$ is the power series

$$Z(t) = \sum_{n=0}^{\infty} A_n t^n \in \mathbb{C}[[t]],$$

which converges for $|t| < \frac{1}{q}$. The *L-polynomial* is defined as

$$L(t) := (1-t)(1-qt)Z(t) \in \mathbb{C}[[t]].$$

It can be shown that

$$L(t) \in \mathbb{Z}[t] \text{ and } \deg L(t) = 2g.$$

Writing

$$L(t) = a_0 + a_1 t + \cdots + a_{2g} t^{2g},$$

it follows that

$$a_1 = N - (q+1),$$

where $N := N(F)$. Now, $L(t)$ is identical to 1 if $g = 0$ and otherwise it factors in $\mathbb{C}[t]$ as

$$L(t) = \prod_{i=1}^{2g} (1 - \alpha_i t),$$

where $\alpha_1, \ldots, \alpha_{2g} \in \mathbb{C}$ are of course algebraic integers. Hence,

$$N = q + 1 + \sum_{i=1}^{2g} \alpha_i.$$

The celebrated *Hasse-Weil Theorem*, [24] also known as the *Riemann Hypothesis for Function Fields*, gives the absolute values of the α_i's.

THEOREM 12.39 (HASSE-WEIL) *Let $F/\mathbb{F}_q$ be an algebraic function field. Let g denote its genus. Suppose $g \geq 1$. Then the $2g$ roots $\alpha_1, \ldots, \alpha_{2g}$ of the L-polynomial $L(t) \in \mathbb{Z}[t]$ satisfy*

$$|\alpha_i| = \sqrt{q}$$

for $i = 1, \ldots, 2g$.

An upper bound on the number of rational places in F follows at once.

THEOREM 12.40 (HASSE-WEIL BOUND) *Let $F/\mathbb{F}_q$ be an algebraic function field. Let g denote its genus. Then*

$$N \leq q + 1 + 2g\sqrt{q},$$

where N denotes the number of rational places of F.

In particular, the number of rational places of $F/\mathbb{F}_q$ is bounded as a function of q and g. Actually, it holds that

$$|N - (q+1)| \leq 2g\sqrt{q},$$

which gives the additional information that if g is *small* compared to q, there cannot be too few rational places.

Serre improved this bound as follows:

THEOREM 12.41 (SERRE) *Notation being as above,*

$$|N - (q+1)| \leq g\lfloor 2\sqrt{q} \rfloor.$$

12.7.7 The Drinfeld-Vladuts Bound and Ihara's Theorem

For each finite field $\mathbb{F}_q$ and for each integer $g \geq 0$, define $N_q(g)$ as the maximum of the number of rational places $N(F)$ where F ranges over all the algebraic function fields whose full field of constants is $\mathbb{F}_q$ and whose genus is g.

DEFINITION 12.42 (IHARA'S CONSTANT)

$$A(q) := \limsup_{g\to\infty} \frac{N_q(g)}{g}$$

where $g > 0$.

[24] See [169], Ch. 5, par. 2, pp. 197-212, Theorem 5.2.1.

Note that

$$0 \leq A(q) < \infty,$$

where the latter inequality follows from the Hasse-Weil Bound.

Note that $A(q) \geq \ell$ for some nonnegative real number ℓ if and only if, for each real number $\epsilon > 0$, there is an infinite family of algebraic functions fields $\{F_i/\mathbb{F}_q\}_{i=0}^{\infty}$ such that

1. The sequence $g_0, g_1, g_2, \ldots$ is strictly increasing, where $g_i := g(F_i)$ for all $i \geq 0$.
2. For all $i \geq 0$, it holds that $N_i/g_i \geq (1-\epsilon)\cdot \ell$, where $N_i := N(F_i)$.

The Serre Bound implies $A(q) \leq \lfloor 2\sqrt{q} \rfloor$. This is improved as follows.

THEOREM 12.43 (DRINFELD-VLADUTS BOUND) *Let $\mathbb{F}_q$ be a finite field. Then*

$$A(q) \leq \sqrt{q} - 1.$$

It is a priori not clear at all whether $A(q) > 0$ for some finite field $\mathbb{F}_q$. Using the theory of modular curves, Ihara [113] showed the following result.

THEOREM 12.44 (IHARA) *Let $\mathbb{F}_q$ be a finite field such that q is a square. Then*

$$A(q) \geq \sqrt{q} - 1.$$

Combining this with the Drinfeld-Vladuts Bound we get

THEOREM 12.45 *Let $\mathbb{F}_q$ be a finite field such that q is a square. Then*

$$A(q) = \sqrt{q} - 1,$$

which is optimal.

In 1982 Tsfasman, Vladuts and Zink [172] built on this result to show that *error correcting codes* exist that exceed the *asymptotic Gilbert-Varshamov bound.* This has spawned substantial research into algebraic function fields with many rational points, which continues to date.

12.7.8 Garcia-Stichtenoth's Explicit Optimal Towers

Garcia and Stichtenoth [98, 99] were first to present an *explicit* construction that shows $A(q) = \sqrt{q} - 1$ when q is square, using *recursive towers.*

It is beyond the scope here to give the full proof. Yet, we wish to give a taste of it by a high-level overview. We restrict our attention to an overview of an explicit proof of the weaker result that $A(q) > 0$ for some finite fields $\mathbb{F}_q$. Their general (and optimal) result is more involved. [25]

[25] See Ch. 1 of [100] for a more advanced overview. Our overview is mostly based on the detailed proof from "first principles" in [169]. For an overview of a nonconstructive proof based on the theory of modular forms, see [141]. See also [144, 173].

A *tower* over $\mathbb{F}_q$ is an infinite sequence $\mathcal{F} = (F_0, F_1, F_2, \ldots)$ of algebraic function fields $F_i/\mathbb{F}_q$ such that the following holds.

- $F_0 \subsetneq F_1 \subsetneq F_2 \subsetneq \ldots \subsetneq F_m \subsetneq \ldots$.
- Each field extension F_{i+1} of F_i is finite and separable $(i = 0, 1, 2, \ldots)$.
- The genera $g(F_i)$ tend to infinity when i tends to infinity.

Recall that, by our definition, the full field of constants of $F_i/\mathbb{F}_q$ equals $\mathbb{F}_q$ for all $i \geq 0$.

For *any* such tower, the following holds. Define the *(Ihara) limit* of the tower as

$$\lambda(\mathcal{F}) := \lim_{i\to\infty} \frac{N(F_i)}{g(F_i)},$$

define the *splitting rate* of the tower as

$$\nu(\mathcal{F}) := \lim_{i\to\infty} \frac{N(F_i)}{[F_i : F_0]},$$

and the *genus* of the tower as

$$\gamma(\mathcal{F}) := \lim_{i\to\infty} \frac{g(F_i)}{[F_i : F_0]}.$$

These limits are well-defined. Moreover,

$$\lambda(\mathcal{F}) \in \mathbb{R}_{\geq 0}\ ,\ \nu(\mathcal{F}) \in \mathbb{R}_{\geq 0}\ ,\ \gamma(\mathcal{F}) \in \mathbb{R}_{>0} \cup \{\infty\}$$

and

$$\lambda(\mathcal{F}) = \frac{\nu(\mathcal{F})}{\gamma(\mathcal{F})},$$

with

$$0 \leq \lambda(\mathcal{F}) \leq A(q).$$

Therefore, in order to achieve $\lambda(\mathcal{F}) > 0$, it is necessary and sufficient that

$$0 < \nu(\mathcal{F}), \gamma(\mathcal{F}) < \infty.$$

In particular, splitting and genus can be treated separately.

Before continuing, we discuss some very helpful basic facts about extensions of algebraic function fields. Along the way, we point out some of the well-known strong similarities with algebraic number fields. [26]

Let $F/\mathbb{F}_q$ and $F'/\mathbb{F}_q$ be algebraic function fields such that F' is a finite-degree separable extension of F. Write $n := [F' : F]$. There is the following relation between the places of F' and those of F. Each place P of F' yields a place of F when intersected with F. If $P \cap F = \wp$ we say that the place P of F' *lies above*

[26] For a in-depth treatment of the similarities between algebraic function fields and algebraic number fields, see [143, 159].

the place $\wp$ of F. Moreover, for each place of F, there is a place of F' that lies above it and there are at most n of them.

The places of F' lying above a place of F obey the *Fundamental Identity.* Let $\wp$ be a place of F, with corresponding valuation ring $\mathfrak{o}_{\wp}$. Let P be a place of F', with corresponding valuation ring $\mathcal{O}_P$. Suppose P lies above $\wp$.

The *relative degree* f of P with respect to $\wp$ is the positive integer defined as

$$f(P|\wp) := [\mathcal{O}_P/P : \mathfrak{o}_{\wp}/\wp].$$

Note that $\mathcal{O}_P/P$ is a finite-degree extension of $\mathbb{F}_q$ that contains $\mathfrak{o}_{\wp}/\wp$ as an intermediate extension; so this is well-defined.

The *ramification index of* $\wp$ *at* P is the positive integer defined as

$$e(P|\wp) = \vartheta_P(t),$$

where ϑ_P is the valuation in F' induced by P and $t \in \mathfrak{o}_{\wp}$ is any uniformizer of $\mathfrak{o}_{\wp}$. Note that $e(P|\wp) = 1$ if and only if t is not only a uniformizer of $\mathfrak{o}_{\wp}$ but also of $\mathcal{O}_P$.

Let $P_1, \ldots, P_k$ be the set of places of F' lying above the place $\wp$ of F. Then the Fundamental Identity states that

$$e_1 f_1 + \cdots + e_k f_k = n,$$

where $e_j = e(P_j|\wp)$ and $f_j = f(P_j|\wp)$ for $j = 1, \ldots, k$.

We say that $\wp$ *splits completely* in F' if there are exactly n places P' above $\wp$. On account of the Fundamental Identity each of these has relative degree $f(P'|\wp) = 1$.

Kummer's Theorem [27] implies the following special case. Let $z \in F'$ be such that $F' = F(z)$ and consider the minimal polynomial $f(X) \in F[X]$ of z over F. Suppose $f(X) \in \mathfrak{o}_{\wp}[X]$, i.e., z is integral over $\mathfrak{o}_{\wp}$. If $f(X)$ is reduced modulo $\wp$ and its factorization in $(\mathfrak{o}_{\wp}/\wp)[X]$ is square-free, [28] then $\wp$ is unramified and there is bijection between the factors and the places above $\wp$. Moreover, for each factor the corresponding place above $\wp$ has relative degree equal to the degree of the factor. In particular, if it factors in distinct linear terms, there are precisely n distinct places above $\wp$ and each of those is of relative degree 1. Therefore, this gives a sufficient criterion for complete splitting.

We say that $\wp$ *ramifies in* F' *at* P if $e(P|\wp) > 1$. If $\wp$ ramifies at P and if p, the characteristic of $\mathbb{F}_q$, divides $e(P|\wp)$, then we say that $\wp$ *wildly ramifies at* P. If $\wp$ ramifies at P and if p does not divide $e(P|\wp)$, then we say it *tamely ramifies at* P.

We say that $\wp$ is *ramified* in F' if it is ramified at some place of F' lying above it. If $\wp$ is not ramified in F', we say that $\wp$ is *unramified* in F'. If no place of F

[27] See e.g. [169], Ch. 3, par. 3, pp. 86–90, Theorem 3.3.7.
[28] Note that this is just factorization in a polynomial ring over a finite extension field of $\mathbb{F}_q$.

ramifies wildly in F', then the extension F'/F is *tame.* Otherwise it is *wild.* A tower is tame if each of the extensions F_i/F_0 is tame. It is wild if it is not tame.

It is useful to develop a different perspective on these definitions. The ring $\mathfrak{o}_\wp$ is a Dedekind domain with F as its field of fractions. Let $\mathcal{O}'$ denote its integral closure in F', i.e., the subring of F' consisting of those elements $x \in F'$ such that the minimal polynomial $f(X) \in F[X]$ of x over F satisfies $f(X) \in \mathfrak{o}_\wp[X]$. This is also a Dedekind domain, with F' as its field of fractions. Moreover, $\mathcal{O}'$ is a free $\mathfrak{o}_\wp$-module of rank n.

Consider the Dedekind-factorization of the ideal $\wp \cdot \mathcal{O}'$ of $\mathcal{O}'$. There is a bijection between the prime ideals of $\mathcal{O}'$ occurring in this factorization and the places P of F' lying above the place $\wp$ of F. Namely, in one direction, the localization of $\mathcal{O}'$ at a prime ideal of $\mathcal{O}'$ that appears in the Dedekind-factorization of the ideal $\wp \cdot \mathcal{O}'$ gives a valuation ring $\mathcal{O}_P$ of F' such that P lies above $\wp$. In the other direction, the intersection with $\mathcal{O}'$ of a valuation ring $\mathcal{O}_P$ of F' such that P lies above $\wp$ gives a prime ideal of $\mathcal{O}'$ that appears in the Dedekind-factorization of the ideal $\wp \cdot \mathcal{O}'$. Moreover, the relative degree of a place P of F' lying above the place $\wp$ of F is the relative degree of the prime ideal $I(P) := \mathcal{O}_P \cap \mathcal{O}'$ of $\mathcal{O}'$ lying above $\wp$, i.e., it equals the degree of the field extension $\mathcal{O}'/I(P) \supset \mathfrak{o}_\wp/\wp$. As an aside, this way one sees that the Fundamental Identity for algebraic function fields over $\mathbb{F}_q$ is in essence the same as the corresponding one for algebraic number fields.

This perspective allows us to describe which places of F ramify and at which places of F' this happens (and how badly!), as this is all captured by the *different.* Since F' is a finite separable extension of F by assumption, the trace $\mathrm{Tr}_{F'/F}$ is nontrivial. As remarked before, this implies that the dual space of F' (as an F-vector space), i.e., the space of F-linear forms $F' \longrightarrow F$ coincides with the F-vector space of maps $\phi_a : F' \longrightarrow F$ defined by the assignment $x \mapsto \mathrm{Tr}_{F'/F}(ax)$ where $a \in F'$.

Now consider the set of elements $a \in F'$ such that the F-linear form ϕ_a associated with a according to this correspondence satisfies $\phi_a(\mathcal{O}') \subset \mathfrak{o}_\wp$. This set happens to constitute a fractional ideal $\mathcal{C}_\wp^{-1}$ of $\mathcal{O}'$, containing $\mathcal{O}'$. If we now invert this fractional ideal in the group of fractional ideals of $\mathcal{O}'$ we get an ideal $\mathcal{C}_\wp$ of $\mathcal{O}'$. Its Dedekind-factorization is nontrivial if and only if $\wp$ ramifies in F'. If it ramifies, this factorization gives a bijection with the places of F' lying above $\wp$ at which $\wp$ ramifies.

For example, if $\mathcal{O}'$ is a simple extension of $\mathfrak{o}_\wp$, i.e., $\mathcal{O}' = \mathfrak{o}_\wp[z]$ for some $z \in \mathcal{O}'$, then $\mathcal{C}_\wp$ is very easy to describe. If $f(X) \in \mathfrak{o}_\wp[X]$ denotes the minimal polynomial of z over F, then $\mathcal{C}_\wp$ is the ideal $f'(z) \cdot \mathcal{O}'$ of $\mathcal{O}'$, where $f'(X) \in \mathfrak{o}_\wp[X]$ denotes the formal derivative of $f(X)$. [29]

The *Dedekind Different Theorem* gives more information about the factorization of $\mathcal{C}_\wp$. As before, let P be a place of F' above the place $\wp$ of F. If this factorization is nontrivial, i.e., $\mathcal{C}_\wp \neq \mathcal{O}'$, and if P corresponds to some prime ideal

[29] See e.g. [143], Ch. 3, par. 2, p. 197, Proposition 2.4.

in its Dedekind-factorization (i.e., $e(P|\wp) > 1$), denote its exponent in this factorization by $d(P|\wp)$, the *different exponent of $\wp$ at P*. Then $d(P|\wp) = e(P|\wp) - 1$ if and only if $\wp$ ramifies tamely at P. If it ramifies wildly, then $d(P|\wp) > e(P|\wp) - 1$. If $\wp$ does not ramify at P, define $d(P|\wp) = 0$ $(= e(P|\wp) - 1)$.

At most finitely many places of F ramify in F'. We sketch a proof of this claim. Fix an arbitrary F-basis $z_1, \ldots, z_n$ of F'. If $z_1, \ldots, z_n \in \mathcal{O}'$, then it can be shown that

$$\sum \mathfrak{o}_\wp \cdot z_i \subset \mathcal{O}' \subset \sum \mathfrak{o}_\wp \cdot z_i^*,$$

where $z_1^*, \ldots, z_n^*$ denotes the dual basis. If, additionally, $z_1^*, \ldots, z_n^* \in \mathcal{O}'$, then it follows directly that this basis as well as its dual constitutes a basis of the free $\mathfrak{o}_\wp$-module $\mathcal{O}'$. The claim now follows from two further facts. First, it can be shown that the dual basis of a basis of the free $\mathfrak{o}_\wp$-module $\mathcal{O}'$ generates $\mathcal{C}_\wp^{-1}$ as $\mathfrak{o}_\wp$-module. Therefore, under the conditions above,

$$\mathcal{C}_\wp^{-1} = \mathcal{C}_\wp = \mathcal{O}'$$

and $\wp$ is unramified in F'. Second, the conditions above are satisfied for all except finitely many valuations rings of F. This follows immediately by inspection of the coefficients of the minimal polynomials in $F[X]$ of each of the elements in this basis and in its dual.

The claim can also be verified using Kummer's Theorem. Let $z \in F'$ be such that $F' = F(z)$ and consider the minimal polynomial $f(X) \in F[X]$ of z over F. The discriminant of $1, z, \ldots, z^{n-1}$ with respect to F'/F equals $N_{F'/F}(f'(z)) \in F$ (up to a sign), where $f'(X) \in F[X]$ is the formal derivative of $f(X)$. Since z is integral over almost all valuation rings of F and since for almost all of those this discriminant is not contained in the corresponding place $\wp$, it follows that the factorization of (the class of) $f(X)$ in $(\mathfrak{o}_\wp/\wp)[X]$ is square-free for each of those places $\wp$. The claim follows.

Define the *different* as

$$\mathrm{Diff}(F'/F) := \sum_{P|\wp} d(P|\wp) \cdot P \quad \in \mathrm{Div}(F'),$$

where $\wp$ runs through all places of F and P runs through all places of F' above $\wp$. Note that this is indeed a divisor: at most finitely many places of F ramify in F' and $d(P|\wp) > 0$ if and only if $\wp$ ramifies at P.

If the extension F'/F is tame, then

$$\mathrm{Diff}(F'/F) = \sum_{P|\wp} (e(P|\wp) - 1) \cdot P \quad \in \mathrm{Div}(F').$$

The different plays a crucial role in the *Hurwitz Genus Formula*. This basically expresses that the genus evolves linearly in $[F' : F]$ and in the genus $g(F)$, except for an additive "error-term" caused by the different. Precisely, it says that

$$2g(F') - 2 = (2g(F) - 2)[F' : F] + \deg \mathrm{Diff}(F'/F).$$

For example, note that if F is the rational function field over $\mathbb{F}_q$ (which has genus 0), then F'/F is ramified. It also follows that in order to show that the genus tends to infinity in a proposed tower, it is sufficient to identify some F_i with $g(F_i) \geq 2$.

Note that application of the Dedekind Different Theorem gives the following lower bound:

$$2g(F') - 2 \geq (2g(F) - 2)[F' : F] + \deg \left(\sum_{P|\wp} (e(P|\wp) - 1) \cdot P \right),$$

with equality if and only if the extension F'/F is tame.

From the Hurwitz Genus Formula it is clear that in order for the genus of the tower to satisfy $\gamma(\mathcal{F}) < \infty$, the ramification in the tower needs to be controlled. A neat trick towards this end uses the Fundamental Identity in combination with the Hurwitz Genus Formula to show it suffices that the number of places $\wp$ of F_0 that ramify in *some* extension F_i in the infinite tower is finite (i.e., there is a "finite ramification locus") and that no such $\wp$ ramifies wildly.

There is also a version for the wild case, where in addition to a finite ramification locus, it is required that for each $\wp$ in the finite ramification locus, all different exponents $d(P|\wp)$ occurring in the tower are upper bounded by $a_\wp \cdot e(P|\wp)$, with $a_\wp \in \mathbb{R}$ a constant depending on $\wp$. [30] It follows that

$$\gamma(\mathcal{F}) \leq g(F_0) - 1 + \frac{1}{2} \sum_{\wp} a_\wp \cdot \deg \wp < \infty,$$

where $\wp$ runs through the finite ramification locus. In the tame case, of course, all $a_\wp$'s can be taken equal to 1.

In order for the splitting rate to satisfy $\nu(\mathcal{F}) > 0$, it is sufficient that its *splitting locus* is nonempty. This is the set of *rational* places $\wp$ of F_0 such that $\wp$ *splits completely* in *each* extension F_i of F_0, i.e., for each such extension F_i there are exactly $[F_i : F_0]$ places above $\wp$. These must all be rational on account of the Fundamental Identity. It follows that

$$\nu(\mathcal{F}) \geq s > 0,$$

where $s > 0$ is the cardinality of the splitting locus. Putting this together, it follows that

$$\lambda(\mathcal{F}) = \frac{\nu(\mathcal{F})}{\gamma(\mathcal{F})} > 0.$$

So it remains to enforce the sufficient conditions, on the ramification locus on the one hand and on the splitting locus on the other hand.

A main handle to achieve this is to consider *recursively defined* towers. Let $f(Y) \in \mathbb{F}_q(Y)$ and $h(X) \in \mathbb{F}_q(X)$ be non-constant rational functions. Write

[30] See [169], Ch. 7.2, p. 249, Theorem 7.2.10.

$f(Y) = f_0(Y)/f_1(Y)$ with $f_0(Y), f_1(Y) \in \mathbb{F}_q[Y]$ with $f_0(Y), f_1(Y)$ relatively prime. Define

$$\deg f(Y) = \max\{\deg f_0(Y), \deg f_1(Y)\}.$$

Let $\mathcal{F} = (F_0, F_1, F_2, \ldots)$ be a sequence of algebraic functions fields over $\mathbb{F}_q$. Then $\mathcal{F}$ is *recursively defined* by the equation

$$f(Y) = h(X)$$

if there are elements $x_i \in F_i$ for all $i \geq 0$ such that

- x_0 is transcendental over $\mathbb{F}_q$ and $F_0 = \mathbb{F}_q(x_0)$.
- For all $i \geq 0$, it holds that $F_{i+1} = \mathbb{F}_q(x_0, \ldots, x_i, x_{i+1})$, or equivalently, $F_{i+1} = F_i(x_{i+1})$.
- For all $i \geq 0$, it holds that $f(x_{i+1})$, $h(x_i)$ are well-defined and satisfy $f(x_{i+1}) = h(x_i)$.
- $[F_1 : F_0] = \deg f(Y)$.

Note that $x_1, x_2, \ldots$ may be selected in an algebraic closure of $\mathbb{F}_q(x_0)$. For $i \geq 0$, define

$$\widehat{f_i}(Y) := f_0(Y) - h(x_i) f_1(Y) \in F_i[Y].$$

The condition $[F_1 : F_0] = \deg f(Y)$ on the first step in the tower (which will play a distinguished role later on) is equivalent to the condition that the polynomial $\widehat{f_0}(Y)$ is irreducible in the polynomial ring $\mathbb{F}_q(x_0)[Y]$. It follows that $[F_{i+1} : F_i] \leq \deg f(Y)$ for all $i \geq 1$, as $\widehat{f_i}(Y)$ may no longer be irreducible in some $F_i[Y]$.

The first question is which recursively defined sequences constitute a tower. Separability, which is required for each extension $F_i \subset F_{i+1}$, is easy: $\widehat{f_i}(Y) := f_0(Y) - h(x_i) f_1(Y) \in F_i[Y]$ has no multiple roots if $f_0(Y), f_1(Y) \notin \mathbb{F}_q(Y^p)$, where p is the characteristic of $\mathbb{F}_q$, i.e., $f(Y) \notin \mathbb{F}_q(Y^p)$. For example, if $\deg f(Y) < p$, then this is automatically satisfied ($f(Y)$ is not constant!).

The conditions that each extension $F_i \subset F_{i+1}$ is strict and that for each F_i the full field of constants is $\mathbb{F}_q$ can be enforced with a single condition. Namely, that for each $i \geq 0$, there exists a place $P \in \mathbb{P}(F_i)$ and a place $Q \in \mathbb{P}(F_{i+1})$ lying above it such that $e(Q|P) = [F_{i+1} : F_i] > 1$. This is easy to justify. The required strict inclusion already follows from $e(Q|P) > 1$, by the Fundamental Identity. By basic theory of constant field extensions, [31] the relative degree $f(Q|P)$ must satisfy $f(Q|P) > 1$ if the field of constants of F_{i+1} is a strict extension of $\mathbb{F}_q$. However, the ramification degree $e(Q|P)$ "eats up" the entire degree of the extension, so there is "no room" in the Fundamental Identity for this to happen.

Finally, the condition that the genus tends to infinity can often be shown using either of the following two handles. If the number of rational points in the tower tends to infinity, then the genus must also tend to infinity on account of the Hasse-Weil Bound. Another option is to show that $g(F_i) \geq 2$ for some $i \geq 0$ and

[31] See [169], Theorem 3.6.3, p. 114.

to draw the desired conclusion by successive applications of the Hurwitz Genus Formula.

A pivotal aspect of recursive towers is the following observation. For all $i \geq 0$ it holds that:

- $\mathbb{F}_q(x_i, x_{i+1}) \simeq \mathbb{F}_q(x_0, x_1) = F_1$ and $\mathbb{F}_q(x_i) \simeq \mathbb{F}_q(x_0) = F_0$.
- There is the parallelogram of inclusions where F_{i+1} is the compositum of F_i and $\mathbb{F}_q(x_i, x_{i+1})$ and where the rational function field $\mathbb{F}_q(x_i)$ is a subfield of each of the latter two.

A main reason why this is so useful, is that it opens the door to reducing the issues about splitting locus and ramification locus of the tower, by means of inductive arguments, to issues about the extension F_1 of the rational function field F_0 *at the bottom of the tower*, which is much easier to analyze. To this end, various "parallelogram-laws" come in handy.

It is beyond the scope to detail all of this here. But we do give a flavor, as follows. Let us view the parallelogram such that F_{i+1} sits in the north corner, F_i in the west, $\mathbb{F}_q(x_i, x_{i+1})$ in the east and the rational function field $\mathbb{F}_q(x_i)$ in the south. It makes sense to speak of the lower-left extension, the upper-right extension, etc. The main diagonal is the north viewed as an extension of the south.

Abhyankar's Lemma, for instance, implies that if a place of the west ramifies in the north, then the place of the south it lies above ramifies in the east; this is an upper-left versus lower-right connection. From Galois theory it follows that if a place of the south splits completely in the east *and* in the west, then it splits completely in the north. This is a lower-left + lower-right versus main diagonal connection. Note that Kummer's Theorem, mentioned earlier, implies a criterion for complete splitting in terms of factorization of polynomials over finite extension fields of $\mathbb{F}_q$.

Facts such as the ones discussed above make it possible to reduce the delicate selection of the polynomials $f(Y)$ and $h(X)$ required to enforce a nontrivial splitting locus *and* a finite ramification locus to establishing a particular, favorable situation just in the extension F_1/F_0 at the bottom of the tower, which is much more manageable. Besides, the resulting conditions on $f(Y), h(X)$ are typically easily stated in terms of basic algebra and are oblivious of the towers once stated. [32]

Here is an example of a tower crafted in this way. For a finite field $\mathbb{F}_q$ with q square and $\ell := \sqrt{q} > 2$, set

$$Y^{\ell-1} = 1 - (X+1)^{\ell-1}.$$

Then the recursive tower induced by this relation has a positive limit. [33] In the case of $\mathbb{F}_9$ this is *optimal*, i.e., its limit is $\sqrt{9} - 1 = 2$.

[32] See for instance [169], Ch. 7.3, p. 259, Theorem 7.3.1.
[33] See [169], Ch. 7.4, p. 260, Theorem 7.3.2.

To reach the Drinfeld-Vladuts Bound for *each* finite field $\mathbb{F}_q$ with q square, Garcia and Stichtenoth [99] constructed a recursive tower such that

- $N(F_i) \geq (q - \sqrt{q})\sqrt{q}^{i-1}$.
- $g(F_i) \leq \sqrt{q}^i$.

Note that, indeed, this means that

$$\lim_{i\to\infty} \frac{N(F_i)}{g(F_i)} \geq \sqrt{q} - 1,$$

while equality follows from the Drinfeld-Vladuts Bound. Setting $q = \ell^2$, this works with the recursive tower defined by the relation

$$Y^\ell - Y = \frac{X^\ell}{1 - X^{\ell-1}}.$$

12.7.9 Other Lower Bounds on Ihara's Constant

No single value of $A(q)$ is known if q is a non-square. However, there are several lower bounds known.

For instance, by using modular curves and explicit function fields, Zink [182], Bezerra-Garcia-Stichtenoth [23] and Bassa-Garcia-Stichtenoth [7] showed that

$$A(q^3) \geq \frac{2(q^2-1)}{q+2}.$$

Recently, Garcia-Stichtenoth-Bassa-Beelen [101] showed an explicit tower of algebraic function fields over finite fields $\mathbb{F}_{p^{2m+1}}$ for any prime p and integer $m \geq 1$ and proved that this tower gives

$$A(p^{2m+1}) \geq \frac{2(p^{m+1}-1)}{p+1+\epsilon} \quad \text{with} \quad \epsilon = \frac{p-1}{p^m-1}.$$

Using class field theory Serre [162] showed that there exists an absolute positive real constant c such that

$$A(q) > c \cdot \log(q)$$

for each prime power q. One can take $c = 1/96$. [34]

Lower bounds on $A(q)$ have been shown for small primes q such as $q = 2, 3, 5, 7, 11, 13, \ldots$. For instance, Xing and Yeo [36] showed that

$$A(2) \geq 0.258.$$

Finally, note that for the case $\mathbb{F}_p$ with p prime no construction of any explicit tower with a positive limit is known. In fact, there exists a certain class of recursions which does give positive results whenever $q \neq p$, [97] but this *necessarily* fails to give any such result if $q = p$, as shown by Lenstra [129].

[34] See [144], p. 133, Theorem 5.2.9.

12.8 Asymptotically Good Arithmetic Secret Sharing Schemes

The history of asymptotically good arithmetic secret sharing schemes is preceded by that of asymptotic study of bilinear complexity of multiplication in extensions of a finite field, i.e., arithmetic embeddings in our language. This was initiated by Chudnovsky and Chudnovsky [57] in the mid-1980s. Here, the finite field $\mathbb{F}_q$ is fixed and an unbounded number of finite extensions of $\mathbb{F}_q$ is considered. The purpose is to derive upper bounds on the asymptotic ratio between bilinear complexity of multiplication in an extension and its degree.

Using a variation on the techniques of Tsfasman, Vladuts and Zink [172] from their 1982 breakthrough improvement of the Gilbert-Vashamov error correcting bound (which relies on deep results from algebraic geometry [113] in combination with Goppa's idea [104] of algebraic geometry codes), they showed that, surprisingly, this ratio is bounded from above by a *constant* (depending on q). More precisely, set $d = 2$ and fix a finite field $\mathbb{F}_q$. Then there is a real number $c > 1$ with $c = c(q)$ such that

$$\liminf_{k\geq 1}\ n_0/k\ \leq c,$$

where, in our language, n_0 is the length of the shortest arithmetic $(n, 2)$-embedding of $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$. [35]

This work was continued by Shparlinski, Tsfasman and Vladuts [165]. See [34] for the history of the topic up to the late 1990s, as well as for definitions, (non-asymptotic) bounds and generalizations. Papers discussing more recent improvements on explicit bounds include [160, 50, 155].

Motivated by showing a suitable asymptotic version of the "Fundamental Theorem on Information-Theoretically Secure Multi-Party Computation" [16, 52] by Ben-Or, Goldwasser and Wigderson and Chaum, Crépeau and Damgaard from 1988, Chen and Cramer [54] initiated in 2006 the study of *asymptotically good arithmetic secret sharing schemes* and showed the first positive results for the strongest notions, using yet another variation on the algebraic geometric techniques of Tsfasman, Vladuts and Zink.

In 2007, the results of [54] played a central role in the surprising work of Ishai, Kushilevitz, Ostrovsky and Sahai [119] on the "secure multi-party computation in the head" paradigm and its application to communication-efficient zero-knowledge for circuit satisfiability. This caused nothing less than a paradigm shift that perhaps appears even as counter-intuitive: secure *multi-party* computation (and in particular, asymptotically good arithmetic secret sharing) is a very powerful abstract primitive for *communication-efficient two-party cryptography.* Subsequent fundamental results that also rely on the asymptotics from [54] include *two-party secure computation* [117, 76, 83], *OT-combiners* [109], *correlation extractors* [116], *amortized zero knowledge* [69] and *OT from noisy channels* [115].

The results of [54] were strengthened in [43]. A more powerful paradigm for

[35] It is also interesting to consider the lim sup of this ratio.

the construction of arithmetic secret sharing schemes based on novel algebraic geometric ideas was presented in [46, 49]. We first explain the results from [54]. This is followed by an overview in Section 12.9 of the results from [46, 49].

We will show the result from [54] that for some fixed finite fields $\mathbb{F}_q$ and for $d \geq 2$ a constant, there exists an unbounded family of $(n, t, d, n-t)$-arithmetic secret sharing schemes for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ with uniformity such that $n \to \infty$, $k = \Omega(n)$, and $t = \Omega(n)$. The only known proof of existence of such a family crucially exploits good towers of algebraic function fields; no "elementary" proof of existence is known.

Before continuing, it should be noted that if

- $(d, n-t)$-multiplicativity is replaced by (d, n)-multiplicativity,
- and the restrictions $k = 1$ and $d = 2$ are made,

then there *is* an elementary proof of existence of the corresponding family: combine the construction from Section 12.4.3 based on self-dual codes with the basic fact from classical coding theory that asymptotically good self dual codes exist. [36] This idea [56] does have some interesting applications, but unfortunately it does not support the powerful applications mentioned above.

We start with a "dualization" lemma that will prove helpful.

Lemma 12.46 *Let K be a field. Let F be an algebraic function field of genus g with full field of constants K. Let $X, X_0 \in \mathrm{Div}(F)$. Let $W \in \mathrm{Div}(F)$ be a canonical divisor. Then*

$$\ell(X + X_0) = \ell(X) + \deg X_0$$

if and only if

$$\ell(W - X - X_0) = \ell(W - X).$$

Proof This is a direct consequence of Theorem 12.37 (Riemann-Roch): in the first equality of the statement of the lemma, substitute $\ell(X + X_0) = \deg X + \deg X_0 + 1 - g + \ell(W - X - X_0)$ and $\ell(X) = \deg X + 1 - g + \ell(W - X)$. It follows at once that the first equality holds if and only if $\ell(W - X - X_0) = \ell(W - X)$. $\square$

Next, we generalize Theorem 11.84 to work over algebraic function fields.

Theorem 12.47 (Lagrange over Algebraic Function Fields) *Let F/K be an algebraic function field of genus g. Let $\mathbb{P}(F)$ denote the set of places of F. Let*

$$P_1, \ldots, P_m \in \mathbb{P}(F)$$

be distinct places of F. For $= 1, \ldots, m$, define

$$L_i := \mathcal{O}_{P_i}/P_i,$$

[36] i.e., for each finite field, there is an infinite family of self-dual $\mathbb{F}_q$-linear codes such that the minimum distance is linear in the length. See e.g. [138], Ch. 19, par. 6, pp. 629–633.

which is an extension field of K of degree $\deg P_i$. Also define

$$P := \sum_{i=1}^{m} P_i \in \mathrm{Div}(F).$$

Let $X \in \mathrm{Div}(F)$ be such that

$$\mathrm{supp}(X) \cap \mathrm{supp}(P) = \emptyset.$$

Let $W \in \mathrm{Div}(F)$ be a canonical divisor of F. Then the K-vector space morphism

$$\mathcal{E} : \mathcal{L}(X) \to \bigoplus_{i=1}^{m} L_i,$$

$$f \mapsto (f(P_i))_{i=1}^{m}$$

has the following properties.

- *It is injective if and only if $\ell(X - P) = 0$.*
- *It is surjective if and only if $\ell(W - X + P) = \ell(W - X)$.*
  In particular, it is surjective if $\ell(W - X + P) = 0$.

PROOF Since $\mathrm{supp}(X) \cap \mathrm{supp}(P) = \emptyset$, the map $\mathcal{E}$ is well-defined. Its injectivity is equivalent to $\ker \mathcal{E}$ being trivial. Since

$$\ker \mathcal{E} = \mathcal{L}(X - P),$$

the injectivity claim follows.

Surjectivity of the map $\mathcal{E}$ is equivalent to $\mathrm{im}\,\mathcal{E}$ having K-dimension $\deg P$, which, in turn, is equivalent to the condition that

$$\deg P = \ell(X) - \dim_K \ker \mathcal{E}.$$

Since $\dim_K \ker \mathcal{E} = \ell(X - P)$, the first surjectivity claim now follows by substitution of the latter equality into the former and subsequent application of Lemma 12.46.

The second surjectivity claim is an immediate consequence. As P is an effective divisor, it holds that $W - X \leq W - X + P$. Hence,

$$0 \leq \ell(W - X) \leq \ell(W - X + P).$$

In conclusion, it is a sufficient condition for surjectivity that $\ell(W - X + P) = 0$. $\square$

From this point on, let F be an algebraic function field with full field of constants $\mathbb{F}_q$. Let g denote the genus of F and let $\mathbb{P}(F)$ denote the set of places of F.

The following theorem gives a sufficient condition for the existence of codices in terms of the solvability of certain systems of *Riemann-Roch equations.*

THEOREM 12.48 *[54, 46] Let n, d, t, r, k be integers such that $d \geq 1$, $0 \leq t < r \leq n$, and $k \geq 1$. Suppose F has at least $n+k$ distinct rational places $P_1, \ldots, P_n, Q_1, \ldots, Q_k \in \mathbb{P}(F)$. Let $W \in \mathrm{Div}(F)$ be a canonical divisor. Define*

$$Q := \sum_{i=1}^{k} Q_i \ \in \mathrm{Div}(F)$$

and, for each set $B \subseteq \{1, \ldots, n\}$, define

$$P_B := \sum_{i \in B} P_i \ \in \mathrm{Div}(F),$$

with $P_\emptyset := 0 \in \mathrm{Div}(F)$.

If the system of "Riemann-Roch equations"

$$\begin{cases} \ell(W - X + Q + P_B) = 0 & \text{for all } B \subset \{1, \ldots, n\}, |B| = t \\ \ell(dX - P_B) = 0 & \text{for all } B \subset \{1, \ldots, n\}, |B| = r \end{cases}$$

has a solution $X := G$, where $G \in \mathrm{Div}(F)$, then there exists an (n, t, d, r)-codex for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ with uniformity.

PROOF Note that if there is a solution $G \in \mathrm{Div}(F)$, we assume without loss of generality that the support of G avoids each of the places $P_1, \ldots, P_n, Q_1, \ldots, Q_k$: just replace G by the equivalent divisor $G + (f)$ where f is selected using Theorem 12.34 (Weak Approximation Theorem) in such a way that the support of $G + (f)$ avoids each of $P_1, \ldots, P_n, Q_1, \ldots, Q_k$. Clearly, the latter divisor is a solution of the system if G is.

Now let $G \in \mathrm{Div}(F)$ be a solution to the system, subject to the assumption above. The theorem is shown by verifying that the conditions from Theorem 12.14 are satisfied. Define the $\mathbb{F}_q$-linear code

$$\widetilde{C} := \{(f(Q_1), \ldots, f(Q_k), f(P_1), \ldots, f(P_n)) \mid f \in \mathcal{L}(G)\} \subseteq \mathbb{F}_q^k \times \mathbb{F}_q^n.$$

Define $\mathcal{I} = \{-k+1, \ldots, 0, 1, \ldots, n\}$ as the index set for $\mathbb{F}_q^k \times \mathbb{F}_q^n$. Also define $\mathcal{Z} = \{-k+1, \ldots, 0\}$ and $\mathcal{I}^* = \{1, \ldots, n\}$. Since

$$0 \leq \ell(W - G + Q) \leq \ell(W - G + Q + P_B) = 0$$

for all $B \subset \mathcal{I}^*$ with $|B| = t$, it follows by Theorem 12.47 on account of the equality $\ell(W - G + Q) = 0$ that

$$\pi_{\mathcal{Z}}(\widetilde{C}) = \mathbb{F}_q^k.$$

Note that, in particular, this means that $\ell(G) > 0$ and $\deg G \geq 0$. Similarly, it follows by Theorem 12.47 on account of the equality $\ell(W - G + Q + P_B) = 0$, that if $t > 1$, then for all $B \subset \mathcal{I}^*$ with $|B| = t$, it holds that

$$\pi_{\mathcal{Z},B}(\widetilde{C}) = \mathbb{F}_q^k \times \mathbb{F}_q^t.$$

Finally, define

$$\widetilde{D} := \{((\widetilde{f}(Q_1), \ldots, \widetilde{f}(Q_k)), (\widetilde{f}(P_1), \ldots, \widetilde{f}(P_n))) \mid \widetilde{f} \in \mathcal{L}(dG)\} \subseteq \mathbb{F}_q^k \times \mathbb{F}_q^n.$$

Since for all $f_1, \ldots, f_d \in \mathcal{L}(G)$ it holds that

$$\widetilde{f} := \prod_{i=1}^{d} f_i \ \in \mathcal{L}(dG),$$

it follows that

$$\widetilde{C}^{*d} \subset \widetilde{D}.$$

Let $B \subset \mathcal{I}^*$ with $|B| = r$. By Theorem 12.47, if $\widetilde{\mathbf{z}}_B = \mathbf{0}$ for some $\widetilde{\mathbf{z}} \in \widetilde{D}$, then the only possible defining function $\widetilde{f} \in \mathcal{L}(dG)$ underlying $\widetilde{\mathbf{z}}$ is the function that is identically 0. Hence, $\widetilde{\mathbf{z}}_{\mathcal{Z}} = \mathbf{0}$. □

Note that it is not guaranteed that these schemes are unital. However, if G is effective, i.e., $G \geq 0$, then the constants are contained in $\mathcal{L}(G)$, which is a sufficient condition for being unital.

We now discuss a straightforward approach to solving the Riemann-Roch system from Theorem 12.48.

Note that

$$\deg(W - G + Q + P_B) = 2g - 2 - \deg G + k + t$$

for all sets $B \subset \mathcal{I}^*$ with $|B| = t$, and that

$$\deg(dG - P_B) = d \cdot \deg G - r$$

for all sets $B \subset \mathcal{I}^*$ with $|B| = r$. Since a divisor of negative degree has Riemann-Roch dimension equal to 0, it suffices to force

$$2g - 2 - \deg G + k + t < 0 \ \text{ and } \ d \cdot \deg G - r < 0.$$

Under the conditions that

1. $r = d(2g - 1 + k + t) + 1$,
2. $d(2g - 1 + k + t) < n$,

it holds that *any* divisor $G \in \mathrm{Div}(F)$ with degree

$$\deg G = 2g - 1 + k + t$$

is a solution to the system.

Indeed,

$$0 \leq t < r \leq n,$$

$$2g - 2 - \deg G + k + t = 2g - 2 - (2g - 1 + k + t) + k + t = -1 < 0,$$

and

$$d \cdot \deg G - r = d(2g - 1 + k + t) - d(2g - 1 + k + t) - 1 = -1 < 0.$$

Note that there always exist divisors of any given degree m. Namely, for each algebraic function field over $\mathbb{F}_q$ there is a divisor of degree 1 and a divisor of degree m is obtained by multiplying such a divisor by m. [37]

REMARK 12.9 (ONE-POINT DIVISORS) If we assume F has at least $n+k+1$ rational places, then there is a rational place P' distinct from $P_1, \ldots, P_n, Q_1, \ldots, Q_k$. In that case, we can define

$$G = (2g - 1 + k + t) \cdot P'$$

so that the resulting scheme is unital.

Thus, we have shown the following theorem.

THEOREM 12.49 (SOLUTION BY DEGREE) *[54] Let F be an algebraic function field with $\mathbb{F}_q$ as its full field of constants. Let g denote its genus. Let n, d, t, k be integers. Suppose that*

1. $k \geq 1$, $d \geq 1$ and $0 \leq t < n$.
2. F has at least $n + k$ distinct rational places.
3. $d(2g + k + t - 1) < n$.

Then there exists an $(n, t, d, d(2g + k + t - 1) + 1)$-codex for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ with uniformity.

In comparison to Theorem 12.16, the condition $q+1 \geq n+k$ has been replaced by the condition that F has at least $n + k$ rational places, which is weaker. However, this does not come entirely for free, as the second condition $d(2g + k + t-1)+1 \leq n$ involves the genus of F. Before we study these results asymptotically, let us point out that a result similar to that in Theorem 12.49 holds for codices for $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$.

THEOREM 12.50 *[55] Let F be an algebraic function field with $\mathbb{F}_q$ as its full field of constants. Let g denote its genus. Let n, d, t, k be integers. Suppose that*

1. $k \geq 2$, $d \geq 1$ and $0 \leq t < n$.
2. F has at least n distinct rational places.
3. F has a place of degree k.
4. $d(2g + k + t - 1) < n$.

Then there exists an $(n, t, d, d(2g + k + t - 1) + 1)$-codex for $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$ with uniformity.

Note that if $k \geq 4g + 3$, then F has at least one place of degree k. [38] A similar observation as in Remark 12.9 applies here as well if we assume that F has at least $n + 1$ rational places.

[37] See [169], Ch. 5, Proposition 5.1.11.
[38] For this result as well as a generalization, please refer to [169], Ch. 5, Corollary 5.2.10.

We now turn to asymptotics. We focus on $(n, t, d, n-t)$-arithmetic secret sharing schemes for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ with uniformity, as these represent the subclass of codices with the most interesting and powerful applications so far. [39]

THEOREM 12.51 *[54] Let $d \geq 2$ be an integer. Suppose Ihara's constant satisfies*

$$A(q) > 2d.$$

Then there exists an infinite family of $(n, t, d, n-t)$-arithmetic secret sharing schemes for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ with uniformity, such that

1. *$n \longrightarrow \infty$.*
2. *$k = \Omega(n)$.*
3. *$t = \Omega(n)$.*

PROOF Let δ be a real number such that

$$0 < \delta < \frac{1}{1+2d} \quad \text{and} \quad A(q) > \frac{2d}{1-\delta(1+2d)}.$$

Select an infinite sequence of algebraic function fields $F_0, F_1, F_2, \ldots$ over $\mathbb{F}_q$ such that the respective genera g_i and the respective numbers of rational places N_i $(i = 0, 1, 2, \ldots)$ satisfy the following conditions

1. g_i tends to infinity as i tends to infinity.
2. $\frac{N_i}{g_i} > \frac{2d}{1-\delta(1+2d)}$ for all large enough i.

Note that this is possible by definition of $A(q)$ in combination with the hypothesis that $A(q) > 2d$.

Set

$$k_i = \lfloor \delta N_i \rfloor, \ t_i = \lfloor \delta N_i \rfloor, \ n_i = N_i - k_i$$

for $i = 0, 1, 2, \ldots$. Since N_i tends to infinity as i tends to infinity and $n_i \geq (1-\delta)N_i$, it follows at once that, as required, n_i tends to infinity and k_i, t_i are linear in n_i.

Furthermore,

$$d(2g_i + k_i + t_i - 1) + 1 \leq 2dg_i + 2d\delta N_i + 1 \leq N_i - \delta N_i \leq N_i - k_i = n_i$$

for all large enough i. Indeed, the first and third inequalities follow by definition and the second follows since $2dg_i < (1-\delta(1+2d))N_i$ for all large enough i. Now apply Theorem 12.49. $\square$

These schemes can be efficiently constructed and operated for certain asymptotically good (optimal) towers [166], i.e., there is an efficient algorithm to construct a generator matrix for the code C and a matrix representing ψ for each codex (C, ψ) in this family. [40]

[39] For an asymptotical result on $(n, t, d, n-t)$-arithmetic secret sharing schemes for $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$, please refer to [55].

[40] See also [146] and the references therein.

Implementing Remark 12.9 (one-point divisors) does not affect the asymptotic result. So if it is deemed convenient, the schemes may be assumed *unital* as well. [41]

By the result from Section 12.5.4, these codices allow for efficient "recovery of the secret given a full vector of shares with t malicious errors" (even in "higher powers" of the codices). See [46] for a statement of Theorem 12.51 that takes this into account. See [55] for an asymptotic version of Theorem 12.50.

REMARK 12.10 (SUFFICIENT CONDITIONS) The conditions of this theorem are satisfied if q is a square and $q > (2d+1)^2$, as follows from Theorem 12.44 (Ihara) as well as from Garcia-Stichtenoth's results (see Section 12.7.8). Alternatively, by Theorem 12.41 (Serre), the conditions are satisfied if q is very large. Additional sufficient conditions can be extracted from Section 12.7.9.

COROLLARY 12.52 *[54] If q is square and $q \geq 49$, it holds that $\widehat{\tau}(q) \geq 1 - \frac{4}{\sqrt{q}-1} > 0$.*

PROOF Since $k = 1$ in this case, the conditions can be relaxed to $0 < \delta < \frac{1}{1+d}$ and $A(q) > \frac{2d}{1-\delta(1+d)}$ after a slight adaptation of the proof. Substituting $A(q) = \sqrt{q}-1$ and $d = 2$, we get the conditions $0 < \delta < \frac{1}{3}$ and $3\delta < 1 - \frac{4}{\sqrt{q}-1}$. Then set $t_i = \lfloor \delta N_i \rfloor$ and $n_i = N_i - 1$. Since 3δ can be selected arbitrarily close to $1 - \frac{4}{\sqrt{q}-1}$, the desired result follows. $\square$

12.9 The Torsion-Limit and Its Applications

An interesting question is whether the contention of Theorem 12.51 holds for *any* finite field, i.e., whether it holds without the condition $A(q) > 2d$.

A version of Theorem 12.51 that is valid for *any* finite field $\mathbb{F}_q$ in the case $d = 2$ is shown in [43]. The idea is to combine Theorem 12.51 over an extension field $\mathbb{F}_{q^\ell}$ for which $A(q^\ell) > 4$ with (one or more steps of) a dedicated field descent [42] involving an arithmetic embedding of $\mathbb{F}_{q^\ell}$ over $\mathbb{F}_q$. Unfortunately, it has some drawbacks. First, this type of descent removes *uniformity*, a property required in some applications such as [115]. Second, the result does not generalize to all finite fields $\mathbb{F}_q$ when $d > 2$. Third, the descent incurs substantial overhead, thereby affecting the magnitude of the constant factors in the expressions for k, t. However, it *does* show that $\widehat{\tau}(q) > 0$ for all finite fields $\mathbb{F}_q$. [43]

For the construction of the arithmetic secret sharing schemes in Theorem 12.51, the method of "solving-by-degree" of the relevant Riemann-Roch systems has

[41] Methods such as the one from [166] even *require* one-point divisors for efficiency.

[42] In the language of coding theory, a field descent refers to a *concatenation* method.

[43] Note that [43] deals with the case $k = 1$ only; however, the result of [43] generalizes to $k = \Omega(n)$ in a straightforward way. It can also be generalized easily to the case $2 \leq d \leq q$.

been combined with the existence of certain asymptotically good towers of algebraic function fields. This approach has been substantially improved in [46, 49], [44] leading to a relaxation of the condition $A(q) > 2d$ in Theorem 12.51. This time, however, uniformity is *preserved*. As it turns out, it follows that Theorem 12.51 holds for much smaller values of q. Moreover, as overhead incurred by the dedicated field descent is avoided, the constant factors in the expressions for k, t are also improved.

The improvement consists of two parts. First, the solving strategy for the Riemann-Roch systems has been refined with the aid of the *torsion limit* of a tower. For a given tower $\mathcal{F}/\mathbb{F}_q$, this strategy does not only exploit information about the Ihara limit of $\mathcal{F}$, [45] but also information on the asymptotics of the ratio between (the base-q logarithm of) the order of the d-torsion subgroup [46] $\mathcal{J}_F[d] \subset \mathrm{Cl}(F)$ and the genus $g(F)$, as F ranges over $\mathcal{F}$. More precisely, the d-torsion limit of $\mathcal{F}$ is defined as

$$J_d(\mathcal{F}) = \liminf_{F \in \mathcal{F}} \frac{\log_q |\mathcal{J}_F[d]|}{g(F)}.$$

This strategy allows for identification of solutions with a smaller degree than the ones selected in the “solving by degree method,” which is a key factor behind the relaxed condition mentioned above.

Second, new parameters are shown that satisfy this relaxed condition. This involves further results from algebraic geometry, notably about torsion-subgroups of abelian varieties and about p-rank in certain Artin-Schreier extensions of functions fields. It leads to the conclusion that Theorem 12.51 holds for much smaller values of q. We now present an outline.

For the moment, take a fixed function field $F/\mathbb{F}_q$ in mind. It is useful to start by recalling the following facts. Write h for the order of the (degree-0 divisor) class group $\mathrm{Cl}(F)$. Let $r \geq 0$ be an integer. The set $\mathrm{Div}_r(F)$ of divisors of degree r is partitioned into exactly h equivalence classes when taken modulo the principal divisors $\mathrm{Prin}(F)$. A divisor of degree r has positive Riemann-Roch dimension if and only if each element of its equivalence class does, which is in turn equivalent to the statement that it contains some effective divisor of degree r. In other words, taking $\mathrm{Div}_r(F)$ modulo $\mathrm{Prin}(F)$, the number of classes having positive Riemann-Roch dimension is at most A_r, the number of *effective* divisors of degree r.

Consider the Riemann-Roch system at hand and let X be the “variable” divisor of degree denoted by s. Attention may be restricted to the class of X modulo the group of principal divisors. The idea is to (greedily) estimate the number m of classes in $\mathrm{Div}_s(F)$ modulo $\mathrm{Prin}(F)$ which represent “non-solutions,” and then to derive a manageable condition under which $h > m$, thereby automatically giving

[44] The article [49] is the extended version of [46]. In addition, it includes applications to frameproof codes and arithmetic embeddings.

[45] Recall that this is the limit of the ratio between the number of rational points and the genus, with the genus tending to infinity.

[46] The subgroup of elements annihilated when multiplied by d.

a sufficient condition for the existence of a solution and, hence, for the existence of the desired codex.

Now consider the relevant system underlying Theorem 12.51, dictated by Theorem 12.48. Each of the equations is of the form $\ell(-X+U)=0$ or of the form $\ell(dX+V)=0$. We focus on the latter type of equation; the treatment of the former is essentially a special case. Write $r=\deg(dX+V)$. Suppose X is a non-solution, i.e., $\ell(dX+V)>0$. Then there are at most A_r possibilities for the class of dX. Consequently, there are at most $A_r\cdot|\mathcal{J}_F[d]|$ possibilities for the class of X: the difference of any two is an element of the class group and this element is annihilated when multiplied by d. Using a "greedy union" argument, an upper bound is obtained on the number of classes in $\mathrm{Div}_s(F)$ modulo $\mathrm{Prin}(F)$ representing a non-solution to *some* equation in the system. This leads to the sufficient condition that

$$h>\binom{n}{t}(A_{r_1}+A_{r_2}|\mathcal{J}_F[d]|),$$

where $r_1:=2g-s+t+k-2$ and $r_2:=ds-n+t$. [47]

Careful inspection of the Zeta function of F yields an upper bound on the ratio A_r/h whenever $0\le r<g$ and $g\ge 1$, given as a function of g,r,q. Namely

$$\frac{A_r}{h}\le\frac{g}{q^{g-r-1}(\sqrt{q}-1)^2}.$$

Substitution of this upper bound turns the sufficient condition into another one just involving the parameters s,k,n,t,g, as well as d,q: the class number h has dropped out.

Now consider a tower $\mathcal{F}/\mathbb{F}_q$ with Ihara-limit $A>0$ and let $d=2$. After some calculus, the ideas discussed above lead to the conclusion that this tower gives rise to the desired schemes from Theorem 12.51 if

$$A>1+J_2(\mathcal{F}).$$

See [49] for the precise condition in the case $d>2$.

Now, for given real number A with $0<A\le A(q)$, let the quantity $J_d(q,A)$ denote the lim inf of $J_d(\mathcal{F})$, taken over all towers over $\mathbb{F}_q$ with Ihara limit at least A. [48] It is now possible to finally state the relaxed condition for Theorem 12.51. For instance, in the case $d=2$, it holds for a given finite field $\mathbb{F}_q$ if there is a real number A such that $0<A\le A(q)$ and such that

$$A>1+J_2(q,A).$$

[47] The method of showing existence of certain codes by solving appropriate (what we call) "Riemann-Roch systems" is due to Vladuts [179] and has been followed by several authors since. A novelty in the systems discussed here is treatment of d-torsion considerations arising from $d>1$.

[48] With some care, this definition can be generalized to infinite *families* of function fields, not just towers. The stated results will still hold.

Note that, in particular, an asymptotically good tower is required, as before. Again, see [49] for the precise condition in the case $d > 2$.

The second step is to investigate when the condition is satisfied. Generally, the order of the d-torsion subgroup can be upper bounded using a theorem of Weil on d-torsion in abelian varieties or by a method involving the Weil-pairing for abelian varieties. This leads to the following "generic" bounds in the case that d is *prime.*

- If $d \mid (q-1)$ then
$$J_d(q, A(q)) \leq \frac{2}{\log_d q}.$$
- If $d \not| \, (q-1)$ then
$$J_d(q, A(q)) \leq \frac{1}{\log_d q}.$$

However, [49] certain Artin-Schreier towers admit a dedicated analysis of d-torsion, as shown in [46, 49]. Under the restriction that $d = p$, where p is the characteristic of $\mathbb{F}_q$, the idea is to apply a combination of the Deuring-Shafarevich formula for p-rank with the Riemann-Hurwitz genus formula so as to obtain a recursion involving p-ranks and genera only (as well as q). [50] Since the p-rank of F equals $\log_p q \cdot \log_q |\mathcal{J}_F[p]|$, this gives in principle an alternative way to access information about the torsion limit for such towers.

Briefly, for this idea to work, it is a convenient requirement that in consecutive steps of the tower, the "error terms" in both formulas (arising from ramification in the tower) differ by a non-zero constant so that the desired recursion is simply obtained from the two formulas by Gaussian Elimination. Precise knowledge of the genus in each step of the tower is another convenient requirement. Once substituted into the recursion, the recursion can thus be solved and precise knowledge of the p-rank in each step of the tower is obtained.

Applying this approach to an optimal tower given by Garcia and Stichtenoth [98] this leads to a substantial improvement on the "generic" approach. Namely:

Theorem 12.53 *[46, 49] If d is a prime, q is a square and $d|q$ then*
$$J_d(q, \sqrt{q}-1) \leq \frac{1}{(\sqrt{q}+1)\log_d q}.$$

[49] The torsion limit [46, 49] described above has been introduced independently in [154], in the context of bounds on frame-proof codes. The generic bounds are also discussed in [154], but not the superior bounds for the Artin-Schreier case from [46, 49] given below. See [46, 49] for additional applications to bilinear complexity of extension field multiplication and frame-proof codes.

[50] The p-rank of a function field $F/\mathbb{F}_q$, a. k. a. the *Hasse-Witt invariant*, is the dimension of the finite dimensional $\mathbb{F}_p$-vector space $\mathcal{J}_{\widehat{F}}[p]$, where $\widehat{F}$ is the function field obtained by extending the field of constants of F to an algebraic closure $\overline{\mathbb{F}}_q$ of $\mathbb{F}_q$, i.e., $\widehat{F} = F \cdot \overline{\mathbb{F}}_q$.

There are some other asymptotically good (optimal) towers known where this approach applies as well, but there the result is not nearly as good [5].

All in all, this leads to the following results.

Theorem 12.54 *[46, 49] Let q be a prime power and let $d = 2$. Suppose there exists a real number A such that $0 < A \leq A(q)$ and such that $A > 1 + J_2(q, A)$. Then there exists an infinite family of $(n, t, 2, n - t)$-arithmetic secret sharing schemes for $\mathbb{F}_q^k$ over $\mathbb{F}_q$ with uniformity, such that*

1. $n \longrightarrow \infty$.
2. $k = \Omega(n)$.
3. $t = \Omega(n)$.

Corollary 12.55 *The conditions of Theorem 12.54 are satisfied for all finite fields $\mathbb{F}_q$ with $q \geq 8$ and $q \neq 11, 13$.*

See [49] for statements in the case $d > 2$.

The proof of Theorem 12.54 also directly implies the following result:

Corollary 12.56 *Let $\mathbb{F}_q$ be a finite field with $q \geq 8$ and $q \neq 11, 13$. Then there exists an infinite family of $\mathbb{F}_q$-linear codes C such that C, $C^{\perp}$ and C^{*2} are simultaneously asymptotically good.*

The case where just C, C^{*2} are considered (so the dual $C^{\perp}$ is left out of consideration) has been shown to hold for all finite fields [156], using an algebraic geometric argument in combination with a refined field descent method. Corollary 12.56 also has a formulation for the case $d > 2$. Techniques similar to the ones employed in the proofs of Theorem 12.54 and Corollary 12.56 also lead to improved results on frameproof codes and arithmetic embeddings of finite fields, see [49].

Remark 12.11 In principle, Theorem 12.54 and Corollary 12.56 are non-constructive. However, a plausibly efficient *randomized* construction is discussed in [49], based in part on the observation that, given a suitable tower, a random divisor class of a certain degree constitutes a solution of the relevant Riemann-Roch system with overwhelming probability: the number of non-solutions is a negligible fraction of the class number.

Finally, it is interesting to note that, at present, there is no *elementary* proof for the existence of the asymptotically good arithmetic secret sharing schemes shown in Theorem 12.51, whose proof is algebraic geometric and requires asymptotically good towers of functions fields.[51] This is, so far, in contrast with the theory of error correcting codes, where the existence of asymptotically good codes can be shown by elementary (probabilistic) methods.

[51] The situation is the same for asymptotically good arithmetic embeddings of finite fields.

12.10 Outlook

There are several meaningful ways to extend the codex notion. We mention a few. Presently, one of the most promising extensions, both from a technical and from an application point of view, is to consider, in addition, *automorphism groups* of codices; this is motivated by recent results [18, 19, 74]. Motivated by a technical argument in [156], it is relevant to consider a generalization of the codex notion where not only multiplication in the presented K-algebra A is considered, but, *simultaneously*, several other multi-linear maps. It also makes sense to define a codex over K-algebras, not just over fields K, i.e., the coordinates of the code take value in a K-algebra rather than in the field K. [52] Moreover, so far, the codex notion is inherently commutative. It is natural to bring non-commutativity into play. On the application side, new uses of codices keep coming up time and again.

The most powerful way to construct asymptotically good arithmetic secret sharing schemes in the subclass addressed in Theorem 12.51 is by using the torsion-limit approach from Theorem 12.54 and Corollary 12.56. Is that approach anywhere *close to optimal?* Or can it be improved, possibly by some alternative approach or a further refinement? An interesting question is whether the asymptotically good arithmetic secret sharing schemes in the subclass addressed in Theorem 12.51 *exist over* $\mathbb{F}_2$. Moreover, with a few exceptions, there has been very little emphasis on the study of *non-asymptotic codex constructions* so far.

For small fields and in the case $k = 1$, the constructive bound from Corollary 12.52 is still *far removed* from the numerical limitations imposed by Theorem 12.33 (and its improvements). [53] A sizeable portion of this gap may be caused by the fact that the techniques for proving limitations are still rather *crude*. Yet, major improvements seem to require new approaches, algebraic-combinatorial or otherwise: known bounding techniques (including those inspired by bounds from the theory of error correcting codes) lack sensitivity to capture well enough the substantial influence of the powering operation.

Improvement of the known *descent methods* is another direction. For instance, it would be interesting to *preserve dual distance* by a descent method. However, the paradigm as we know it works “locally” in that, in particular, it replaces each coordinate by some (necessarily) redundant expansion and thus turns any good dual distance very bad. Therefore, in order to solve this, an entirely new idea is required that works in a more *global* way, so to speak. In addition, presently known descent techniques do not suffice to show the existence of codes C such that both C and C^{*d} are asymptotically good *in the case* $d > 2$.

On the strictly algebraic geometric side, research on the torsion limit has just started. A major question is *whether it is possible that* $J_d(q, A) = 0$ for some finite field $\mathbb{F}_q$, some integer $d \geq 2$ and some real number $A > 0$. Improving the non-trivial bounds known so far is also important, but a particularly intriguing

[52] A special case has been defined and studied in [68]. Another motivation for such an extension is given by the results in [14].

[53] See the discussion in [46, 49].

question is whether there are (many?) *other good towers of the Artin-Schreier type* than just the one from [98] that give such bounds (or perhaps even better ones) using the Deuring-Shafarevich approach, when compared to Theorem 12.53.

Although the understanding of explicit equations for recursive towers from *modular curves* is steadily improving, [54] it is presently not clear whether the result from Theorem 12.53 can be explained from the theory of modular curves or whether it can perhaps even be improved using that theory. Moreover, are there good bounding techniques dedicated to good towers of *another type* than Artin-Schreier? Finally, there are important questions in *computational algebraic geometry* about the *efficiency of solving Riemann-Roch systems* (see Remark 12.11 for a possible direction).

[54] See [6], where curves on higher dimensional (Drinfeld) modular varieties are used to obtain equations for good recursive towers over all non-prime finite fields.

Notation

- $x \overset{\text{def}}{=} y$, states that x is defined by y, e.g., $\delta(x, y) \overset{\text{def}}{=} |x - y|$.
- $x = y$, the statement that x is equal to y, e.g., $2 + 2 = 4$.
- $x \leftarrow y$, the action that x is assigned y, e.g., $i \leftarrow 1$.
- $x \leftarrow D$, where D is a distribution, means that x is sampled according to D.
- $x \in_{\mathrm{R}} S$, where S is a finite set, means that x is sampled uniformly at random from S.
- $a \leftarrow A(x; r)$, running randomized algorithm on input x and randomness r and assigning the output to a.
- $\mathbb{N} \overset{\text{def}}{=} \{0, 1, 2, \ldots\}$.
- $\mathbb{Z}_n \overset{\text{def}}{=} \{0, \ldots, n - 1\}$.
- $\mathbb{F}$, generic finite field.
- $\mathrm{GF}(q^e)$, the Galois field with q^e elements.
- $[a, b]$, closed interval from a to b.
- $\kappa \in \mathbb{N}$, the security parameter.
- ϵ, the empty string.
- $\Pr[E|C]$, probability of event E given event C.
- $\Pr[a \leftarrow A; b \in_{\mathrm{R}} B : E]$, probability of event E given a and b are generated as specified.
- $\delta(X, Y)$, statistical distance between random variable X and Y.
- $\textsc{Adv}_A(X, Y)$, (distinguishing) advantage of A for random variables X and Y.
- $\overset{\text{perf}}{\equiv}, \overset{\text{stat}}{\equiv}, \overset{\text{comp}}{\equiv}$, perfectly indistinguiahable, statistically indistinguiahable respectively computationally indistinguiahable.
- $\overset{\text{perf}}{\not\equiv}, \overset{\text{stat}}{\not\equiv}, \overset{\text{comp}}{\not\equiv}$, *not* perfectly indistinguiahable, statistically indistinguiahable respectively computationally indistinguiahable.
- $\mathtt{X}$, formal variable in polynomials, $f(\mathtt{X})$.
- $1, 2, \ldots, n \in \mathbb{F}$, naming of n distinct, non-zero elements from $\mathbb{F}$, for secret sharing.
- $[a; f] = (f(1), \ldots, f(n))$, secret sharing of a using polynomial f. States that $f(0) = a$.
- $[a]$, secret sharing of a. States that there exists $f(\mathtt{X})$ such that $[a] = [a; f]$.
- $\deg(f(\mathtt{X}))$, degree of polynomial.
- $[a]_d$, degree-d secret sharing of a. States that there exists $f(\mathtt{X})$ such that $\deg(f(\mathtt{X})) \leq d$ and $[a] = [a; f]$.
- $[a; \rho]_M = M(a, \rho)$, secret sharing of a using matrix M.

- U, V, M, vector spaces and matrices are in capital, normal font.
- $\mathsf{A}, \mathsf{B}, \mathsf{P}_i$, agents and parties in interactive systems.
- $\mathcal{IS}$, interactive system.
- $\mathcal{IS}_1 \diamond \mathcal{IS}_2$, composing interactive systems.
- A^T, matrix transpose.
- AB, matrix multiplication.
- $\mathbf{r} = (r_1, \ldots, r_\ell)$, column vector.
- $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^\mathsf{T}\mathbf{b}$, inner product.
- For $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{b} = (b_1, \ldots, b_n)$, $\mathbf{a} * \mathbf{b} = (a_1b_1, \ldots, a_nb_n)$, Schur product.
- a_{ij}, entry in row i, column j in matrix A.
- $\mathbf{a}_i$ and $\mathbf{a}_{i*}$, row i of matrix A.
- $\mathbf{a}_{*j}$, column j of A.
- ker, kernel of a function, $\ker(f) = \{x | f(x) = 0\}$.
- im, image of a function, $\mathrm{im}(f) = \{y | \exists x (y = f(x))\}$.
- ord, order of a group element.
- supp, support.
- Div
- Prin
- Cl

List of Algorithms

References

[1] *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, Chicago, Illinois, 2–4 May 1988.

[2] M.F. Atiyah and I.G. Macdonald. *Introduction to Commutative Algebra.* Addison-Wesley Reading, 1969.

[3] Simeon Ball. On sets of vectors of a finite vector space in which every subset of basis size is a basis. *J. Eur. Math. Soc.*, 14:733–748, 2012.

[4] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195. IEEE Computer Society, 2004.

[5] Alp Bassa and Peter Beelen. The Hasse-Witt invariant in some towers of function fields over finite fields. *Bulletin of the Brazilian Mathematical Society*, 41:4:567–582, 2010.

[6] Alp Bassa, Peter Beelen, Arnaldo Garcia, and Henning Stichtenoth. Towers of function fields over non-prime finite fields. *Preprint, 2012. arXiv:1202.5922v2.*

[7] Alp Bassa, Arnaldo Garcia, and Henning Stichtenoth. A new tower over cubic finite fields. *Moscow Mathematical Journal*, 8(3):401–418, September 2008.

[8] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology - Crypto '91*, pages 420–432, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science Volume 576.

[9] Donald Beaver. Foundations of secure interactive computing. In Joan Feigenbaum, editor, *Advances in Cryptology - Crypto '91*, pages 377–391, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science Volume 576.

[10] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 503–513, Baltimore, Maryland, 14–16 May 1990.

[11] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure mpc with linear communication complexity. In Canetti [38], pages 213–230.

[12] Amos Beimel. *Secure Schemes for Secret Sharing and Key Distribution.* PhD thesis, Department of Computer Science, Technion, 1996.

[13] Amos Beimel. Secret-sharing schemes: A survey. In Yeow Meng Chee, Zhenbo Guo, San Ling, Fengjing Shao, Yuansheng Tang, Huaxiong Wang, and Chaoping Xing, editors, *IWCC*, volume 6639 of *Lecture Notes in Computer Science*, pages 11–46. Springer, 2011.

[14] Amos Beimel, Aner Ben-Efraim, Carles Padró, and Ilya Tyomkin. Multi-linear secret-sharing schemes. In *Proc. TCC '14*, 2014.

[15] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Simon [168], pages 1–10.

[16] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Simon [168], pages 1–10.

[17] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. *IACR Cryptology ePrint Archive*, 2011:629, 2011.

[18] Eli Ben-Sasson, Ariel Gabizon, Yohay Kaplan, Swastik Kopparty, and Shubhangi Saraf. A new family of locally correctable codes based on degree-lifted algebraic geometry codes. In Boneh et al. [30], pages 833–842.

[19] Eli Ben-Sasson, Yohay Kaplan, Swastik Kopparty, Or Meir, and Henning Stichtenoth. Constant rate PCPs for Circuit-SAT with sublinear query complexity. In *FOCS*, pages 320–329. IEEE Computer Society, 2013.

[20] Josh Cohen Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In Shafi Goldwasser, editor, *CRYPTO*, volume 403 of *Lecture Notes in Computer Science*, pages 27–35. Springer, 1988.

[21] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Paterson [150], pages 169–188.

[22] Michael Bertilsson and Ingemar Ingemarsson. A construction of practical secret sharing schemes using linear block codes. In Jennifer Seberry and Yuliang Zheng, editors, *AUSCRYPT*, volume 718 of *Lecture Notes in Computer Science*, pages 67–79. Springer, 1992.

[23] J. Bezerra, A. Garcia, and H. Stichtenoth. An explicit tower of function fields over cubic finite fields and Zink's lower bound. *J. Reine Angew. Math.*, 589, December 2005.

[24] G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference 48*, pages 313–317, 1979.

[25] G. R. Blakley and Catherine Meadows. Security of ramp schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO*, volume 196 of *Lecture Notes in Computer Science*, pages 242–268. Springer, 1984.

[26] GR Blakley and GA Kabatianski. Ideal perfect threshold schemes and MDS codes. In *Information Theory, 1995. Proceedings., 1995 IEEE International Symposium on*, page 488. IEEE, 1995.

[27] Carlo Blundo, Alfredo De Santis, and Ugo Vaccaro. Efficient sharing of many secrets. In Patrice Enjalbert, Alain Finkel, and Klaus W. Wagner, editors, *STACS*, volume 665 of *Lecture Notes in Computer Science*, pages 692–703. Springer, 1993.

[28] Carlo Blundo, Alfredo De Santis, and Ugo Vaccaro. On secret sharing schemes. *Inf. Process. Lett.*, 65(1):25–32, 1998.

[29] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343. Springer, 2009.

[30] Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors. *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013.* ACM, 2013.

[31] G. Bracha. An o(log n) expected rounds randomized byzantine generals protocol. *J. ACM*, 34(4):910–920, 1987.

[32] Ernie Brickell. Some ideal secret sharing schemes. *J. Combin. Math. and Combin. Comput.*, 9:105–113, 1989.

[33] Nader H. Bshouty. Multilinear complexity is equivalent to optimal tester size. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:11, 2013.

[34] Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic Complexity Theory.* Grundlehren der mathematischen Wissenschaften. Springer, 1997.

[35] K. A. Bush. Orthogonal arrays of index unity. *Ann. Math. Stat.*, 23:426–434, 1952.

[36] S.L. Yeo. C. Xing. Algebraic curves with many points over the binary field. *J. of Algebra*, 311:775–780, 2007.

[37] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, 14–17 October 2001. IEEE. Full version in [**?**].

[38] Ran Canetti, editor. *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*. Springer, 2008.

[39] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.

[40] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 639–648, Philadelphia, Pennsylvania, 22–24 May 1996.

[41] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.

[42] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on the Theory of Computing*, pages 494–503, Montreal, Quebec, Canada, 2002.

[43] Ignacio Cascudo, Hao Chen, Ronald Cramer, and Chaoping Xing. Asymptotically good ideal linear secret sharing with strong multiplication over *any* fixed finite field. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 466–486. Springer, 2009.

[44] Ignacio Cascudo, Ronald Cramer, Diego Mirandola, Carles Padró, and Chaoping Xing. On secret sharing with nonlinear product reconstruction. *Preprint*, 2013.

[45] Ignacio Cascudo, Ronald Cramer, Diego Mirandola, and Gilles Zémor. On powers of codes. *Preprint*, 2013.

[46] Ignacio Cascudo, Ronald Cramer, and Chaoping Xing. The torsion-limit for algebraic function fields and its application to arithmetic secret sharing. In Rogaway [158], pages 685–705. Early versions had been widely circulated since November 2009.

[47] Ignacio Cascudo, Ronald Cramer, and Chaoping Xing. The Arithmetic Codex. *IACR Cryptology ePrint Archive*, 2012:388, 2012. A 5-page summary also appeared in Proceedings of IEEE Information Theory Workshop (ITW) 2012.

[48] Ignacio Cascudo, Ronald Cramer, and Chaoping Xing. Bounds on the threshold gap in secret sharing and its applications. *IEEE Transactions on Information Theory*, 59(9):5600–5612, 2013.

[49] Ignacio Cascudo, Ronald Cramer, and Chaoping Xing. Torsion limits and Riemann-Roch systems for function fields and applications. *IEEE Transactions in Information Theory, to appear*, 2014.

[50] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and An Yang. Asymptotic bound for multiplication complexity in the extensions of small finite fields. *IEEE Transactions on Information Theory*, 58(7):4930–4935, 2012.

[51] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In ACM [1], pages 11–19.

[52] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In Simon [168], pages 11–19.

[53] David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In Carl Pomerance, editor, *Advances in Cryptology - Crypto '87*, pages 87–119, Berlin, 1987. Springer-Verlag. Lecture Notes in Computer Science Volume 293.

[54] Hao Chen and Ronald Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 521–536. Springer, 2006.

[55] Hao Chen, Ronald Cramer, Robbert de Haan, and Ignacio Cascudo Pueyo. Strongly multiplicative ramp schemes from high degree rational points on curves. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 451–470. Springer, 2008.

[56] Hao Chen, Ronald Cramer, Shafi Goldwasser, Robbert de Haan, and Vinod Vaikuntanathan. Secure computation from random error correcting codes. In Naor [142], pages 291–310.

[57] D. V. Chudnovsky and G. V. Chudnovsky. Algebraic complexities and algebraic curves over finite fields. *J. Complexity*, pages 285–316, 1988.

[58] Gil Cohen, Ivan Bjerre Damgård, Yuval Ishai, Jonas Kölker, Peter Bro Miltersen, Ran Raz, and Ron D. Rothblum. Efficient multiparty protocols via log-depth threshold formulae - (extended abstract). In Ran Canetti and Juan A. Garay, editors, *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 185–202. Springer, 2013.

[59] Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *GTM*. Springer, 1993.

[60] Irene Marquez Corbella and Ruud Pellikaan. Error-correcting pairs for a public-key cryptosystem. *CoRR*, abs/1205.3647, 2012.

[61] Alain Couvreur, Philippe Gaborit, Valérie Gauthier-Umaña, Ayoub Otmani, and Jean-Pierre Tillich. Distinguisher-based attacks on public-key cryptosystems using reed-solomon codes. *CoRR*, abs/1307.6458, 2013. To appear in Des. Codes Crypto.

[62] Thomas Cover and Joy Thomas. *Elements of Information Theory.* Wiley, 1991.

[63] Ronald Cramer. The Arithmetic Codex: Theory and applications. In Paterson [150], page 1. Abstract of invited talk.

[64] Ronald Cramer, Ivan Damgaard, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology - EuroCrypt 2001*, pages 280–300, Berlin, 2001. Springer-Verlag. Lecture Notes in Computer Science Volume 2045.

[65] Ronald Cramer, Ivan Damgård, and Stefan Dziembowski. On the complexity of verifiable secret sharing and multiparty computation. In *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing*, pages 325–334, Portland, OR, USA, 21–23 May 2000.

[66] Ronald Cramer, Ivan Damgård, Stefand Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In Jacques Stern, editor, *Advances in Cryptology - EuroCrypt '99*, pages 311–326, Berlin, 1999. Springer-Verlag. Lecture Notes in Computer Science Volume 1592.

[67] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *Advances in Cryptology - EuroCrypt 2000*, pages 316–334, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1807.

[68] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2000.

[69] Ronald Cramer, Ivan Damgård, and Valerio Pastro. On the amortized complexity of zero knowledge protocols for multiplicative relations. In Adam Smith, editor, *ICITS*, volume 7412 of *Lecture Notes in Computer Science*, pages 62–79. Springer, 2012.

[70] Ronald Cramer, Vanesa Daza, Ignacio Gracia, Jorge Jiménez Urroz, Gregor Leander, Jaume Martí-Farré, and Carles Padró. On codes, matroids, and secure multiparty computation from linear secret-sharing schemes. *IEEE Transactions on Information Theory*, 54(6):2644–2657, 2008. Preliminary version in CRYPTO 2005.

[71] Ronald Cramer and Serge Fehr. The mathematical theory of information and its applications to privacy amplification (and more). *Course notes, 29pp., Mathematical Institute, Leiden University, version 2.0, 2011.* Available from `www.cwi.nl/crypto/`.

[72] Ronald Cramer and Serge Fehr. Optimal black-box secret sharing over arbitrary abelian groups. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 272–287. Springer, 2002.

[73] Ronald Cramer, Serge Fehr, and Martijn Stam. Black-box secret sharing from primitive sets in algebraic number fields. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 344–360. Springer, 2005.

[74] Ronald Cramer, Carles Padró, and Chaoping Xing. Optimal algebraic manipulation codes. *Manuscript*, 2013.

[75] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006.

[76] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2010.

[77] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In Wagner [180], pages 241–261.

[78] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In Mihir Bellare, editor, *Advances in Cryptology - Crypto 2000*, pages 432–450, Berlin, 2000. Springer-Verlag. Lecture Notes in Computer Science Volume 1880.

[79] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In D. Boneh, editor, *Advances in Cryptology - Crypto 2003*, pages 247–264, Berlin, 2003. Springer-Verlag. Lecture Notes in Computer Science Volume 2729.

[80] Ivan Damgard, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535, 2011.

[81] Ivan Damgård and Rune Thorbek. Linear integer secret sharing and distributed exponentiation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2006.

[82] Ivan Damgård and Rune Thorbek. Non-interactive proofs for integer multiplication. In Naor [142], pages 412–429.

[83] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *TCC*, pages 621–641, 2013.

[84] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127. Springer, 1987.

[85] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer, 1989.

[86] Yvo Desmedt and Yair Frankel. Perfect homomorphic zero-knowledge threshold schemes over any finite abelian group. *SIAM J. Discrete Math.*, 7(4):667–679, 1994.

[87] Yevgeniy Dodis, Amit Sahai, and Adam Smith. On perfect and adaptive security in exposure-resilient cryptography. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 301–324. Springer, 2001.

[88] Iwan Duursma. Algebraic geometry codes: general theory. In D. Ruano, E. Martínez-Moro, and C. Munuera, editors, *Advances in algebraic geometry codes*, pages 1–48. World Scientific, 2008.

[89] Iwan Duursma and Ralf Kötter. Error-locating pairs for cyclic codes. *IEEE Transactions on Information Theory*, 40(4):1108–1121, 1994.

[90] Iwan Duursma and Seungkook Park. Coset bounds for algebraic geometric codes. *Finite Fields and Their Applications*, 16(1):36–55, 2010.

[91] Iwan Duursma and Jiashun Shen. Multiplicative secret sharing schemes from Reed-Muller type codes. In *ISIT*, pages 264–268. IEEE, 2012.

[92] W en Ai Jackson and Keith Martin. A combinatorial interpretation of ramp schemes. *Australasian Journal of Combinatorics*, 14:51–60, 1996.

[93] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 121–136. Springer, 1998.

[94] Matthias Fitzi and Ueli M. Maurer. Efficient byzantine agreement secure against general adversaries. In Shay Kutten, editor, *DISC*, volume 1499 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 1998.

[95] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *STOC*, pages 699–710. ACM, 1992.

[96] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *STOC*, pages 699–710. ACM, 1992.

[97] A. Garcia, H. Stichtenoth, and M. Thomas. On towers and composita of towers of function fields over finite fields. *Finite Fields and Their Applications*, 3:257–274, 1997.

[98] Arnaldo Garcia and Henning Stichtenoth. A tower of Artin-Schreier extensions of function fields attaining the Drinfeld-Vlăduţ bound. *Invent. Math.*, pages 211–222, 1995.

[99] Arnaldo Garcia and Henning Stichtenoth. On the asymptotic behavior of some towers of function fields over finite fields. *J. Number Theory*, 61:248–273, 1996.

[100] Arnaldo Garcia and Henning Stichtenoth, editors. *Topics in Geometry, Coding Theory and Cryptography.* Springer, 2007.

[101] Arnaldo Garcia, Henning Stichtenoth, Alp Bassa, and Peter Beelen. Towers of function fields over non-prime finite fields. Technical report, arXiv, 2012. See http://arxiv.org/abs/1202.5922.

[102] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[103] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, 25–27 May 1987.

[104] V. D. Goppa. Codes on algebraic curves. *Soviet Math. Dokl*, 24:170–172, 1981.

[105] Ron Graham, Martin Grötschel, and Laszlo Lovász, editors. *Handbook of Combinatorics.* The MIT Press, 1995.

[106] Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.

[107] Venkatesan Guruswami and Chaoping Xing. List decoding Reed-Solomon, algebraic-geometric, and Gabidulin subcodes up to the Singleton bound. In Boneh et al. [30], pages 843–852.

[108] G.H. Hardy and E. M. Wright. *An introduction to the theory of numbers.* Oxford University Press, 1979.

[109] Danny Harnik, Yuval Ishai, Eyal Kushilevitz, and Jesper Buus Nielsen. OT-combiners via secure computation. In Canetti [38], pages 393–411.

[110] Martin Hirt and Ueli Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, winter 2000.

[111] Dennis Hofheinz and Victor Shoup. Gnuc: A new universal composability framework. *IACR Cryptology ePrint Archive*, 2011:303, 2011.

[112] W. C. Huffman and V. Pless. *Fundamentals of Error Correcting Codes.* Cambridge University Press, 2003.

[113] Y. Ihara. Some remarks on the number of rational points of algebraic curves over finite fields. *J. Fac. Sci. Tokyo*, 3:721–724, 1981.

[114] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boheh, editor, *Advances in Cryptology - Crypto 2003*, pages 145–161, Berlin, 2003. Springer-Verlag. Lecture Notes in Computer Science Volume 2729.

[115] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and Jürg Wullschleger. Constant-rate oblivious transfer from noisy channels. In Rogaway [158], pages 667–684.

[116] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.

[117] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In Wagner [180], pages 572–591.

[118] M. Ito, A. Saito, and T. Nishizeki. Secret sharing schemes realizing general access structures. In *Proc. IEEE GlobeCom '87, Tokyo, Japan*, pages 99–102, 1987.

[119] David S. Johnson and Uriel Feige, editors. *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007.* ACM, 2007.

[120] Mauricio Karchmer and Avi Wigderson. On span programs. In *Structure in Complexity Theory Conference*, pages 102–111, 1993.

[121] Ehud D. Karnin, J. W. Greene, and Martin E. Hellman. On secret sharing systems. *IEEE Transactions on Information Theory*, 29(1):35–41, 1983.

[122] Joe Kilian. Founding cryptography on oblivious transfer. In ACM [1], pages 20–31.

[123] Ralph Kötter. A unified description of an error locating procedure for linear codes. In *Proc. Algebraic and Combinatorial Coding Theory, Voneshta Voda*, pages 113–117, 1992.

[124] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):381–401, July 1982.

[125] Serge Lang. *Algebraic Number Theory.* Springer, 1994.

[126] Serge Lang. *Algebra.* Graduate Texts in Mathematics. Springer, 2002.

[127] Serge Lang. *Undergraduate Algebra.* Springer, 2002.

[128] H. W. Lenstra, Jr. Euclidean number fields of large degree. *Invent. Math.*, 38:237–254, 1977.

[129] H.W. Lenstra, Jr. On a problem of Garcia, Stichtenoth, and Thomas. *Finite Fields and Their Applications*, 8:166–170, 2002.

[130] H.W. Lenstra, Jr. *Galois Theory for Schemes.* 2008. Available at `websites.math.leidenuniv.nl/algebra/`.

[131] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[132] Keith Martin. *Discrete structures in the theory of secret sharing.* PhD thesis, PhD thesis, University of London, 1991.

[133] Keith Martin. New secret sharing schemes from old. *J. Combin. Math. Combin. Comput.*, 14:65–77, 1993.

[134] Jim Massey. Minimal codewords and secret sharing. *Proceedings of the 6-th Joint Swedish-Russian Workshop on Information Theory, Molle, Sweden*, pages 269–279, 1993.

[135] Jim Massey. Some applications of coding theory in cryptography. *Codes and Ciphers: Cryptography and Coding IV*, pages 33–47, 1995.

[136] Ueli Maurer. Constructive cryptography - a new paradigm for security definitions and proofs. In Sebastian Mödersheim and Catuscia Palamidessi, editors, *TOSCA*, volume 6993 of *Lecture Notes in Computer Science*, pages 33–56. Springer, 2011.

[137] Robert J. McEliece and Dilip V. Sarwate. On sharing secrets and Reed-Solomon codes. *Commun. ACM*, 24(9):583–584, 1981.

[138] F. J. McWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977.

[139] Silvio Micali and Phillip Rogaway. Secure computation. In Joan Feigenbaum, editor, *Advances in Cryptology - Crypto '91*, pages 392–404, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science Volume 576.

[140] Diego Mirandola and Gilles Zémor. Schur products of linear codes: a study of parameters. *Diss. Master Thesis (under the supervision of G. Zémor), Univ. Bordeaux*, 2012.

[141] Carlos Moreno. *Algebraic Curves over Finite Fields*. Cambridge Tracts in Mathematics. Cambridge University Press, 1991.

[142] Moni Naor, editor. *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*. Springer, 2007.

[143] Jürgen Neukirch. *Algebraic Number Theory*. Graduate Texts in Mathematics. Springer, 1999. (German edition, 1992).

[144] Harald Niederreiter and Chaoping Xing. *Rational points on curves over finite fields*. Cambridge University Press, 2001.

[145] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2012.

[146] Francesco Noseda, Gilvan Oliveira, and Luciane Quoos. Bases for Riemann-Roch spaces of one-point divisors on an optimal tower of function fields. *IEEE Transactions on Information Theory*, 58(5):2589–2598, 2012.

[147] Wakaha Ogata and Kaoru Kurosawa. Some basic properties of general nonperfect secret sharing schemes. *J. UCS*, 4(8):690–704, 1998.

[148] Carles Padró. Lecture notes in secret sharing. *Eprint 2012/674, 46p. Available from eprint.iacr.org*, 2012.

[149] Pascal Paillier. Public-key cryptosystems based on composite degree residue classes. In Jacques Stern, editor, *Advances in Cryptology - EuroCrypt '99*, pages 223–238, Berlin, 1999. Springer-Verlag. Lecture Notes in Computer Science Volume 1592.

[150] Kenneth G. Paterson, editor. *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*. Springer, 2011.

[151] Ruud Pellikaan. On decoding by error location and dependent sets of error positions. *Discrete Mathematics*, 106-107:369–381, 1992.

[152] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Secure reactive systems. Technical Report RZ 3206, IBM Research, Zürich, May 2000.

[153] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 73–85, Seattle, Washington, 15–17 May 1989.

[154] Hugues Randriambololona. Hecke operators with odd determinant and binary frameproof codes beyond the probabilistic bound? *IEEE Information Theory Workshop (ITW 2010)*, pages 1–5, 2010.

[155] Hugues Randriambololona. Bilinear complexity of algebras and the Chudnovsky-Chudnovsky interpolation method. *J. Complexity*, 28(4):489–517, 2012.

[156] Hugues Randriambololona. Asymptotically good binary linear codes with asymptotically good self-intersection spans. *IEEE Transactions on Information Theory*, 59(5):3038–3045, 2013.

[157] Hugues Randriambololona. On products and powers of linear codes under componentwise multiplication. *arXiv preprint arXiv:1312.0022*, 2013.

[158] Phillip Rogaway, editor. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011.

[159] Michael Rosen. *Number Theory in Function Fields*. Graduate Texts in Mathematics. Springer, 2002.

[160] R. Rolland. S. Ballet. On the bilinear complexity of the multiplication in finite fields. In *Arithmetic, Geometry and Coding Theory (AGCT 2003), Séminaires et Congrès 11*, pages 179–188, 2005.

[161] Pierre Samuel. *Algebraic theory of numbers*. Hermann Paris, 1970.

[162] J.-P. Serre. Rational points on curves over finite fields. *Rational points on curves over finite fields. 1985, notes of lectures at Harvard University.*

[163] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[164] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005.

[165] Igor Shparlinski, Michael Tsfasman, and Serge Vlăduţ. Curves with many points and multiplication in finite fields. *Lecture Notes in Mathematics*, 1518:145–169, 1992.

[166] Kenneth W. Shum, Ilia Aleshnikov, P. Vijay Kumar, Henning Stichtenoth, and Vinay Deolalikar. A low-complexity algorithm for the construction of algebraic-geometric codes better than the Gilbert-Varshamov bound. *IEEE Transactions on Information Theory*, 47(6):2225–2241, 2001.

[167] G. Simmons, W.-A. Jackson, and K. Martin. The geometry of shared secret schemes. *Bull. Inst. Combin. Appl.*, 1:71–88, 1991.

[168] Janos Simon, editor. *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. ACM, 1988.

[169] Henning Stichtenoth. *Algebraic Function Fields and Codes*. Graduate Texts in Mathematics. Springer, second edition, 2008.

[170] Douglas R. Stinson. Decomposition constructions for secret-sharing schemes. *IEEE Transactions on Information Theory*, 40(1):118–125, 1994.

[171] Madhu Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *J. Complexity*, 13(1):180–193, 1997.

[172] M. Tsfasman, S. Vlăduţ, and Th. Zink. Modular curves, Shimura curves, and Goppa codes, better than Varshamov Gilbert bound. *Math. Nachr.*, pages 21–28, 1982.

[173] Michael Tsfasman, Serge Vlăduţ, and Dmitry Nogin. *Algebraic Geometric Codes: Basic Notions*, volume 139 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2007.

[174] Leslie G. Valiant. Short monotone formulae for the majority function. *J. Algorithms*, 5(3):363–366, 1984.

[175] Marten van Dijk. Secret key sharing and secret key generation. *PhD Thesis, Eindhoven University of Technology, The Netherlands*, 1997.

[176] J. H. van Lint. *Introduction to Coding Theory*. Graduate Texts in Mathematics. Springer, third edition, 1999.

[177] J. H. van Lint and R. M. Wilson. On the minimum distance of cyclic codes. *IEEE Transactions on Information Theory*, 32(1):23–40, 1986.

[178] J. H. van Lint and R. M. Wilson. *A Course in Combinatorics*. Cambridge University Press, 2nd edition, 2001.

[179] S. Vlăduţ. An exhaustion bound for algebraic-geometric modular code. *Probl. Inf. Transm*, 23:22–34, 1987.

[180] David Wagner, editor. *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*. Springer, 2008.

[181] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, 27–29 October 1986. IEEE.

[182] Th. Zink. Degeneration of Shimura surfaces and a problem in coding theory. In Lothar Budach, editor, *FCT*, volume 199 of *Lecture Notes in Computer Science*, pages 503–511. Springer, 1985.

Index